Системы счисления и действия в них

По материалам курса «Информатика»
Кафедры ЮНЕСКО по НИТ

Цель: рассмотреть

- основные понятия числовых систем;
- правила построения систем;
- выполнение действий в системах счисления.

Система *счисления*

Алфавит *X* из *p* символов и правила записи (изображения) и обработки чисел с помощью символов этого алфавита называются системой *счисления* (нумерацией) с основанием *p*. Число *x* в *системе* с основанием *p* обозначается как $(x)_p$ или x_p.

Система счисления

- Любая *система счисления* – это *система* кодирования числовых величин (количеств), позволяющая выполнять операции кодирования и декодирования, то есть по любой количественной величине однозначно находить его кодовое представление и по любой кодовой записи – восстанавливать соответствующую ей числовую величину.
- Наиболее используемые в информатике *системы счисления*:
  - двоичная, над алфавитом X = {0,1};
  - восьмеричная, над X = {0, 1, 2, 3, 4, 5, 6, 7};
  - шестнадцатеричная, над X = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}, где символы A, B, C, D, E, F имеют десятичные веса 10, 11, 12, 13, 14, 15.

Основные понятия *кодирования* и *шифрования*

- Все системы счисления строятся по общему принципу: определяется величина *p* – *основание* системы, а любое число *x* записывается в виде комбинации степеней веса *p* от *0*-й до *n*-й степени следующим образом:
- $(x)_{10} = x_n p^n + x_{n-1} p^{n-1} + \ldots + x_1 p^1 + x_0 p^0$, где $(n+1)$ – количество разрядов в числе x

Пример.

$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 4 + 1 = 13_{10}$,

$157_8 = 1 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 = 64 + 40 + 7 = 111_{10}$,

$A6F_{16} = 10 \times 256 + 6 \times 16 + 15 \times 1 = 2671_{10}$.

$110{,}001_2 = 1x2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 6{,}125_{10}$;

$A{,}B_{16} = A \times 16^0 + B \times 16^{-1} = 10 \times 1 + 11 \times 0{,}0625 = 10{,}6875_{10}$.

Позиционные и непозиционные системы счисления

- *Система счисления* в которой вес цифры (или символа алфавита) зависит от ее места в записи числа или слова называется *позиционной*; в противном случае *система* называется *непозиционной*.

Непозиционная система – древняя римская *система* записи чисел с алфавитом вида X={I (1), V (5), X (10), L (50), C (100), D (500), M (1000)}.

Примеры римских чисел: III (3), IV (4), V (5), VI (6), IX (9), XI (11), DCL (650).

Запись числа в этой *системе* получается двусторонней конкатенацией, причем правая конкатенация ассоциируется с добавлением, а левая конкатенация – с убавлением (например, IV и VI).

Процедура перевода десятичных чисел в *p*-ную систему счисления:

- перевести отдельно целую часть числа x:
  - последовательно делить целую часть $[x]_{10}$, а затем все частные (получаемые при делении) на *p* до тех пор, пока не получим в очередном частном число меньшее p; результат получается последовательным приписыванием к последнему частному остатков от деления – от последнего до первого;
- перевести отдельно дробную часть (мантиссу) числа:
  - последовательно умножать исходную мантиссу, а затем мантиссы получаемых чисел на *p* до тех пор, пока не получим мантиссу, равную нулю, или нужное количество цифр в $\{x\}_p$; результат получается приписыванием к целой части первого произведения второй такой же цифры и т.д., до последней цифры целой части;
- итоговый результат будет иметь вид (x)p = [x]p, {x}p .

Пример: найти: $12{,}8_{10} = ?_2$

- Переводим целую часть: $12_{10} = 1100_2$;
- переводим дробную часть (выделены цифры, идущие в изображение мантиссы в двоичной *системе*) посредством умножения на 2:

0,8 x 2 = 1,6;

0,6 x 2 = 1,2;

0,2 x 2 = 0,4;

0,4 x 2 = 0,8;

и так далее до тех пор, пока не получится нулевая дробная часть или не будет достигнута требуемая точность

$0{,}8_{10} = 0{,}1100110..._2$;

- результат *перевода*: $12{,}8_{10} = 1100{,}1100110011..._2$.

Примеры

- Найдем $29{,}25_{10} = ?_{8}$.

Решение имеет вид 1) $29_{10} = 35_{8}$; 2) $0{,}25_{10} = 0{,}2_{8}$;

3) $29{,}25_{10} = 35{,}2_{8}$.

- Найдем $79{,}26_{10} = ?_{16}$.

Решение: 1) $79_{10} = 4F_{16}$; 2) $0{,}26_{10} = 0{,}40_{16}$;

3) $79{,}26_{10} = 4F{,}4_{16}$.

При *переводе* дробной части ограничились нахождением двух значащих цифр после запятой, так как *перевод* точно сделать невозможно.

Для *перевода* из 2-ной в 8-ную и наоборот, из 2-ной в 16-ную и наоборот, из 8-ной в 16-ную и обратно, используется таблица.

При *переводе* в 8-ную *систему* или из нее необходимо группировать в тройки биты, а при *переводе* в 16-ную или из нее – группировать их в четверки битов. Можно добавлять, если нужно, незначащие нули (слева от целой части и справа от мантиссы) или отбрасывать их.

ОСНОВАНИЕ СИСТЕМЫ			
10	2	8	16
0	0	000	0000
1	1	001	0001
2	—	010	0010
3	—	011	0011
4	—	100	0100
5	—	101	0101
6	—	110	0110
7	—	111	0111
8	—	—	1000
9	—	—	1001
10	—	—	1010
11	—	—	1011
12	—	—	1100
13	—	—	1101
14	—	—	1110
15	—	—	1111

Переводы в смешанных системах

- Из 2-ной *системы* в 8-ную (двоично-восьмеричное изображение):

$$101,10111_2 = \underset{5_8}{\underline{101}}, \underset{5_8}{\underline{101}} \underset{6_8}{\underline{110_2}} = 5,56_8;$$

- из 8-ной *системы* в 2–ную (восьмерично-двоичное изображение):

$$6,24_8 = \underset{110_2,}{\underline{6}}, \underset{010_2}{\underline{2}} \underset{100_2}{\underline{4_8}} = 110,0101_2;$$

Переводы в смешанных системах

- из 2-ной *системы* в 16-ную (двоично-шестнадцатеричное изображение):

$$101,10111_2 = \underset{5_{16}}{0101}\ ,\ \underset{11(B)_{16}}{1011}\ \underset{8_{16}}{1000_2} = 5,B8_{16};$$

- из 16-ной *системы* в 2-ную (шестнадцатерично-двоичное изображение):

$$1A,F3_{16} = \underset{0001_2}{1}\ \underset{1010_2,}{A}\ ,\ \underset{1111_2}{F}\ \underset{0011_2}{3_{16}} = 11010,11110011_2.$$

Арифметические операции:

- *Сложение* в двоичной *системе счисления* осуществляется по правилам

0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, 1 + 1 = 2_{10} = 10_2 (единица идет в старший разряд).

- *Вычитание* в двоичной *системе счисления* имеет вид

0 – 0 = 0, 1 – 0 = 1, 1 – 1 = 0, 10 – 1 = 1.

- Умножение в двоичной *системе счисления* имеет вид

0 x 0 = 0, 0 x 1 = 0, 1 x 0 = 0, 1 x 1 = 1.

- Деление в двоичной *системе счисления* имеет вид

  0 : 0 = не определено, 1 : 0 = не определено,

  0 : 1 = 0, 1 : 1 = 1.

Как представляются целые числа?

- Обычно занимают в памяти компьютера один или два байта. В однобайтовом формате принимают значения от 00000000_2 до 11111111_2. В двубайтовом формате - от $00000000\ 00000000_2$ до $11111111\ 11111111_2$.

Диапазоны значений целых чисел без знака

Формат числа в байтах	Диапазон	
	Запись с порядком	Обычная запись
1	$0 \ldots 2^8-1$	0 ... 255
2	$0 \ldots 2^{16}-1$	0 ... 65535

Примеры:

- а) число $72_{10} = 1001000_2$ в **однобайтовом** формате:

Номера разрядов	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Биты числа	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**

- б) это же число в **двубайтовом** формате:

Номера разрядов	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Биты числа	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**

- в) число 65535 в **двубайтовом** формате:

Номера разрядов	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Биты числа	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

Целые числа со знаком

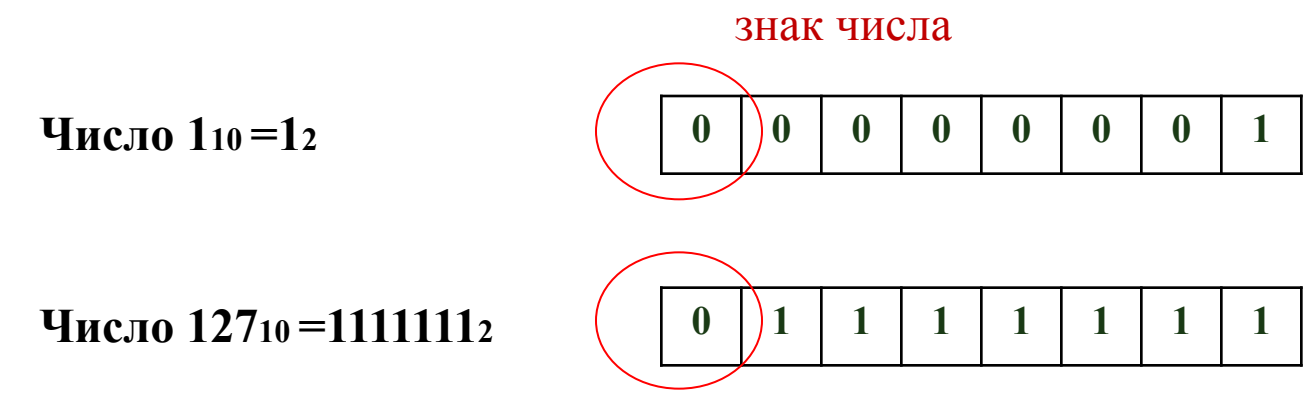
- Обычно занимают в памяти компьютера один, два или четыре байта, при этом самый левый (старший) разряд содержит информацию о знаке числа.

Диапазоны значений целых чисел со знаком

Формат числа в байтах	Диапазон	
	Запись с порядком	Обычная запись
1	$-2^7 \dots 2^7-1$	-128 ... 127
2	$-2^{15} \dots 2^{15}-1$	-32768 ... 32767
4	$-2^{31} \dots 2^{31}-1$	-2147483648 ... 2147483647

Прямой, обратный и дополнительный код.

- Рассмотрим особенности записи целых чисел со знаком на примере **однобайтового формата**, при котором для знака отводится один разряд, а для цифр абсолютной величины - семь разрядов.
- **Положительные числа** в прямом, обратном и дополнительном кодах изображаются одинаково - двоичными кодами с цифрой 0 в знаковом разряде. Например:



Прямой, обратный и дополнительный код.

- **Отрицательные числа** в прямом, обратном и дополнительном кодах имеют разное изображение.

- 1. **Прямой код**. В знаковый разряд помещается цифра 1, а в разряды цифровой части числа - двоичный код его абсолютной величины. Например:

Прямой код числа -1

1 0 0 0 0 0 0 1

Знак числа “-”

Прямой код числа -127

1 1 1 1 1 1 1 1

Знак числа “-”

- 2. **Обратный код**. Получается инвертированием всех цифр двоичного кода абсолютной величины числа, включая разряд знака: нули заменяются единицами, а единицы — нулями. Например:

Прямой код числа -1

1 0 0 0 0 0 0 1

Обратный код числа -1

1 1 1 1 1 1 1 0

Прямой код числа -127

1 1 1 1 1 1 1 1

Обратный код числа -127

1 0 0 0 0 0 0 0

3. **Дополнительный код**. Получается образованием обратного кода с последующим прибавлением единицы к его младшему разряду. Например:

Дополнительный код числа -1

1 1 1 1 1 1 1 1

Дополнительный код числа -127

1 0 0 0 0 0 0 1

Прямой, обратный и дополнительный код.

- Обычно **отрицательные десятичные числа при вводе в машину автоматически преобразуются в обратный или дополнительный двоичный код** и в таком виде хранятся, перемещаются и участвуют в операциях. При выводе таких чисел из машины происходит **обратное преобразование в отрицательные десятичные числа.**

Сложение обратных кодов.

- Здесь при сложении чисел A и B имеют место четыре основных и два особых случая:
- **1. A и B положительные.** При суммировании складываются все разряды, включая разряд знака. Так как знаковые разряды положительных слагаемых равны нулю, разряд знака суммы тоже равен нулю.

Десятичная запись	Двоичные коды
+ 3	+ 0 0000011
7	0 0000111
10	0 0001010

- **2. А положительное, В отрицательное и по абсолютной величине больше, чем А.**

Десятичная запись	Двоичные коды	
+ 3	+ 0 0000011	
-10	1 1110101	Обратный код числа -10
-7	1 1111000	Обратный код числа -7

- Получен правильный результат в обратном коде. При переводе в прямой код биты цифровой части результата инвертируются: 1 0000111 = -7_{10}.
- **3. А положительное, В отрицательное и по абсолютной величине меньше, чем А.**

Десятичная запись	Двоичные коды	
+ 10	+ 0 0001010	
-3	1 1111100	Обратный код числа -3
7	0 0000110	
	→ +1	
	0 0000111	

- Компьютер исправляет полученный первоначально неправильный результат (6 вместо 7) **переносом единицы** из знакового разряда в младший разряд суммы.

4. A и B отрицательные.

- Полученный первоначально неправильный результат (обратный код числа -11_{10} вместо обратного кода числа -10_{10}) компьютер исправляет переносом единицы из знакового разряда в младший разряд суммы. При переводе результата в прямой код биты цифровой части числа инвертируются:

  $1\ 0001010 = -10_{10}$.

При сложении может возникнуть ситуация, когда старшие разряды результата операции не помещаются в отведенной для него области памяти. Такая ситуация называется **переполнением разрядной сетки формата числа.** Для обнаружения переполнения и оповещения о возникшей ошибке в компьютере используются специальные средства.

- **5. A и B положительные, сумма A+B больше, либо равна 2^{n-1},** где n — количество разрядов формата чисел (для однобайтового формата n=8, $2^{n-1} = 2^7 = 128$).

Десятичная запись	Двоичные коды	
+ 65	+ 0 1000001	
97	0 1100001	
162	1 0100010	Переполнение

- Семи разрядов цифровой части числового формата **недостаточно** для размещения восьмиразрядной суммы ($162_{10} = 10100010_2$), поэтому **старший разряд суммы оказывается в знаковом разряде.** Это вызывает **несовпадение знака суммы и знаков слагаемых**, что **является свидетельством переполнения разрядной сетки**.

- **6. A и B отрицательные, сумма абсолютных величин A и B больше, либо равна 2^{n-1}.**

Десятичная запись	Двоичные коды	
- 63	1 1000000	Обратный код числа -63
+ -95	+ 1 0100000	Обратный код числа -95
158	0 1100000	Переполнение
	→ +1	

- Здесь **знак суммы** тоже **не совпадает со знаками слагаемых**, что свидетельствует о **переполнении разрядной сетки**.

Сложение дополнительных кодов.

- Здесь также имеют место рассмотренные выше шесть случаев:
- **1. A и B положительные.** Здесь нет отличий от случая 1, рассмотренного для обратного кода.
- **2. A положительное, B отрицательное и по абсолютной величине больше, чем A.**

Десятичная запись	Двоичные коды	
3	0 0000011	
+ -10	+ 1 1110110	Дополнительный код числа -10
-7	1 1111001	Дополнительный код числа -7

- Получен правильный результат в дополнительном коде. При переводе в прямой код биты цифровой части результата инвертируются и к младшему разряду прибавляется единица: 1 0000110 + 1 = 1 0000111 = -7_{10}.

- **3. А положительное, В отрицательное и по абсолютной величине меньше, чем А.**

Десятичная запись	Двоичные коды
+ 10	+ 0 0001010
-3	1 1111101 Дополнительный код числа -3
7	0 0000111
	→ перенос отбрасывается

- Получен правильный результат. Единицу переноса из знакового разряда компьютер отбрасывает.
- **4. А и В отрицательные.**

Десятичная запись	Двоичные коды
+ -3	+ 1 1111101 Дополнительный код числа -3
-7	1 1111001 Дополнительный код числа -7
-10	1 1110110 Дополнительный код числа -10
	→ перенос отбрасывается

Получен правильный результат в дополнительном коде. **Единицу переноса** из знакового разряда компьютер **отбрасывает**.

Обратный и дополнительный код

- *Обратным кодом* числа в *системе* с основанием *p* называется число в этой *системе*, получаемое заменой цифры, символа в каждом разряде числа на его дополнение до максимальной цифры в *системе* (то есть до *p – 1*).
- *Дополнительный код = обратный код* + единица в младшем разряде.
- *Пример.* 10011 двоичное число,

01100 *обратный код* этого двоичного числа,

01101 *дополнительный код* этого двоичного числа;

Найти *обратный и дополнительный коды* для: 457_8, $A9_{16}$.

Точность

- Точность в чистой математике – понятие абстрактное и в вычислительной математике может возникать иллюзия точности там, где ее на самом деле нет, – если нет корректной договоренности о пределах возможных значений неизбежных погрешностей в рамках рассматриваемых вычислительных ресурсов, например, трудоемкости и времени, а также не оговорена стратегия управления этой погрешностью.

Точность

- Так как диапазон n-разрядных чисел *системы счисления* с основанием *p* находится в пределах , то для представления дробных чисел этот диапазон еще снижается, поскольку часть разрядов необходимо отвести под изображение мантиссы. Таким образом, имеются так называемые "*зоны нечувствительности*" форм представления чисел в n-разрядных арифметиках.

- В 1937 году *Конрадом Цузе* для увеличения диапазона чисел, представимых в арифметике двоичных чисел, а также для повышения точности этого представления, было предложено использовать представление чисел в *нормализованной форме с плавающей запятой*, т. е. число *x* представляется в виде:

$$x = m \times p^k$$

где *m* – мантисса числа, *k* – целый порядок числа,

p – основание $p^{-1} \leq |m| < 1$

-0.000001 (одна миллионная):-0.1×10^{-5}

В вычислительных машинах показатель степени принято отделять от мантиссы буквой "E" (exponent).

$1{,}528535047 \times 10^{-25}$ в большинстве языков программирования высокого уровня записывается как 1.528535047E-25.

Пример:

- Пусть даны два числа:

$$x = m \times p^k \qquad y = n \times p^l$$

($k > l$). Тогда можно проверить, что результаты выполнения операций будут равны:

$$\begin{aligned} X + y &= (m + n \times p^{l-k}) \times p^k, \\ X - y &= (m - n \times p^{l-k}) \times p^k, \\ x \times y &= (m \times n) \times p^{k+l}, \\ x / y &= (m / n) \times p^{k-l}. \end{aligned}$$

- Если из *n* разрядов, отводимых под изображение чисел, m двоичных разрядов отвести под мантиссу, *k* – под порядок, один разряд – под знак числа и один разряд – под знак порядка (например, 0 – плюс, 1 – минус), то диапазон представимых в форме с плавающей запятой чисел резко увеличивается ($m + k + 2 = n$):

$$-(0.111\ldots1)_2 \times (10)_2^{+(111\ldots1)_2} \leq x \leq +(0.111\ldots1)_2 \times (10)_2^{+(111\ldots1)_2}$$

(многоточие соответствует k единицам).

Пример:

Рассмотрим представление R с 3-разрядной мантиссой со знаком в диапазоне $0{,}1 \leq |f| < 1$ и 2-разрядной экспонентой со знаком.

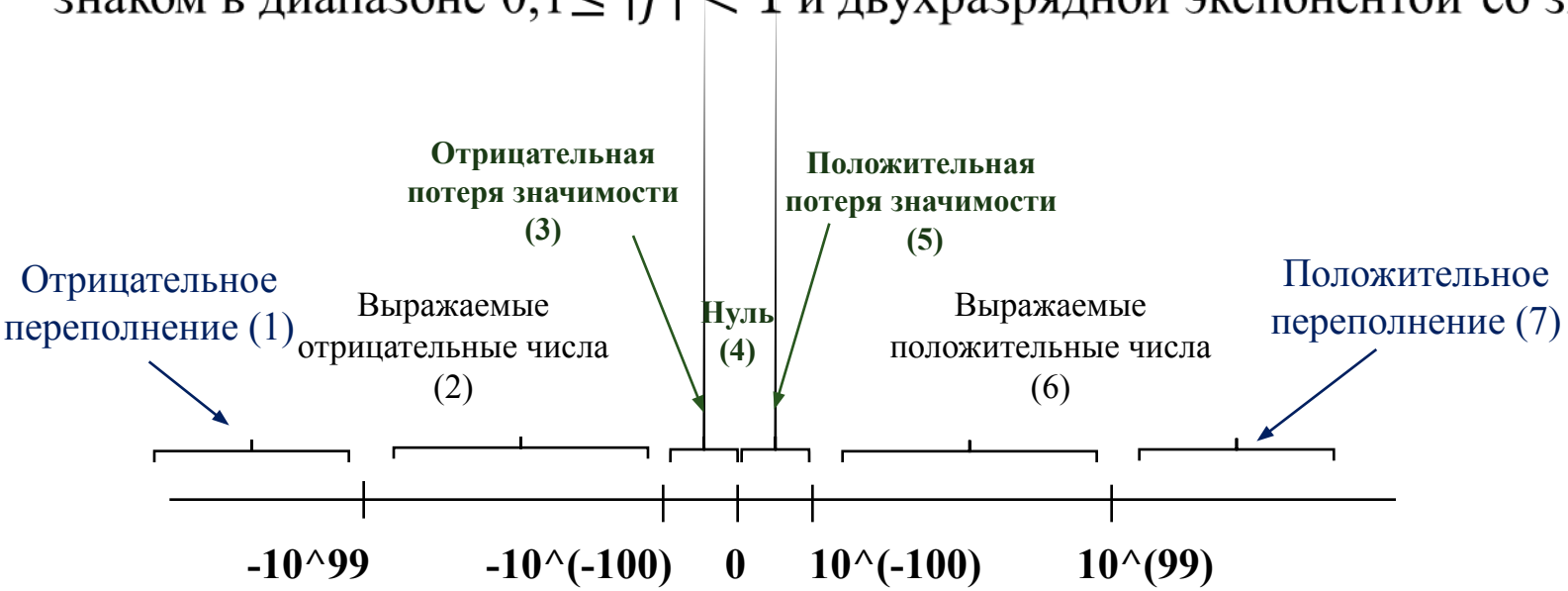
- Диапазон от -0,100×10^(-99) до +0,999×10^(+99)

  [199 разрядов, а для записи требуется 5 разрядов и 2 знака]

- Числа, меньшие нижней границы положительных чисел и большие верхней границы отрицательных чисел, считаются равными нулю, не различаются между собой.
- Числа, большие верхней границы положительных чисел, полагаются равными положительной бесконечности (меньшие нижней границы отрицательных – отрицательной бесконечности).
- Сравнение двух разных по величине чисел в арифметике с ограниченной разрядностью может поэтому приводить к неверному результату, как и сравнение двух равных в таких *системах* чисел с точки зрения математической.

- Такое представление очень удобно для хранения в ЭВМ, так как на самом деле необходимо хранить не само число, а его знак, мантиссу, порядок и знак порядка, и все операции с числами сводятся к операциям с этими объектами.
- Операции же с этими объектами просты: сравнение знаков, увеличение, уменьшение порядка, сложение мантисс, нормализация, то есть в конечном итоге сводятся к достаточно просто реализуемым операциям сдвига, выравнивания, сравнения разрядов.

Пример. Рассмотрим представление R с трехразрядной мантиссой со знаком в диапазоне $0{,}1 \leq |f| < 1$ и двухразрядной экспонентой со знаком.



Ось действительных чисел

(1) и (7) – ошибка переполнения
(3) и (5) – ошибка потери значимости
(2) и (6) – промежутки между числами не постоянны

Числа с плавающей точкой не образуют континуума. В двухзнаковой пятиразрядной системе можно выразить ровно 179 100 положительных чисел, 179 100 отрицательных числе и 0, т.е. всего 358201 чисел.

К "неудобствам" этой формы представления чисел можно отнести возможность возникновения следующих "особо опасных" ситуаций:

- если число достаточно мало, например, a = 0.12E + 00, то оно может быть представлено любым числом из наименьшего интервала включающего a, в частности числом 0.120000001 или 0.199999999 и в этом случае сравнивать на равенство "в лоб" нельзя (вещественные числа в форме с плавающей запятой опасно сравнивать на совпадение);
- порядок выполнения операций может влиять на результат, например, в 4-разрядной арифметике с фиксированной запятой 20.0000 + 0.0001 = 20.0001, но при этом 0.2000E+02 + 0.1000E–05 = 0.2000E + 02;

- может возникнуть так называемая ситуация "переполнения порядка" при сложении (умножении) очень больших чисел или "исчезновения порядка" при сложении (умножении) "очень малых чисел“:

- ✔ 0.6000E+39 × 0.1200E+64 = 0.9999E+99 (или же не определено)
- ✔ 0.6000E–35 × 0.0200E–65 = 0.9999E – 99 (или же не определено)

- при сложении чисел с плавающей запятой (а, в конечном счете, все операции выполняются через сложение) происходит выравнивание порядков для последующего сложения мантисс, а при выравнивании степеней может происходить потеря (усечение) младших разрядов, например, такая ситуация может возникнуть при сложении одного "очень большого числа" с одним "очень малым числом"

Ненормализованная форма. Основание степени 2

$$\underbrace{0}_{\text{знак}} \quad \underbrace{1010100}_{\text{экспонента со смещением 64}} \quad . \underbrace{0000\ \ 0000\ \ 0001\ \ 1011}_{\text{мантисса}} = 2^{20}(1\times2^{-12}+1\times2^{-13}+1\times2^{-15}+1\times2^{-16}) = 432$$

$+$

$(84-64=20)$

$1\times2^{-12}+1\times2^{-13}+1\times2^{-15}+1\times2^{-16}$

Для приведения к нормализованному виду нужно сдвинуть мантиссу влево на 11 бит и вычесть 11 из экспоненты

Нормализованная форма. Основание степени 2

$$\underbrace{0}_{\text{знак}} \quad \underbrace{1001001}_{\text{экспонента со смещением}} \quad . \underbrace{1101\ \ 1000\ \ 0000\ \ 0000}_{\text{мантисса}} = 2^{9}(1\times2^{-1}+1\times2^{-2}+1\times2^{-4}+1\times2^{-5}) = 432$$

$+$

$73-64=9$

$1\times2^{-1}+1\times2^{-2}+1\times2^{-4}+1\times2^{-5}$

*В примерах указана 16-разрядная мантисса (включая знаковый бит) и 7-разрядная экспонента.

Ненормализованная форма. Основание степени 16

$$\underbrace{0}_{\text{знак}} \quad \underbrace{1000101}_{\text{экспонента со смещением}} \quad .\overbrace{0\,0\,0\,0 \quad 0\,0\,0\,0 \quad 0\,0\,0\,1 \quad 1\,0\,1\,1}^{16^{-1}\quad 16^{-2}\quad 16^{-3}\quad 16^{-4}} = 16^5(1\times16^{-3}+B\times16^{-4}) = 432$$

$$+ \qquad 69-64=5 \qquad \text{мантисса:} 1\times16^{-3}+B\times16^{-4}$$

Для приведения к нормализованному виду нужно сдвинуть мантиссу влево на 2 шестнадцатеричных разряда и вычесть 2 из экспоненты

Нормализованная форма. Основание степени 16

$$\underbrace{0}_{\text{знак}} \quad \underbrace{1000011}_{\text{экспонента со смещением}} \quad .\underbrace{0\,0\,0\,1 \quad 1\,0\,1\,1 \quad 0\,0\,0\,0 \quad 0\,0\,0\,0} = 16^3(1\times16^{-1}+B\times16^{-2}) = 432$$

$$+ \qquad 67-64=3 \qquad \text{мантисса:} 1\times16^{-1}+B\times16^{-2}$$

*В примерах указана 16-разрядная мантисса (включая знаковый бит) и 7-разрядная экспонента.

Стандарт IEEE 754

- 1985 г. институт IEEE выпустил стандарт IEEE 754, которому в настоящее время соответствуют команды с плавающей точкой большинства процессоров (Intel, SRARC и JVM).

Разработчик стандарта *Вильям Каган* (William Kahan, университет Беркли)

IEEE 754 определяет 3 формата:

- ✔ с одинарной точностью (32 бит);
- ✔ с удвоенной точностью (64 бит);

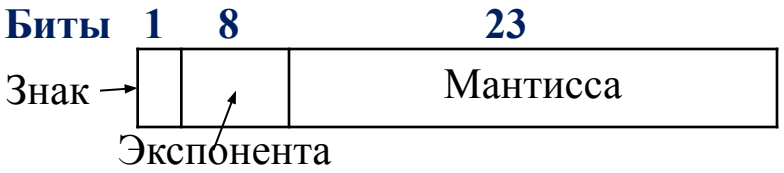
] используется основание степени 2 для мантисс и смещенная экспонента

- ✔ с повышенной точностью (80 бит).

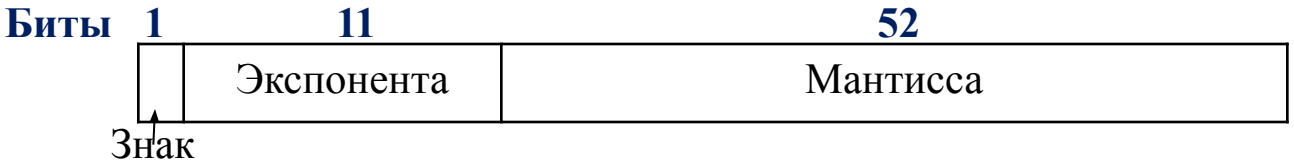
используется в арифметике с плавающей точки для уменьшения ошибки округления

Форматы стандарта IEEE с плавающей запятой

Одинарная точность



Удвоенная точность



- 1. Знаковый бит (0 – положительное число; 1 – отрицательное)
- 2. Смещенная экспонента: смещения 127 (одинарная точность) и 1023 (удвоенная точность)

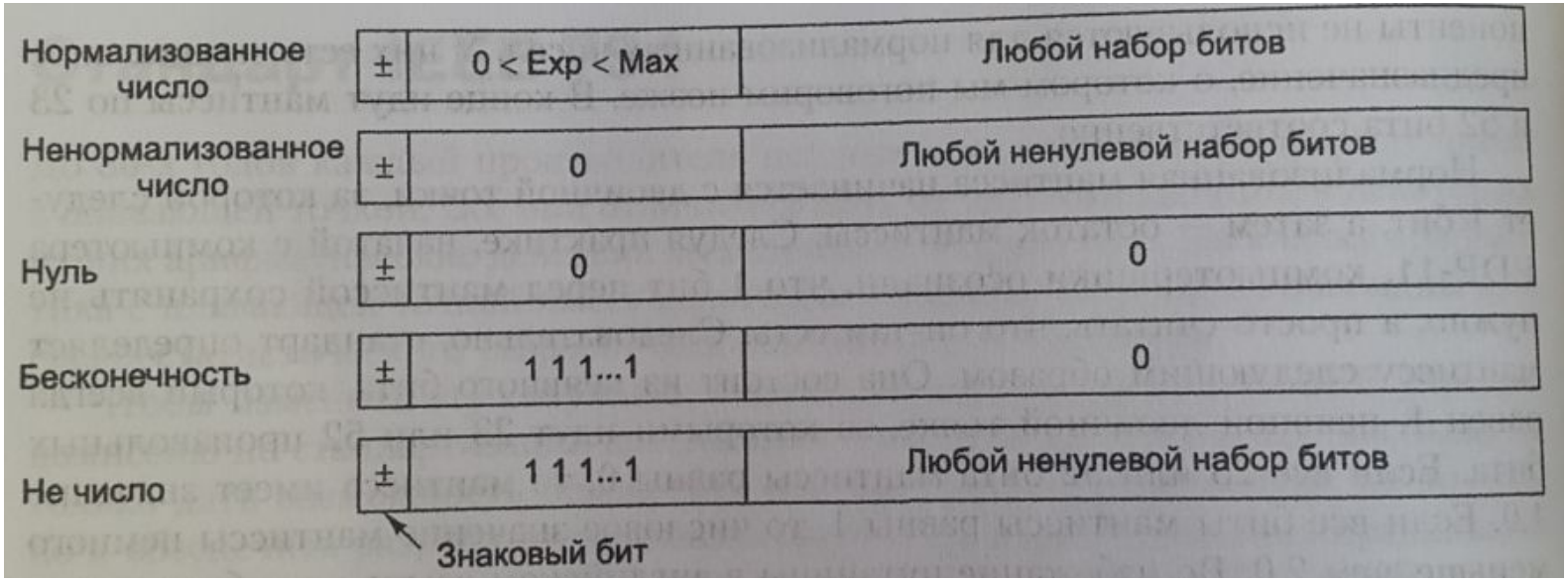
  [минимальная (0) и максимальная (255 и 2047) экспоненты не используются для нормализованных чисел]
- 3. Мантиссы по 23 и 52 бит

Форматы стандарта IEEE с плавающей запятой

Нормализованная мантисса начинается с двоичной точки за которой следует 1 бит, а затем – остаток мантиссы.

- В стандарте IEEE мантисса состоит из неявного бита, который всегда равен 1, неявной двоичной точки, за которыми идут 23 или 52 произвольных бита. В этом случае говорят о *значащей части числа* (significant).
- Значащая часть числа (s) всех нормализованных чисел лежит в диапазоне $1 \leq s < 2$
- Проблемы: переполнение, потеря значимости и неинициализированные числа.

Числовые типы стандарта IEEE



Если модуль результата меньше самого маленького нормал-ого числа с плавающей точкой => результат 0 или ошибка потери значимости.

<u>В IEEE введены ненормализованные числа:</u>

✔Имеют экспоненту 0 и мантиссу 23 и 52 бит.

✔Неявный бит 1 слева от двоичной точки превращается в 0

Ненормализованные числа можно легко отличить от нормализованных, т.к. у последних нет нулевой экспоненты

Форматы стандарта IEEE с плавающей запятой

- Самое маленькое число 1,0×2^(-126)
  [1 в экспоненте и 0 в мантиссе]
- Самое большое число примерно 0,9999999×2^(127)
  [0 в экспоненте и все 1 в мантиссе]

По мере уменьшения результат экспонента по прежнему остается равной 0, а первые несколько бит мантиссы превращаются в 0 (сокращается значение и число значимых бит мантиссы).

Самое маленькое ненулевое ненормализованное число содержит 1 в крайнем правом бите, все остальные биты 0.

Экспонента представляет 2^(-127), мантисса – 2^(-23), т.е. значение равно 2^(-150)

Форматы стандарта IEEE с плавающей запятой

- Такая схема предусматривает постепенное исчезновение значимых разрядов, а не перескакивает на 0, когда результат не удается выразить в виде нормализованного числа.
- Присутствует два нуля, положительный и отрицательный, определяемые по знаковому биту. Оба имеют экспоненту 0 и мантиссу 0. Бит слева от двоичной точки по умолчанию равен 0, а не 1.