Форматы с плавающей и фиксированной точками

Ситкарев Г.А., <sitkarev@komitex.ru>

Сыктывкарский Государственный Университет
Лаборатория Прикладной Математики и Программирования
http://amplab.syktsu.ru

1.1. Позиционные системы счисления

Позиционные системы счисления используют число 0. Без нуля запись в позиционной системе счисления была бы невозможна:

$$(12450)_{10} = 1 \times 10^4 + 2 \times 10^3 + 4 \times 10^2 + 5 \times 10^1 + 0 \times 10^0.$$

1.2. Числа с фиксированной точкой

Рациональные числа могут быть представлены как отношение двух чисел: числителя и знаменателя. Такое представление не всегда удобно. Предположим, что компьютер умеет выполнять основные арифметические действия с целыми знаковыми числами. Пользуясь только целочисленной арифметикой, мы можем представить рациональные числа в специальном формате. Формат числа с фиксированной точкой состоит из трёх полей. Эти три поля соответственно хранят бит знака, биты целой части, и биты дробной части. Пусть для хранения числа с фиксированной точкой выделено 32 бита, один бит выделен для хранения знака, 15 бит отведено для целой части, и для дробной части выделены оставшиеся 16 бит. Приближённое значение числа 1/15 в таком представлении будет храниться как

0	000000000000000	0001000100010001

,

а число 11/2 тогда будет храниться как

0	000000000000101	1000000000000000

.

Арифметика с фиксированной точкой реализуется поверх целочисленной арифметики. Сложение, вычитание, умножение и деление чисел с фиксированной точкой выполняются на компьютере также, как с обычными целыми знаковыми числами, но в результат $\tilde{c}$ операций умножения (1) и деления (2) вносится поправка, с учётом масштабирующего коэффициента α.

$$\tilde{c} = \alpha a + \alpha b = \alpha(a + b), \;\; c = \tilde{c}$$

$$\tilde{c} = \alpha a \times \alpha b = \alpha^2(a \times b), \;\; c = \tilde{c}/\alpha \qquad (1)$$

$$\tilde{c} = \alpha a / \alpha b = a / b, \;\; c = \alpha\tilde{c} \qquad (2)$$

Умножение и деление на масштабирующий коэффициент α обычно осуществляют арифметическим битовым сдвигом влево и вправо. Для выбранного ранее формата

представления числа с фиксированной точкой $\alpha = 2^{16}$, а для коррекции результата умножения и деления число сдвигают соответственно на 16 бит вправо и 16 бит влево[1].

Минимальное положительное число, представимое в выбранном ранее формате с фиксированной точкой, это

$$1 \times 2^{-16} = 2^{-16} = 0{,}0000152587890625,$$

а максимальное положительное число

$$(2^{15} - 1) + (1 - 2^{-16}) = 32767{,}9999847412109375 \approx 32768.$$

Максимальное абсолютное значение отрицательного числа в таком формате

$$-2^{15} + (1 - 2^{-16}) = -32768{,}9999847412109375.$$

1.3. Числа с плавающей точкой

Представление чисел с плавающей точкой основано на научной записи чисел. Действительное число $x \neq 0$ в таком представлении записывается как

$$x = \pm S \times 10^E, \quad \text{где } 1 \leq S < 10,$$

и E – целое число. Обычно S называют *мантиссой числа*, а E называют *экспонентой числа*. Например, число 1284,45 записывают как

$$1{,}28445 \times 10^3,$$

а число 0,000325 как

$$3{,}25 \times 10^{-4}.$$

На компьютере числа с плавающей точкой удобнее представлять в двоичном виде, поэтому число x представляют так:

$$x = \pm S \times 2^E, \quad \text{где } 1 \leq S < 2.$$

В двоичном виде S это

$$S = (b_0,b_1b_2b_3\ldots)_2, \quad \text{где } b_0 = 1.$$

Так как $b_0 = 1$, мы можем записать S как

$$S = (1,b_1b_2b_3\ldots)_2.$$

Такое представление называют *нормализованным представлением* числа x.

Для хранения числа в таком формате нужны поля для хранения знака, экспоненты E и мантиссы S.

±	E	S

Пусть у нас есть 32 бита для хранения этих полей. Мы выделим один бит для хранения знака: если бит знака будет 0, то это положительное число, иначе, если бит знака 1, то число отрицательное. Для хранения экспоненты E мы выделим 8 бит. Теоретически, мы бы могли хранить в этом поле числа от −128 до +127 в двоичном коде с дополнением до двух. Стандарт IEEE 754 использует несколько иной формат для хранения экспоненты, и мы пока будем предполагать, что в этих битах может храниться экспонента со значениями от −126 до +127. Под хранение мантиссы мы выделяем оставшиеся 23 бита, в которых

[1] Здесь следовало бы ещё произвести и округление, но мы опустим этот шаг для простоты изложения.

будут храниться биты $b_1,b_2,b_3,\ldots,b_{23}$. Бит b_0 в явном виде в S храниться не будет, так как предполагается, что $b_0 = 1$. Число x, в точности представимое в таком виде, называется *числом с плавающей точкой*. Если число x точно в таком виде представить нельзя, тогда его придётся округлять.

Число 11/2 в таком формате будет храниться как

0	ebits(2)	01100000000000000000000

а число

$$8234 = (1{,}0000000101010)_2 \times 2^{13}$$

будет храниться как

0	ebits(13)	00000001010100000000000

Здесь «*ebits*(E)» обозначает битовое представление экспоненты E, которое мы пока не показываем. Так как мантисса хранит только дробную часть, мы должны мысленно представлять единичный бит b_0, который всегда присутствует, но не хранится в мантиссе. Если число с плавающей точкой есть в точности степень двойки, то мантисса будет состоять полностью из нулей. Например, число

$$1 = (1{,}00000\ldots0)_2 \times 2^0$$

будет храниться как

0	ebits(0)	00000000000000000000000

1.4. Точность и машинное эпсилон

Точностью представления числа с плавающей точкой называют количество бит, выделенных для хранения мантиссы, включая скрытый бит b_0. Любое число с плавающей точкой в нормализованном виде может быть представлено как

$$x = \pm(1{,}b_1b_2b_3\ldots b_{p-2}b_{p-1})_2 \times 2^E. \tag{3}$$

Для выбранного ранее формата представления чисел с плавающей точкой, $p = 24$ (23 бита хранящихся явно и один скрытый бит $b_0 = 1$). Ближайшее число с плавающей точкой, которое будет больше 1, это

$$(1{,}000\ldots001)_2 \times 2^0 = 1 + 2^{-(p-1)}.$$

Промежуток от числа 1 до ближайшего большего числа с плавающей точкой называют *машинным эпсилон*:

$$\varepsilon = 1 - 1 + 2^{-(p-1)} = 2^{-(p-1)}.$$

Для любого числа с плавающей точкой x в формате (3) также определяют

$$\mathrm{ulp}(x) = (0{,}000\ldots001)_2 \times 2^E = 2^{-(p-1)} \times 2^E = \varepsilon \times 2^E.$$

Если $x > 0$, тогда $\mathrm{ulp}(x)$ — промежуток от x до ближайшего большего числа, если $x < 0$, тогда $\mathrm{ulp}(x)$ — промежуток от x до ближайшего меньшего числа. Ulp это сокращение от *units in the last place*.

1.5. Представление нуля

Число 0 нормализовать не получится, поэтому представить его в виде (3) не удастся. Число, где $b_1 = b_2 = \ldots = b_{p-1} = 0$ это 1, а не 0. Примерно до 1975-го года все форматы

чисел с плавающей точкой хранили бит b_0 явно. Тогда, задав все биты $b_0 = b_1 = \ldots = b_{p-1} = 0$, получался 0. При этом приходилось жертвовать одним битом точности. В IEEE 754 используется другой подход: для представления нуля резервируется специальное значение поля экспоненты.

1.6. Игрушечная система с плавающей точкой

Наша цель получить представление о том, что из себя представляют числа с плавающей точкой. С этой целью мы спроектируем игрушечную систему с плавающей точкой. Практическая ценность такой системы умозрительна, и для реальных расчётов она вряд ли подойдёт, зато с её помощью мы на уровне интуиции почувствуем, как работает арифметика с плавающей точкой. Числа в нашей игрушечной системе будут представлены как

$$\pm(b_0,b_1 b_2)_2 \times 2^E,$$

где b_0 хранится непосредственно, все числа имеют нормализованное представление, а экспонента E принимает значения −1, 0 и 1. Все возможные значения мантиссы в такой системе:

$$(1{,}00)_2 = 1{,}0,$$
$$(1{,}01)_2 = 1{,}25,$$
$$(1{,}10)_2 = 1{,}50,$$
$$(1{,}11)_2 = 1{,}75.$$

Точность такой системы будет $p = 3$, а максимальное $N_{\max}$ и минимальное $N_{\min}$ положительные числа, представимые в такой системе, тогда

$$N_{\max} = (1{,}11)_2 \times 2^1 = 3{,}5 \quad \text{и} \quad N_{\min} = (1.\,00)_2 \times 2^{-1} = 0{,}5.$$

Машинное эпсилон в нашей игрушечной системе $\varepsilon = 1{,}25 - 1{,}0 = 0{,}25$.

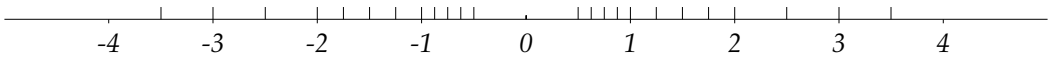

Рис. 1. Игрушечная система с плавающей точкой.

На рис. 1 отмечено расположение чисел с плавающей точкой игрушечной системы на числовой оси. Видно, что числа на оси располагаются неравномерно, и чем больше становится экспонента, тем больше промежуток до ближайшего большего числа — этот промежуток и есть ulp.

1.7. Форматы IEEE 754 одинарной и двойной точности

Стандарт IEEE 754 (полное название ANSI/IEEE Std 754-1985) был принят в 1985 г. Как международный стандарт он был принят в 1989 г., и получил первоначально название IEC 559. Позже международное обозначение стандарта обновилось до IEC 60559. Главные введения стандарта:

- совместимое представление чисел с плавающей точкой на машинах разной архитектуры;

- корректно округляемые операции (+,–,/,×,√);
- весьма логичный и целостный, с математической точки зрения, подход к обработке исключительных ситуаций, таких как «деление на ноль» и прочих.

Стандарт вводит два специальных числа 0 и ∞, оба имеют знак ±0, ±∞. Специальная последовательность битов NaN (Not-a-Number) предназначается для сигнализации об ошибке.

1.7.1. Формат IEEE одинарной точности

Числа с плавающей точкой одинарной точности IEEE хранятся в 32-битовом слове, так как показано на рис. 2. Экспонента хранится как обычное беззнаковое целое со смещением 127. Таким образом, для формата IEEE одинарной точности

$$\text{ebits}(E) = E + 127.$$

Максимальное и минимальное значения экспоненты одинарного формата IEEE соответственно $E_{\min} = -126$ и $E_{\max} = 127$. Порядок полей числа выбран не случайно: их расположение позволяет сравнивать числа IEEE одинарной и двойной точностей как обычные 32/64-битные знаковые целые числа. Точность формата $p = 24$, минимально представимое положительное нормализованное число

0	00000001	00000000000000000000000

$$N_{\min} = 1 \times 2^{-126} \approx 1{,}2 \times 10^{-38},$$

и максимально представимое положительное нормализованное число

0	11111110	11111111111111111111111

$$N_{\max} = \left(2 - 2 \times 2^{-(p-1)}\right) \times 2^{127} \approx 2^{128} \approx 3{,}4 \times 10^{38}.$$

В табл. 1 представление формата IEEE одинарной точности приводится обобщённо.

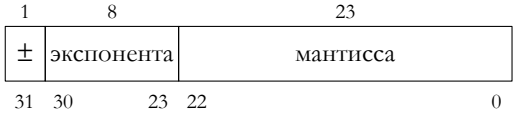

Рис. 2. Формат IEEE одинарной точности.

1.7.2. Формат IEEE двойной точности

Числа с плавающей точкой одинарной точности IEEE хранятся в 64-х битовом слове, как показано на рис. 3. Экспонента хранится как обычное беззнаковое целое со смещением 1023. Таким образом, для формата IEEE двойной точности

$$\text{ebits}(E) = E + 1023.$$

Максимальное и минимальное значения экспоненты формата двойной точности IEEE соответственно $E_{\min} = -1022$ и $E_{\max} = 1023$. Точность формата $p = 53$, минимально представимое положительное нормализованное число

$$N_{\min} = 2^{-1022} \approx 2{,}2 \times 10^{-308},$$

и максимально представимое положительное нормализованное число

±	$a_1 a_2 \ldots a_8$	$b_1 b_2 b_3 \ldots b_{23}$

Если биты экспоненты $a_1, \ldots, a_8$ это	тогда значение будет
$(00000000)_2 = (0)_{10}$	$(0, b_1 b_2 b_3 \ldots b_{23)} \times 2^{-126}$
$(00000001)_2 = (1)_{10}$	$(1, b_1 b_2 b_3 \ldots b_{23)} \times 2^{-126}$
$(00000010)_2 = (2)_{10}$	$(1, b_1 b_2 b_3 \ldots b_{23)} \times 2^{-125}$
$\downarrow$	$\downarrow$
$(01111111)_2 = (127)_{10}$	$(1, b_1 b_2 b_3 \ldots b_{23)} \times 2^{0}$
$(10000000)_2 = (128)_{10}$	$(1, b_1 b_2 b_3 \ldots b_{23)} \times 2^{1}$
$\downarrow$	$\downarrow$
$(11111100)_2 = (252)_{10}$	$(1, b_1 b_2 b_3 \ldots b_{23)} \times 2^{124}$
$(11111111)_2 = (253)_{10}$	$(1, b_1 b_2 b_3 \ldots b_{23)} \times 2^{125}$
$(11111110)_2 = (254)_{10}$	$(1, b_1 b_2 b_3 \ldots b_{23)} \times 2^{126}$
$(11111111)_2 = (255)_{10}$	если $b_1 = b_1 = \ldots = b_{23} = 0$ то 0, иначе NaN.

Табл. 1.

$$N_{\max} = \left(2 - 2^{-52}\right) \times 2^{1023} \approx 2^{1024} \approx 1.8 \times 10^{308}.$$

В табл. 2 представление формата IEEE двойной точности приводится обобщённо.

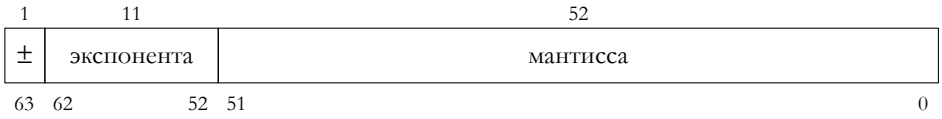

Рис. 3. Формат IEEE двойной точности.

1.8. Какой формат выбрать?

Стандарт IEEE выдвигает требования к обязательной реализации только формата одинарной точности. Формат двойной точности является опциональным, хотя все известные архитектуры, заявляющие совместимость с IEEE, реализуют поддержку, как минимум, и двойной и одинарной точностей. Часто одинарной точности для удовлетворительного результата вычислений недостаточно. Потому рекомендуется всегда использовать формат двойной точности. Формат одинарной точности хорошо подходит для экономичного хранения большого количества чисел с плавающей точкой.

1.9. Формат расширенной точности

Стандарт IEEE настоятельно рекомендует к реализации формат расширенной точности, где для хранения экспоненты выделено, как минимум, 15 бит, а для хранения дробной части мантиссы отведено, как минимум, 63 бита. Некоторые архитектуры имеют поддержку такого формата, но детали реализации отличаются от производителя к производителю. Так, например, микропроцессоры Intel поддерживают формат расширенной точности с 1 битом знака, 15-битной экспонентой и 64-битной мантиссой, и хранят их в регистрах шириной 80 бит. Микропроцессоры Sun SPARC реализуют поддержку формата расширенной точности программно, а для хранения таких чисел задействуется 128 бит.

±	$a_1 a_2 \ldots a_{11}$	$b_1 b_2 b_3 \ldots b_{52}$

Если биты экспоненты $a_1,\ldots,a_{11}$ это	тогда значение будет
$(00000000000)_2 = (0)_{10}$	$(0,b_1 b_2 b_3 \ldots b_{52)} \times 2^{-1022}$
$(00000000001)_2 = (1)_{10}$ $(00000000010)_2 = (2)_{10}$ ↓ $(01111111111)_2 = (1023)_{10}$ $(10000000000)_2 = (1024)_{10}$ ↓ $(11111111100)_2 = (2044)_{10}$ $(11111111101)_2 = (2045)_{10}$ $(11111111110)_2 = (2046)_{10}$	$(1,b_1 b_2 b_3 \ldots b_{52)} \times 2^{-1022}$ $(1,b_1 b_2 b_3 \ldots b_{52)} \times 2^{-1021}$ ↓ $(1,b_1 b_2 b_3 \ldots b_{52)} \times 2^{0}$ $(1,b_1 b_2 b_3 \ldots b_{52)} \times 2^{1}$ ↓ $(1,b_1 b_2 b_3 \ldots b_{52)} \times 2^{1021}$ $(1,b_1 b_2 b_3 \ldots b_{52)} \times 2^{1022}$ $(1,b_1 b_2 b_3 \ldots b_{52)} \times 2^{1023}$
$(11111111111)_2 = (2047)_{10}$	если $b_1 = b_1 = \ldots = b_{52} = 0$ то 0, иначе NaN.

Табл. 2.

1.10. Денормализованные числа

Если обратить внимание на первые строки табл. 1 и табл. 2., то можно увидеть, что для представления нуля в форматах IEEE зарезервирована специальная битовая комбинация поля экспоненты, состоящая только из нулевых битов. Если задействовать биты дробной части мантиссы $b_1,b_2,\ldots,b_{p-1}$, положив при этом экспоненту $E = E_{\min}$, то мы обретём возможность представить число $x < N_{\min}$, как

$$x = (0,b_1 b_2 \ldots b_{p-1})_2 \times 2^{E_{\min}}.$$

Этот подход лучше всего продемонстрировать на примере нашей игрушечной системы с плавающей точкой. Для этого найдём все положительные денормализованные значения, полученные как

$$(0,b_1 b_2)_2 \times 2^{-1},$$

для всех возможных комбинаций дробной части $(0,b_1 b_2)_2$:

$$(0,01)_2 \times 2^{-1} = 0,125,$$
$$(0,10)_2 \times 2^{-1} = 0,25,$$
$$(0,11)_2 \times 2^{-1} = 0,375.$$

На рис. 2 эти положительные числа, а также отрицательные денормализованные, нанесены на числовую ось, вместе с нормализованными числами с плавающей точкой. Как видно, денормализованные числа равномерно заполняют промежуток от нуля до $\pm N_{\min}$. Польза от введения денормализованных чисел заключается в том, что они гарантируют ненулевую разность для двух положительных или отрицательных нормализованных чисел, не равных друг другу, как близко к нулю они бы не располагались. Например, пусть в игрушечной системе с плавающей точкой нет денормализованных чисел. Если взять два достаточно малых числа

$$x = (1.01)_2 \times 2^{-1}$$

и

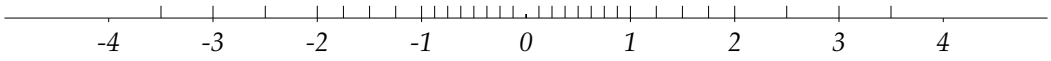

Рис. 2.

$$y = (1.00)_2 \times 2^{-1},$$

то результат их разности

$$x - y = \left((1.01)_2 - (1.00)_2\right) \times 2^{-1} = (0.01)_2 \times 2^{-1} = (0.125)_{10}$$

не имеет представления, как число с плавающей точкой, и мы должны округлить его в 0 или в $N_{\min}$. Такой результат противоречит алгебраическому правилу для любых $x \neq y$, что

$$x - y = 0$$

только в том случае, если $x \equiv y$. Если мы введём денормализованные числа, то результат разности $x - y$ всегда представим для любых нормализованных x и $y0$, а их разность $x - y$ будет равняться нулю только в том случае, если $x \equiv y$. Мотивация для введения такого усложнения теперь очевидна: система с плавающей точкой при введении денормализованных значений обретает алгебраическую целостность.

Когда стандарт IEEE ещё находился в стадии согласования и обсуждения, нововведения касательно денормализованных чисел вызвали оживлённую дискуссию. Тогда произошло разделение как бы на два лагеря — сторонников введения денормализованных чисел в IEEE, и противников этого. Дело в том, что аппаратная реализация этой части спецификации усложняет схему микропроцессора. Некоторые известные реализации эмулируют арифметику с денормализованными числами на микропрограммном уровне, а часть эмулирует вообще чисто программно. Вне сомнений, денормализованные числа весьма полезны, и любая реализация IEEE должна их поддерживать.

1.11. Значащие разряды

Сколько десятичных разрядов содержат в себе числа IEEE одинарной и двойной точностей? Ответ на этот вопрос можно дать только приблизительно. Так как для чисел с одинарной точностью $p = 24$, то

$$2^{-p} = 2^{-24} \approx 10^{-7},$$

а значит десятичных разрядов в них

$$-\log_{10} 2^{-24} \approx 7.$$

Для чисел с двойной точностью $p = 53$, тогда

$$2^{-p} = 2^{-53} \approx 10^{-16},$$

а значит десятичных разрядов содержится в них

$$-\log_{10} 2^{-53} \approx 16.$$

Это не более, чем приблизительная оценка.

1.12. Порядок байт

Все современные компьютеры адресуют память побайтно. Это значит, что любую ячейку памяти можно адресовать или обратиться к ней непосредственно. Так, например, 32-битное слово состоит из четырёх байт, обозначим их как $B_0 B_1 B_2 B_3$. Адрес байта B_3 это адрес байта $B_0 + 3$. Теперь положим, что мы храним число в формате IEEE одинарной точности в таком 32-битном слове. Биты числа IEEE будут храниться там, как

$$\sigma a_1 a_2 \dots a_8 b_1 b_2 b_3 \dots b_{23},$$

где σ — бит знака. Первые 8 бит этого числа

$$\sigma a_1 a_2 a_3 a_4 a_5 a_6 a_7$$

составляют один байт. Где хранится этот байт — на месте B_0 или на месте B_3? Ответ на этот вопрос зависит от того, каков порядок байт принят для данной архитектуры компьютера. Так, например, на x86 этот байт будет храниться в B_0, а на SPARC в B_3. Порядок байт в числах IEEE обычно совпадает с порядком байт, принятым на компьютере. При обмене числами в формате IEEE, например, по сети, следует учитывать, что порядок байт на передающей и принимающей сторонах совсем не обязательно совпадает.

1.13. Округление

Напомним, числа в формате IEEE представляются в виде

$$\pm(b_0,b_1 b_2 \dots b_{p-1})_2 \times 2^E, \tag{4}$$

где p — точность; для нормализованных чисел — $b_0 = 1$, $E_{\min} \le E \le E_{\max}$; для денормализованных — $b_0 = 0$, $E = E_{\min}$. Мы будем говорить, что число x находится в нормализованном диапазоне, если

$$E_{\min} \le |x| \le E_{\max}.$$

Числа ± 0 и $\pm\infty$, хотя и являются числами IEEE, не находятся в нормализованном диапазоне.

Пусть x не является числом с плавающей точкой и не представимо точно в виде (4). Тогда возможны два варианта:

- x выходит за нормализованный диапазон;
- для точного представления x не хватает p бит, например, число

  $$1 + 2^{-25} = (1,00000000000000000000000001)_2$$

  нельзя точно представить в формате IEEE одинарной точности.

И в том и в другом случаях нужно взять аппроксимацию x. Введём два обозначения x_- и x_+ для чисел, связанных с x так, что

- x_- будет ближайшим к x числом с плавающей точкой, для которого выполняется $x_- \le x$;
- x_+ будет ближайшим к x числом с плавающей точкой, для которого выполняется $x_+ \ge x$.

Пусть x находится в нормализованном диапазоне. Запишем его тогда как

$$x = (1,b_1 b_2 b_3 \dots b_{p-2} b_{p-1} b_p b_{p+1} \dots)_2 \times 2^E.$$

Отбрасывая все биты после b_{p-1}, получим

$$x_- = (1,b_1 b_2 b_3 \dots b_{p-2} b_{p-1})_2 \times 2^E.$$

Тогда

$$x_+ = \left((1,b_1 b_2 b_3 \ldots b_{p-2} b_{p-1})_2 + (0,00000 \ldots 01)_2 \right) \times 2^E,$$

где во втором члене суммы единичный бит установлен на месте b_{p-1}. Заметим, что

$$x_+ = x_- + \text{ulp}(x_-).$$

С битовым представлением x_+ не всё так просто. При сложении, если $b_1 = b_2 = \ldots = b_{p-2} = b_{p-1} = 1$, может возникнуть перенос в целый разряд, что должно вызвать увеличение экспоненты на единицу. Сказать точно, какое битовое представление x_+ будет получено в результате сложения, можно только в случае, если известны все биты x от b_1 до b_{p-1} включительно.

Рассмотрим теперь случай, когда x не входит в нормализованный диапазон. Если $x > N_{\max}$, тогда

$$x_- = N_{\max};$$
$$x_+ = \infty.$$

Если $x < N_{\min}$, тогда

$$x_- = 0 \text{ или } x_- \text{ денормализованное;}$$
$$x_+ = N_{\min} \text{ или } x_+ \text{ денормализованное.}$$

1.13.1. Корректно округлённые значения

Стандарт IEEE определяет «корректно округлённое значение» x, которое мы будем обозначать как round(x). Если x — число с плавающей точкой, то x = round(x). В противном случае значение зависит от режима округления:

- округление вниз (или к $-\infty$)

  $$\text{round}(x) = x_-.$$

- округление вверх (или к $+\infty$)

  $$\text{round}(x) = x_+.$$

- округление к нулю

  $$\text{round(x)} = x_-, \text{ если } x > 0;$$
  $$\text{round(x)} = x_+, \text{ если } x < 0.$$

- округление к ближайшему

  round(x) будет или x_+ или x_-, в зависимости от того, какое из них окажется ближе к x. Если x лежит точно посередине между x_- и x_+, то выбирается чётное (то, у которого $b_{p-1} = 0$).

  В правиле округления до ближайшего есть одно единственное исключение. Когда $x > N_{\max}$, то round(x) = $N_{\max}$ если $x < N_{\max} + \text{ulp}(N_{\max})/2$, иначе round($x$) = ∞.

1.13.2. Абсолютная и относительная ошибки округления

Определим ошибку округления. Пусть x это число, тогда

$$\text{abserr}(x) = |\text{round}(x) - x|.$$

Применительно к IEEE, если

$$x = \pm(1,b_1 b_2 b_3 \ldots b_{p-2} b_{p-1} b_p b_{p+1} \ldots)_2 \times 2^E,$$

и x находится в диапазоне нормализованных значений, тогда абсолютная ошибка округления abserr(x) будет меньше, чем расстояние между x_- и x_+, вне зависимости от режима округления, а значит справедливо и

$$\text{abserr}(x) = |\text{round}(x) - x| < 2^{-(p-1)} \times 2^E.$$

Неформально, абсолютная ошибка округления, вне зависимости от выбранного режима округления, будет меньше ulp(x). Когда выбран режим округления до ближайшего, мы можем сказать, что абсолютная ошибка округления будет меньше или равна половине расстояния от x_- до x_+:

$$\text{abserr}(x) = |\text{round}(x) - x| \le \frac{2^{-(p-1)} \times 2^E}{2} = 2^{-p} \times 2^E.$$

Относительная ошибка округления для числа x определяется как

$$\text{relerr}(x) = |\delta|,$$

где

$$\delta = \frac{\text{round}(x) - x}{x}.$$

Если x находится в нормализованном диапазоне и не является числом с плавающей точкой, то можно считать

$$|x| > 2^E.$$

Тогда для всех режимов округления

$$\text{relerr}(x) = |\delta| = \left|\frac{\text{round}(x) - x}{x}\right| < \frac{2^{-(p-1)} \times 2^E}{2^E} = 2^{-(p-1)} = \varepsilon,$$

и для режима округления до ближайшего

$$\text{relerr}(x) = |\delta| = \left|\frac{\text{round}(x) - x}{x}\right| < \frac{2^{-p} \times 2^E}{2^E} = 2^{-p} = \frac{\varepsilon}{2}.$$

Таким образом, любое округлённое число x можно представлять как

$$\text{round}(x) = x(1 + \delta).$$

Отсюда следует очень важный вывод. Не важно, как мы храним и представляем x, оно будет точным, как по фактору $(1 + \delta)$.

1.14. Корректно округляемые операции с плавающей точкой

Стандарт IEEE вводит обязательные к реализации:

- правильно округляемые операции (+,–,×,/);
- правильно округляемые $\sqrt{}$ и *mod* (остаток от деления);
- правильно округляемые преобразования форматов.

1.14.1. Корректно округляемая арифметика

Результат операции с двумя числами с плавающей точкой может и не являться числом с плавающей точкой. Например, 1 и 2^{-24} — числа с плавающей точкой, а их сумма в

формате одинарной точности — нет.

Пусть x и y числа с плавающей точкой, а $+,-,\times,/$ операции, и пусть $\oplus,\ominus,\otimes,\oslash$ эти же операции, но обозначающие их реализацию на компьютере. Тогда $x + y$ может не быть числом с плавающей точкой, но $x \oplus y$ будет таковым. До появления IEEE результаты этих операций могли различаться от компьютера к компьютеру. Стандарт IEEE устанавливает следующие правила для арифметических операций:

$$x \oplus y = \text{round}(x + y);$$
$$x \ominus y = \text{round}(x - y);$$
$$x \otimes y = \text{round}(x \times y);$$
$$x \oslash y = \text{round}(x / y).$$

Последовательность из нескольких арифметических операций может и не давать корректно округлённого результата. Предположим, $x = z = 1$, а $y = 2^{-25}$. Если эти числа представлены в формате IEEE одинарной точности, то

$$x + y = (1{,}00000000000000000000000001)_2,$$

в то время как

$$x \oplus y = 1.$$

Поэтому

$$(x \oplus y) \ominus z = 0,$$

но точный результат

$$(x + y) - z = 2^{-25}. \tag{5}$$

Отсюда следует важный вывод: операции арифметики IEEE не обладают свойством коммутативности. Порядок выполнения операций в некоторых случаях может оказаться важным. Изменив порядок суммирования в (5), мы бы получили точный результат

$$(x \ominus z) \oplus y = 2^{-25},$$

так как

$$x \ominus z = 0.$$

1.15. Исключения

Обработка исключительных ситуаций, как правило, содержится в каждой программе и является её наиболее сложной частью. Типичная исключительная ситуация при счёте на компьютере это «деление на ноль». До появления IEEE, программисту было сложно: обработка и сигнализация об исключительных ситуациях на разных компьютерах осуществлялась не одинаково. Числа хранились в форматах, не совпадающих друг с другом, обработка исключений осуществлялась по-разному. Исключения, как это предложено в IEEE, существенно облегчают написание переносимых программ и упрощают алгоритмы.

Следует помнить, что числа IEEE одинарной и двойной точности могут хранить специальные значения:

- бесконечность (∞), которое всегда больше любого числа;
- минус бесконечность ($-\infty$), которое меньше любого числа;

- плюс ноль (+0);
- минус ноль (−0);
- не число (NaN), обозначающее неверное значение, возникающее при операции не имеющей математического смысла, например, при делении нуля на ноль.

Соглашения, принятые в IEEE, существенно облегчают жизнь программиста в части обработки исключительных ситуаций. Например, исключения при делении на ноль фиксируются, но могут быть и попросту игнорированы. Поведение IEEE арифметики при делении на ноль весьма упрощает программирование. Так, для любого $a > 0$, справедливо:

$$+a/+0 = +\infty,$$
$$-a/+0 = -\infty.$$

Такая особенность IEEE арифметики освобождает программиста от необходимости проверять делитель на равенство или близость к 0. Предположим, нужно посчитать значение a по формуле

$$a = \frac{1}{\frac{1}{b} + \frac{1}{c}}.$$

До появления IEEE, на множестве компьютеров программист должен был проверить условие на равенство $b = 0$ и $c = 0$, иначе программа могла аварийно завершиться с ошибкой «деление на ноль». В арифметике IEEE, в том случае если $b = 0$ или $c = 0$, будет получен ожидаемый результат $a = 0$. Для любого $a > 0$, следующие правила при делении на бесконечность будут верны:

$$+a/(+\infty) = +0,$$
$$-a/(+\infty) = -0,$$
$$+a/(-\infty) = -0,$$
$$-a/(-\infty) = +0.$$

Результат сравнения 0 и −0 в IEEE всегда истина. В связи с этим, в IEEE арифметике существует следующий феномен:

$$0 = -0 \text{ и } a = b; \text{ но } 1/a \neq 1/b.$$

Прочие действия с бесконечностью в IEEE арифметике ведут себя логично и целостно с математической точки зрения. Так, для любого $a > 0$, справедливо:

$$\infty + \infty = +\infty,$$
$$\infty - \infty = \text{NaN},$$
$$\infty \times \infty = \infty,$$
$$\infty/\infty = \text{NaN},$$
$$\infty/a = \infty,$$
$$\infty/0 = \infty,$$
$$0/0 = \text{NaN}.$$

Операции сравнения с бесконечностью ведут себя, как и ожидается:

1. Все конечные числа меньше $+\infty$.
2. Все конечные числа больше $-\infty$.
3. $-\infty$ меньше, чем $+\infty$.

Выражения с NaN имеют следующие свойства:

1. Любое арифметическое выражение, где задействован NaN, в результате даёт NaN.
2. Любая операция сравнения, где задействован NaN, даёт ложь.

Такие свойства IEEE способствует удобству при обработке исключительных ситуаций, и позволяют избежать дополнительных проверок на специальные значения. Предположим, у нас есть следующий алгоритм:

$a = f(x)$
если (a > 0) **тогда**
выполнить что-то

Возможно, что функция $f(x)$ вернёт NaN или ∞. Но условие **если** будет ложно для $a = -\infty$ и $a = \text{NaN}$, в то время как для $a = +\infty$ оно будет истинным. Значит дополнительных проверок на $a = \text{NaN}$ и $a = -\infty$ добавлять в программу не нужно. Код выглядит проще и понятнее.

1.15.1. Исключения IEEE

Стандарт IEEE определяет пять исключений:

Деление на ноль (divide-by-zero)
: возникает тогда, когда операция над конечным числом даёт в результате $\pm\infty$.

Переполнение (overflow)
: возникает тогда, когда результат операции не вмещается в конечное представление ($|x| > N_{\max}$ или $|x| < N_{\min}$).

Потеря значимости (underflow)
: возникает тогда, когда результат операции не вмещается в нормализованное представление с плавающей точкой и происходит денормализация, а значит и потеря значащих разрядов.

Неточное значение (inexact)
: возникает тогда, когда результат операции не является числом с плавающей точкой и был округлён.

Неверная операция (invalid)
: возникает тогда, когда операция не определена или не имеет смысла ($0/0$, $\infty - \infty$ или $\sqrt{-1}$).

В табл. 3 приведены исключения и действия по умолчанию для них.

Исключение	Действие или результат
Неверная операция	NaN
Деление на ноль	$\pm\infty$
Переполнение	$\pm\infty$ или $\pm N_{\max}$
Потеря значимости	денормализация или ± 0
Неточное значение	установить в корректно округлённое

Табл. 3.

Свойства IEEE арифметики способствуют следующему подходу к программированию вычислительных алгоритмов: сначала вычисления выполняются самым простым

способом «в лоб», и если только были обнаружены исключительные ситуации, тогда они обрабатываются отдельно, вне основного потока программы. Этот подход можно продемонстрировать на примере вычисления

$$\sqrt{a^2 + b^2}.$$

Даже в том случае, когда результат находится в нормализованном диапазоне, операция возведения в квадрат может вызвать переполнение. Если исключений, как это предложено в IEEE, нет, то программисту пришлось бы выполнять проверку значений a и b, масштабировать их на $\max(|a|,|b|)$, до того, как они возводятся в квадрат. В случае с IEEE, вычисление можно сразу выполнять напрямую, предварительно сбросив флаги исключений. Затем флаги исключений проверяют, и если для переполнения или потери значимости они окажутся установлены, выполняют соответствующие действия.