Лекция 5. Представление чисел в компьютере и действия с ними

Так как компьютер может различить только нулевое и единичное состояния бита, то он работает в системе счисления с основанием «2». Вся информация в компьютере – команды и целые программы, аудио и видеоданные, текстовые файлы, числовые массивы представляются только при помощи нулей и единиц. В современном компьютере минимальной адресуемой единицей информации является байт, представляющий собой 8 битов, или 8-разрядное двоичное число. При этом невозможно различить различные типы данных, если неизвестно заранее, что записано в конкретной ячейке памяти. Биты 01000001 могут представлять как число 65, так и букву «А». Если программа определяет элемент данных для арифметических целей, то 01000001 представляет двоичное число, эквивалентное числу 65. Если программа определяет элемент данных как символ, тогда 01000001 представляет собой букву «А».

Каждый бит является логическим аргументом или логической функцией, и с ними можно проводить логические операции. Основной особенностью представления чисел в памяти компьютера является то, что, в отличие от записи числа на бумаге, компьютерные ячейки имеют ограниченный размер и, следовательно, вынуждают использовать при записи чисел и действиях с ними конечное количество разрядов. Это приводит к тому, что бесконечное множество чисел заменяется конечным множеством их представлений. Способы представления и допустимые над ними действия различны для следующих числовых множеств:

- целые положительные числа (целые числа без знака);
- целые числа со знаком;
- вещественные нормализованные числа.

Рассмотрим подробнее перечисленные группы.

Целые числа

Минимальный объем памяти, выделяемый для хранения целого числа, один байт. Один байт – это восемь бит, тогда количество различных значений, которые можно уместить в этот объем, равняется 2^8=256.

Давайте поразмышляем, какие числа мы поместили бы в один байт памяти компьютера? Рассмотрим все возможные варианты ответов.

От 1 до 256. Удобен ли этот диапазон? Даже если нам никогда не понадобятся отрицательные числа, то нулевое значение нам нужно будет всегда.

От 0 до 255. Да такой вариант существует. Например, тип «byte» в Паскале использует этот формат хранения данных. Это представление называется *беззнаковым*, и целые числа в памяти компьютера хранятся в явном прямом коде. Представление двоичных чисел типа «byte» в памяти компьютера иллюстрируется таблицей 1.

Таблица. 1

Значение	**Представление**
0	00000000
1	00000001
2	00000010
3	00000011
…	…
255	11111111

Операция сложения беззнаковых чисел происходит по стандартному алгоритму двоичной арифметики.

Сложим двоичные числа 65 и 42.

01000001	65
+	+
00101010	42
01101011	107

Очевиден следующий вопрос – что произойдет, если сумма двух 8-рарядных беззнаковых чисел превысит значение 255? Например, 167+198.

10100111	167
+	+
11000110	198
1 01101101	365

Произошел перенос значащего разряда за разрядную сетку. Следовательно, в разрядной сетке останется число 01101101, которое интерпретируется компьютером как 109. Если в программе, в которой происходит данное действие, контролируются подобные ситуации, то появится сообщение об ошибке. Если такая проверка не производится, то результат вычислений будет неправильным.

Помимо однобайтового представления беззнаковых чисел (byte) в языке программирования Паскаль поддерживается двухбайтовый тип «Word».

Знаковое представление целых чисел. Давайте придумаем способ, с помощью которого компьютер будет отличать положительные числа от отрицательных. Количество различных значений в байте памяти, как говорилось выше, 2^8=256. Получается 128 отрицательных значений и 128 положительных. Естественное предположение, что старший бит (под номером 7) отдается на обозначение знака. Пусть 1 в старшем разряде представляет отрицательное число, а 0 – число положительное. Возможный вариант представления целых чисел со знаком приведен в таблице 2.

Таблица 2

Значение	**Представление**
…	…
+2	00000010
+1	00000001
0	00000000
-1	10000001
-2	10000010
…	…

Получаем 127 положительных значений, 127 отрицательных значений и нулевое значение, то есть 255 значений. Потерялось одно значение 10000000.

Если цифровая часть положительных и отрицательных чисел всегда содержит абсолютную величину числа, то такой способ представления знаковых чисел называют *прямым кодом*. Обработка чисел, представленных в прямом коде:

- требует отдельных операций над цифровой и знаковой частями,
- требует альтернативного выполнения операций сложения и вычитания,
- приводит к появлению двух представлений нуля: +0 (00000000) и -0 (10000000).

Наиболее сложной для реализации в данном представлении является операция вычитания.

С этой проблемой столкнулся еще Блез Паскаль в XVII веке при конструировании своей суммирующей машины. При реализации идеи автоматического переноса десятков Паскаля остановила определенная трудность: изобретенный им механизм переноса

десятков работал при вращении счетных колес только в одном направлении, а это не позволяло производить вычитание вращением колес в противоположную сторону. Простой и остроумный выход из этого положения, найденный Паскалем, был настолько удачен, что используется в современных ЭВМ. Паскаль *заменил вычитание сложением с дополнением вычитаемого*.

Для 8-разрядной машины Паскаля, работавшей в десятичной системе счисления, дополнением числа A будет $(100000000 - A)$, поэтому операция вычитания $B - A$ может быть заменено сложением $B + (100000000 - A) = 100000000 + (B - A)$. Получившееся число будет больше искомой разности на 100.000.000, но так как машина 8-разрядная, то единица в девятом разряде просто пропадает при переносе десятков из восьмого.

В компьютере происходит все то же, только в двоичной системе счисления.

Дополнением *k*-разрядного целого числа Z в двоичной системе счисления называется величина

$$D(Z,k) = 2^k - Z$$

Данную формулу можно представить в ином виде:

$$D(Z,k) = ((2^k - 1) - Z) + 1.$$

Число $2^k - 1$, состоит из *k* наибольших в данной системе счисления цифр (в нашем случае одного байта – 11111111). Поэтому $(2^k - 1) - Z$ можно получить путем дополнения до 1 каждой цифры числа Z и последующим прибавлением к последнему разряду 1.

Так как в двоичной системе счисления дополнением 1 является 0, а дополнением 0 является 1, построение $D(Z,k)$ сводится к инверсии данного числа, т.е. замена нулей единицами и единиц нулями, и прибавлением 1 к последнему разряду. Другими словами, дополнение двоичного числа формируется в два этапа:

1. строится инвертированное представление исходного числа (в каждом бите 0 заменяется на 1, а 1 заменяется на 0). Этот код числа называется обратным кодом.
2. к обратному коду прибавляется 1 по правилам двоичной арифметики.

Дополнительный код двоичных целых чисел строится по следующим правилам:

- для $Z \geq 0$ дополнительный код совпадает с самим числом;
- для $Z < 0$ дополнительный код совпадает с дополнением модуля числа, т.е. $D(|Z|,k)$.

Найдем диапазон знаковых целых чисел, занимающих один байт в памяти компьютера. Будем строить таблицу от нуля, затем представление +1 и -1, +2 и -2, и т.д. (см. таблицу 3).

Таб.3

Значение	**Представление**
+127	01111111
+126	01111110
…	…
+2	00000010
+1	00000001
0	00000000
-1	11111111
-2	11111110
…	…
-126	10000010
-127	10000001
-128	10000000

После построения такой таблицы становится очевидным, что ось симметрии диапазона проходит между 0 и -1. Следовательно, диапазон знаковых чисел, занимающих один байт в памяти компьютера – от -128 до +127. В языке программирования Паскаль этот тип данных обозначается служебным типом ShortInt (короткое целое).

Если число занимает два байта, то диапазоны следующие: 0...65535 (Word – слово), -32768...32767 (Integer – целое). Если число занимает четыре байта, то диапазоны следующие: 0..4294967295 (такого типа в Паскале нет), -2147483648.. 2147483647 (LongInt – длинное целое).

Анализируя полученную выше таблицу значений можно сделать вывод, что *отрицательные двоичные числа содержат единичный бит в старшем разряде.*

Число 65:	01000001	
Инверсионные биты:	10111110	
Плюс 1:	10111111	равно -65

Для определения абсолютного значения отрицательного двоичного числа в машинном представлении просто повторяют предыдущие операции: инвертируют все биты и прибавляют единицу.

Двоичное дополнение:	10111111	
Инверсионные биты:	01000000	
Плюс 1:	01000001	равно +65

Проверим, даст ли сумма +65 и -65 в результате 0:

01000001	65
+	+
10111111	-65
1 00000000	0

Все восемь бит имеют нулевое значение. Единичный бит, вынесенный за разрядную сетку влево, потерян.

Рассмотрим в общем случае сложение двух знаковых двоичных числа *C*=A+*B*. Все возможные варианты представлены в таблице 4.

Рассмотрим все подобные случаи на примерах.

Убедимся, что результат, получаемый компьютером при сложении положительного и отрицательного числа, получается корректным.

Вычтем 42 из 65. Найдем дополнительный код числа -42.

Число 42:	00101010	
Инверсионные биты:	11010101	
Плюс 1:	11010110	равно -42

Прибавим к +65 -42 по правилам машинной арифметики.

01000001	65
+	+
11010110	-42
1 00010111	23

Результат 23 является корректным.

Таблица 4

Старший бит числа *A*	Старший бит числа *B*	Старший бит числа *C*	Перенос единицы из разрядной сетки	Комментарий
0	0	0	Нет	Сложение двух положительных чисел без переполнения. Знак остается таким же, как у обоих операндов. Результат корректен
0	0	1	Нет	Перенос единицы в знаковый разряд при сложении двух положительных чисел. Произошло изменение знака результата (поскольку оба числа положительны, результат должен быть также положительным). Результат некорректен.
1	1	1	Есть	Сложение двух отрицательных чисел. Знак совпадает со знаками обоих операндов. Несмотря на перенос единицы в 9-й разряд, результат корректен.
1	1	0	Есть	При сложении двух отрицательных чисел произошло изменение знака результата. Результат некорректен.
0	1	0 (1)	Есть (Нет)	Сложение чисел с разными знаками; $A>\|B\|$ ($A<\|B\|$). Знак результата равен знаку одного из слагаемых. Вне зависимости от того, произошел ли перенос за границы разрядной сетки, результат корректен.

Найдем по правилам машинной арифметики сумму +42 и -65.

Решение.

Число 65: 01000001

Инверсионные биты: 10111110

Плюс 1: 10111111 равно -65

Прибавим к -65 +42 по правилам машинной арифметики.

$$\begin{array}{r} 10111111 \\ +\ 00101010 \\ \hline 11101001 \end{array} \qquad \begin{array}{r} -65 \\ +\ 42 \\ \hline ? \end{array}$$

Убедимся, что результат корректный. Так как старший седьмой бит результата равен единице, то это отрицательное число в дополнительном коде. Найдем его абсолютное значение.

Число ?: 11101001

Инверсионные биты: 00010110

Плюс 1: 00010111 равно 23

Следовательно, в результате мы получили число -23, результат корректный.

Найдем по правилам машинной арифметики сумму -42 и -65.

Решение. Сначала переводим оба числа в дополнительный код. Затем складываем эти числа.

	10111111		-65
+		+	
	11010110		-42
	1 10010101		?

Переход единицы в знаковый разряд и за разрядную сетку не вызывает ошибки. Проверим полученный результат. Старший бит равен 1, следовательно, полученное число отрицательное и представлено в дополнительном коде.

Число ?:	10010101	
Инверсионные биты:	01101010	
Плюс 1:	01101011	равно 107

Результат корректный.

Найдем результат сложения двух положительных знаковых целых числа: 102 и 54.

Решение.

	01100110		102
+		+	
	00110110		54
	10011100		?

Произошел перенос единичного разряда в знаковый разряд, выноса за разрядную сетку не было, следовательно, результат будет некорректным. Попробуем его перевести в десятичную систему счисления. Так как тип данных – знаковый, то единица в старшем разряде говорит нам о том, что число отрицательное и представлено в дополнительном коде.

Число ?:	10011100	
Инверсионные биты:	01100011	
Плюс 1:	01100100	равно 100

Следовательно, результат должен интерпретироваться как -100.

Умножение целых чисел

При умножении двух целых беззнаковых чисел размером в n бит максимальный результат, получаемый микропроцессором, может иметь длину $2n$ бит.

Простейший способ умножения целых беззнаковых чисел $X \times Y = Z$ есть суммирование множимого X с накоплением его столько раз, сколько единиц содержит значение множителя Y. Этот способ требует при больших значениях Y значительных затрат времени на выполнение многократных операций сложения, и поэтому применение его в компьютерах ограниченно.

Сокращение количества операций сложения при реализации умножения основано на методах совместного анализа цифр множителя. Эти методы дробят процесс получения произведения на ряд шагов, связанных с формированием *частичных произведений* (ЧП) – произведений множимого на отдельные разряды или группы разрядов множителя – и их суммированием, то есть формированием *суммы частичных произведений* (СЧП).

Методы *умножения двоичных беззнаковых чисел* основаны на представлении произведения в виде полинома:

$$Z = X \cdot Y = X \cdot \left(\sum_{i=0}^{n-1} y_i \cdot 2^i \right) = \sum_{i=0}^{n-1} \left(X \cdot y_i \right) \cdot 2^i = \sum_{i=0}^{n-1} ЧП_i \cdot 2^i , \qquad (2)$$

где $y_i \in \{0,1\}$ – двоичные цифры множителя, $ЧП_i = X \cdot y_i$ – частичные произведения ($ЧП_i = X$, если $y_i = 1$, и $ЧП_i = 0$, если $y_i = 0$).

Заметим, что умножение $ЧП_i \neq 0$ на степень 2^i эквивалентно сдвигу множимого на i разрядов влево.

Формирование произведения, согласно формуле (2), представляется как последовательность таких действий:

1. обнуление *СЧП*;
2. анализ младшего разряда множителя: если y_0=1, выполняется сложение $ЧП_0$=X с *СЧП* и переход к п.3; если y_0=0, сразу переход к п.3;
3. сдвиг множимого влево;
4. анализ очередного разряда множителя: если y_1=1, выполняется сложение $ЧП_1 \cdot 2^1$ с *СЧП* и переход к п.5; если y_1=0, непосредственно переход к п.5;
5. сдвиг множимого влево;
6. анализ очередного y_2 разряда множителя и т.д.

После анализа старшего y_{n-1} разряда множителя осуществляется последнее сложение $ЧП_{n-1} \cdot 2^{n-1}$ с *СЧП* (если y_{n-1}=1), и процесс прекращается. Результирующая *СЧП* является искомой. Очевидно, что неподвижную *СЧП* и сдвигаемое влево на n-1 разряд множимое необходимо размещать в 2n-разрядных форматах, причем операция сложения должна обрабатывать 2n-разрядные данные.

Рассмотрим на примере умножение двух целых беззнаковых чисел. Найдем произведение 45 и 29, пользуясь машинным алгоритмом. В десятичной системе счисления 45·29=1305.

Переведем множимое и множитель в двоичную систему счисления: $45_{10}=101101_2$, $29_{10}=11101_2$.

Шаг i	y_i	ЧП	СЧП
0	1	101101	101101
1	0	1011010	101101
2	1	10110100	101101 +10110100
3	1	101101000	101101 +10110100 +101101000
4	1	1011010000	101101 +10110100 +101101000 +1011010000 10100011001

Таб.5

Итак, результат произведения $10100011001_2=1305_{10}$. То есть, можно сделать вывод, что алгоритм работает правильно.

Очевидно, что при работе с однобайтовыми беззнаковыми числами результат таковым не будет. Так как он занимает 11 разрядов. Можно предположить, что в разрядной сетке результата будет 00011001, то есть в 25_{10}. Проведем машинный эксперимент. Действительно, при отключенной проверке выхода результата за разрядную сетку, на экран выводится 25.

Методы *умножения беззнаковых чисел* применимы и для вычисления произведения *двоичных знаковых* чисел в дополнительных кодах.

Проведем эксперимент. Попробуем умножить два знаковых целых числа по правилам умножения беззнаковых целых чисел.

В двоичном представлении	**Беззнаковые числа**	**Знаковые числа**
11111111	255	-1
×	×	×
11111111	255	-1
1111111000000001	65025	-511

Можно сделать вывод, что правило умножения для беззнаковых целых чисел не подходит для знаковых чисел, представленных в дополнительном коде. Следовательно, просто переносить метод умножения беззнаковых чисел на знаковые нельзя.

Пусть целые двоичные сомножители X и Y представлены в дополнительном коде в n-разрядном формате, старший разряд которого используется для представления знака. При умножении этих чисел возможны четыре ситуации:

1. $X>0$, $Y>0$. Так как $D(X, n)=X$ и $D(Y, n)=Y$, то $D(X\ Y, n)=X\ Y$, то есть результат умножения совпадает с произведением беззнаковых чисел.
2. $X<0$, $Y>0$. Согласно формуле (1), $D(X<0, n)= 2^n+X$, и произведение, полученное методом умножения беззнаковых чисел, - *псевдопроизведение* – равно $D(X,n)\cdot D(Y,n) = 2^n \cdot Y + X \cdot Y$. Правильный же результат равен $D(X \cdot Y < 0, n) = 2^{2n} + X \cdot Y$. Очевидно, что для получения правильного результата из псевдопроизведения необходимо к последнему прибавить 2^{2n} (эту операцию практически не надо выполнять, так как данное число выходит за рамки формата произведения) и вычесть множитель $Y \cdot 2^n$, то есть вычесть множитель, сдвинутый влево на n разрядов.
3. $X>0$, $Y<0$. Эта ситуация аналогична предыдущей. Для получения правильного произведения необходимо вычесть множимое, сдвинутое влево на n разрядов.
4. $X<0$, $Y<0$. Так как $D(X<0, n)=2^n+X$ и $D(Y<0, n)=2^n+Y$, то псевдопроизведение равно $D(X,n)\cdot D(Y,n) = 2^{2n} + 2^n \cdot X + 2^n \cdot Y + X \cdot Y$, а правильное произведение $D(X \cdot Y, n) = X \cdot Y$. Следовательно, для получения правильного произведения из псевдопроизведения необходимо вычесть число $2^n \cdot X + 2^n \cdot Y$, то есть вычесть и множитель и множимое, сдвинутые предварительно на n разрядов влево.

Приведем примеры для каждого рассмотренного случая.

1. При умножении двух положительных знаковых чисел, алгоритм полностью совпадает с алгоритмом умножения беззнаковых чисел. См. примеры выше.
2. Пусть X= -5, а Y=32. Дополнительный код числа X равняется $D(X,$ 8)=11111011, Y=00100000. Тогда псевдопроизведение равно:

11111011	-5
×	×
00100000	32
0001111101100000	? -160

Найдем реальное произведение, отняв от полученного псевдопроизведения следующую поправку: $Y \cdot 2^8 = 00100000_2 \cdot 100000000_2 = 0010000000000000_2$

$$\begin{array}{r} 0001111101100000 \\ - \\ 0010000000000000 \\ \hline \mathbf{1}..1111111101100000 \end{array}$$

Число 1111111101100000 является дополнительным кодом числа -160. Следовательно, результат правильный.

3. Данный случай решается аналогично предыдущему.

4. Пусть X= -5, а Y= -32. Тогда $D(X, 8)$=11111011, $D(Y, 8)$=11100000. Отсюда, псевдопроизведение равно:

$$\begin{array}{r} 11111011 \\ \times\ 11100000 \\ \hline 1101101110100000 \end{array} \qquad \begin{array}{r} -5 \\ \times\ -32 \\ \hline ?\ 160 \end{array}$$

Найдем реальное произведение, отняв от полученного псевдопроизведения следующую поправку

$$2^8 \cdot X + 2^8 \cdot Y = 11111011_2 \cdot 100000000_2 + 11100000_2 \cdot 100000000_2 =$$
$$= (11111011_2 + 11100000_2) \cdot 100000000_2 = 11101101100000000_2$$

$$\begin{array}{r} 1101101110100000 \\ - \\ 11101101100000000 \\ \hline \mathbf{1}..0000000010100000 \end{array}$$

Итак, в разрядной сетке осталось число +160. Что и требовалось получить.

Над множеством целых чисел со знаком операция деления не определена, поскольку в общем случае ее результатом будет вещественное число. Однако допустимыми являются операции целочисленного деления и нахождения остатка от целочисленного деления (те, что в паскале обозначаются div и mod). Точнее, значения обеих величин находятся одновременно в одной процедуре, которая в конечном счете сводится к последовательности вычитаний или, еще точнее, сложений с дополнительным кодом делителя.

Вещественные числа

Система вещественных чисел, применяемая при ручных вычислениях, предполагается бесконечно непрерывной. Это означает, что не существует никаких ограничений на диапазон используемых чисел и точность их представления. Для любого вещественного числа имеется бесконечно много чисел, которые больше или меньше его, а между любыми двумя вещественными числами также находится бесконечно много вещественных чисел.

Реализовать такую систему в технических устройствах невозможно. Во всех компьютерах размеры ячеек памяти фиксированы, что ограничивает систему представимых чисел. Ограничения касаются как диапазона, так и точности представления чисел, т.е. система машинных чисел оказывается конечной и дискретной, образуя подмножество системы вещественных чисел.

Число A_{10} называется нормализованным, если оно представлено в виде:

$$A_{10} = \pm M_{10} \cdot 10^{\pm P_{10}}$$

где M_{10} – мантисса, десятичная правильная дробь, то есть $0{,}1 \le M_{10} < 1$*; P_{10} – порядок, целое десятичное число.*

При нормализации числа происходит своеобразное расчленение его на составляющие: знак числа, знак порядка, модуль мантиссы, модуль порядка.

$297_{10} = 0{,}297*10^3$ $\quad M_{10} = 0{,}297$ $\quad P_{10} = 3$

$0{,}031_{10} = 0{,}31*10^{-1}$ $\quad M_{10} = 0{,}31$ $\quad P_{10} = -1$

$-34{,}176 = -0{,}34176*10^2$ $\quad M_{10} = -0{,}34176$ $\quad P_{10} = 2$

Пример 1. Приведите к нормализованному виду следующие числа, не переводя их в десятичную систему счисления: $-0{,}0000010111_2$, $98765{,}ABC_{16}$.

Решение.

$-0{,}0000010111_2 = -0{,}10111_2 \cdot 10_2^{-101_2}$

$98765{,}ABC_{16} = 0{,}8765ABC_{16} \cdot 10_{16}^{+5_{16}}$

Пример 2. Запишите в естественной форме следующие нормализованные числа: $0{,}1011_2 \cdot 10_2^{10}$, $0{,}12345_{10} \cdot 10_{10}^{3}$, $0{,}ABCD_{16} \cdot 10_{16}^{-1}$

Решение.

$0{,}1011_2 \cdot 10_2^{10} = 0{,}1011_2 \cdot 100_2 = 10{,}11_2$

$0{,}12345_{10} \cdot 10_{10}^{3} = 0{,}12345_{10} \cdot 1000_3 = 123{,}45_{10}$

$0{,}ABCD_{16} \cdot 10_{16}^{-1} = 0{,}ABCD_{16} \cdot 0{,}1_{16} = 0{,}0ABCD_{16}$

В ЭВМ арифметические устройства работают с нормализованными числами, но не с десятичными, а с двоичными.

Для вещественных чисел применяется *формат с плавающей точкой*. Значащие цифры числа находятся в поле мантиссы; поле порядка показывает фактическое положение двоичной запятой в разрядах мантиссы, а бит знака определяет знак числа.

Мантисса, называемая также «дробью» *F (Fraction)*, представлена в прямом коде. Порядок *E (Exponent)* дается в смещенной форме: он равен истинному порядку, увеличенному на значение смещения. Значение смещения для трех разных форматов равно 127 (8 бит), 1023 (10 бит) и 16383 (15 бит). Задание порядка в форме со смещением упрощает операцию сравнения чисел с плавающей точкой, превращая ее в операцию сравнения целых чисел. Смещенный порядок называется также «характеристикой», ее можно считать целым положительным и беззнаковым числом.

Значение числа равно:

$$(-1)^S * 2^{E\text{-смещение}} * F_0,F_1F_2...F_n,\ F_0=1$$

где *n* для разных форматов равно 23, 52, или 63.

S	*Exponent*	*Fraction*

Отметим наличие в мантиссе бита единиц F_0. В коротком и длинном вещественных форматах бит F_0 при передачах чисел и хранении их в памяти не фигурирует. Это так называемый скрытый или неявный бит, который в нормализованных числах содержит единицу. Числа во временном вещественном формате имеют явный бит F_0. Такой формат позволяет несколько повысить скорость выполнения операций. Числа во временном вещественном формате называются еще числами с расширенной точностью.

Пример 3. Представим десятичное число -247,375 в вещественных форматах. Двоичный код его равен $-11110111{,}011_2$ и истинный порядок получается $+7_{10}=111_2$.Смещенный порядок в трех вещественных форматах равен: $134_{10}=10000110_2$, $1030_{10}=10000000110_2$ и $16390_{10}=100000000000110_2$.

	Знак	Порядок	Мантисса
Короткое вещественное	1	10000110	11101110110...0
Длинное вещественное	1	10000000110	11101110110...0
Временное вещественное	1	100000000000110	111101110110...0

Пример 4. Значение переменной *A* представлено в формате с плавающей точкой в шестнадцатеричной системе счисления A=C2F20000$_{16}$. Тип переменной *A* – single для языка программирования Паскаль. Найти десятичное значение числа *A*.

Решение.

Преобразуем шестнадцатеричное представление в двоичное. Вычленим из представления знак, порядок и мантиссу:

A=C2F20000$_{16}$= $11000010111100100000000000000000_2$.

Знак числа – старший бит – 1. Следовательно, искомое число отрицательное.

Порядок числа в памяти компьютера хранится в прямом коде со смещением. Найдем его: $P_{см} = 10000101_2$. Найдем реальный порядок, от порядка со смещением отнимем величину смещения 127_{10}:

$$P_{реал} = 10000101_2 - 01111111_2 = 110_2 = 6_{10}.$$

Следовательно, знак «двоичной» запятой в мантиссе нужно сместить вправо на 6 позиций.

Вспомним, что первый бит мантиссы – неявный, поэтому реальная мантисса имеет вид $M = 1{,}111001_2$.

Совмещаем знак, мантиссу и порядок: $A = -1111001_2 = -121{,}0_{10}$.

Вещественная арифметика

Пусть $a = \pm 0, m_a \cdot 2^{q_a}$, $b = \pm 0, m_b \cdot 2^{q_b}$ – два нормализованных двоичных числа, и $q_a \geq q_b$ (в противном случае мы можем просто поменять их местами). Результатом их *сложения* или *вычитания* будет являться следующее выражение: $c = (0, m_a \pm 0, m_b \cdot 2^{q_b - q_a}) \cdot 2^{q_a}$.

Порядок вычислений при этом таков:

1. Порядки чисел *a* и *b* выравниваются по большему из них (в нашем случае это q_a). Для этого мантисса числа *b* сдвигается на q_a-q_b разрядов вправо(часть значащих цифр при этом могут оказаться утерянными), а его порядок становится равным q_a.
2. Выполняется операция сложения (вычитания) над мантиссами с округлением по значению *n*+1-ой значащей цифры результата.
3. Мантисса результата должна быть нормализована (получившийся после нормализации порядок может отличаться от q_a как в меньшую, так и в большую сторону).

Все последующие упражнения будем выполнять в двоичной системе счисления без перевода в машинное представление!

Пример 5. Сложите два числа с плавающей запятой в двоичной системе счисления: $0{,}1001_2 \cdot 10_2^{101}$ и $0{,}111_2 \cdot 10_2^{-11}$.

Решение.

Перед сложением необходимо выровнять порядки – меньший порядок «приводится» к большему. В данном случае второе слагаемое приводится к виду: $0{,}00000000111_2 \cdot 10_2^{101}$. После чего выполняем сложение:

$$\begin{array}{r} 0{,}00000000111 \times 10^{101} \\ + \\ 0{,}10010000000 \times 10^{101} \\ \hline 0{,}10010000111 \times 10^{101} \end{array}$$

Если наша ячейка памяти имела бы восемь разрядов для хранения мантиссы, то последние три единичные разряда бы пришлось округлять.

При вычислении по данному алгоритму могут возникнуть следующие ошибки.

1. Потеря значащих цифр мантиссы у меньшего из чисел при выравнивании порядков.
2. Потеря крайней справа значащей цифры результата при сложении или вычитании мантисс.
3. Выход за границу допустимого диапазона значений того или иного вещественного типа при нормализации результата.

Рассмотрим теперь умножение и деление нормализованных чисел *a* и *b* в машинной арифметике (с ограниченным числом разрядов):

$$d = a \cdot b = (0, m_a \cdot 0, m_b) \cdot 2^{q_a + q_b};$$

$$f = a \div b = (0, m_a \div 0, m_b) \cdot 2^{q_a - q_b}$$

Для получения результата в данном случае производятся следующие действия, причем уже не важно, какое из двух данных чисел больше.

1. Мантиссы перемножаются (или одна делится на другую), при этом результат округляется до n значащих цифр (отметим, что при умножении может получиться $2n$ значащих цифры в мантиссе, а при делении – бесконечно много).
2. Порядки при умножении складываются, а при делении вычитаются.
3. При необходимости мантисса результата нормализуется.

Над мантиссами в арифметическом устройстве компьютер должен уметь выполнять все четыре операции (сложение, вычитание, умножение и деление), а также операции сдвига, тогда как над порядками производятся только действия сложения и вычитания. Для осуществления этих сложных для компьютера операций в его составе должен быть арифметический сопроцессор.

Пример 6. Выполнить умножение двух нормализованных двоичных числа, пользуясь алгоритмом машинной арифметики: $0{,}1001_2 \cdot 10_2^{101}$ и $0{,}111_2 \cdot 10_2^{-11}$.

Решение.

Складываем порядки этих чисел: 101−11=01.

Перемножаем мантиссы:

$$\begin{array}{r} 0{,}1001 \\ \times\ 0{,}111 \\ \hline 0{,}0111111 \end{array}$$

В результате получили число: $0{,}0111111_2 \cdot 10_2^{10}$. Приведем мантиссу в нормализованный вид: $0{,}111111_2 \cdot 10_2^{1}$.

Если бы в этом примере были большие значения порядков (например, 32 и 76), то в машинном представлении результата биты, выделенные под хранение порядка, не смогли бы вместить истинный порядок числа. Следовательно, операция умножения не будет произведена по причине *переполнения порядка*.

Пример 7. Выполнить деление двух нормализованных двоичных числа, пользуясь алгоритмом машинной арифметики: $0{,}1001_2 \cdot 10_2^{101}$ разделить на $0{,}111_2 \cdot 10_2^{-11}$.

Решение.

Найдем порядок результата: 101−(−11)=101+11=1000.

Делим мантиссы (столбиком!): $0{,}1001 \div 0{,}111 = 0{,}10(1001)$. Мантисса получилась нормализованной, но(!) периодической. Что же делать?

Когда говорят о точности представления десятичных вещественных чисел, нужно помнить следующее: десятичное число невозможно записать точно в любом из машинных вещественных типов. Объясняется это тем, что конечные десятичные дроби часто оказываются бесконечными периодическими двоичными дробями (см. упражнения по переводу правильных дробей из десятичной системы счисления в двоичную). Следовательно, в нормализованном виде такое число будет иметь бесконечную мантиссу и не может быть представлено точно. При записи подобной мантиссы в память компьютера число *округляется*.

Если под мантиссу отведено n бит и n+1 значащая цифра двоичной нормализованной мантиссы равна 0, то цифры начиная с n+1-ой просто отбрасываются, если же n+1-ая цифра равна единице, то к целому числу, составленных из n значащих цифр мантиссы, прибавляется единица.