Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Вятский государственный гуманитарный университет»

Дополнительная подготовка школьников по дисциплине «Информатика и информационные технологии»

Учебный модуль
Представление целых и вещественных чисел в компьютере

Е. В. Котельников

Киров
2011

СОДЕРЖАНИЕ

1. Ячейки памяти 3
2. Представление целых чисел в памяти компьютера 5
   2.1. Беззнаковые типы данных 5
   2.2. Знаковые типы данных 6
   2.3. Определение машинного представления десятичного числа 9
   2.4. Определение десятичного вида числа по его машинному представлению 10
   2.5. Диапазоны типов данных 12
   2.6. Целочисленные типы данных в Паскале 14
3. Битовые операции 15
4. Вещественные числа 21
   4.1. Стандарт IEEE 754-2008 21
   4.2. Особые числа 25
   4.3. Диапазоны типов данных 28
5. Задания для самостоятельного решения 31
Литература 33

1. Ячейки памяти

Основными ячейками компьютерной памяти являются бит и байт. *Бит* (*bit, binary digit* – двоичная цифра) – это минимальная ячейка памяти, которая может хранить один двоичный разряд (значения «0» или «1»). Последовательность смежных двоичных цифр длиной в 8 бит[1] получила название *байт* (*byte, binary term*[2]). Байт – это наименьшая адресуемая компьютером единица информации.

Максимально возможное число, которое размещается в байте, в двоичном формате будет состоять из восьми единиц, а в шестнадцатеричном – из двух цифр FF:

$$1111\ 1111_2 = FF_{16} = 255_{10}.$$

Слово (*word*) – это совокупность нескольких байт, которую процессор обрабатывает как единое целое. Процессор производит операции именно над словами, например, считывает из памяти слово целиком и записывает в память, складывает или умножает два слова и т. д.

Размер слова зависит от разрядности процессора, которая в свою очередь определяется разрядностью основных регистров процессора. Например, у 16-разрядного процессора Intel 8086 основные регистры были 16-разрядные и размер слова был равен двум байтам, а современный 32-разрядный процессор Intel Core имеет 8 основных регистров разрядностью 32 бита, поэтому размер слова у него – четыре байта.

Максимально возможное целое число, которое можно поместить в двухбайтовое слово:

$$1111\ 1111\ 1111\ 1111_2 = FFFF_{16} = 65535_{10}.$$

Для 32-разрядного слова максимальное число будет следующим:

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = FFFF\ FFFF_{16} = 4\ 294\ 967\ 295_{10}.$$

Большинство современных процессоров 32- или 64-разрядные.

[1] Термин «байт» появился в 50-х гг. XX века и сначала размер байта в разных компьютерах отличался (например, были байты в 6, 8, 9, 12 бит и др.). Примерно в 70-е гг. под байтом стали понимать всегда 8 бит (иногда такой набор ещё называют «октет»).

[2] Ещё одна версия происхождения слова «байт» связана с тем, что первоначально последовательность бит называли *bite* – кусок, а затем, чтобы не путать со словом *bit*, заменили *i* на *y*.

Следует отметить, что разрядность ячеек памяти, начиная с байта, увеличивается каждый раз вдвое, т. е. используются ячейки размером 1 байт, 2 байта, 4 байта, 8 байт, 16 байт, но не применяются ячейки 7- или 9-байтовые[1].

[1] Это утверждение почти всегда справедливо при представлении целых чисел, но, например, существует 10-байтовая ячейка памяти для вещественных чисел (соответствующий тип данных называется Extended).

2. Представление целых чисел в памяти компьютера

2.1. Беззнаковые типы данных

Машинное представление чисел неразрывно связано с понятием типа данных. Напомним его определение. *Тип данных* – это характеристика данных, определяющая диапазон их значений, набор допустимых операций над ними и внутреннее представление в памяти компьютера.

Типы данных для хранения целых чисел делятся на беззнаковые и знаковые. В *беззнаковых типах данных* можно хранить только неотрицательные целые числа. *Знаковые типы* предназначены для хранения и неотрицательных, и отрицательных чисел. Рассмотрим сначала беззнаковые типы.

Числа, представленные беззнаковыми типами, хранятся в памяти в двоичном виде, поэтому для получения представления беззнакового десятичного числа в памяти компьютера нужно просто перевести это число в двоичную систему счисления и дописать незначащие нули слева до требуемой разрядности.

Пример. Пусть имеется десятичное целое число 100. В каком типе данных его можно представить? Сколько требуется бит для хранения числа и каково будет его внутреннее (машинное) представление?

Решение. Для начала нужно перевести это число в двоичную систему счисления:

$$100_{10} = 110\ 0100_2 = 64_{16}.$$

По количеству полученных бит (7) видно, что для хранения такого числа понадобится минимум 8-разрядная ячейка памяти. Для Паскаля это беззнаковый тип `Byte`.

Чтобы получить внутреннее представление числа в памяти, возьмем двоичный вид и допишем слева незначащий ноль до разрядности 8 бит:

$$100_{10} = \underline{0}110\ 0100_2 = 64_{16}.$$

Можно это число разместить и в 16-битном беззнаковом типе (например, `Word`), при этом придется дописывать уже 9 незначащих нулей:

$$100_{10} = \underline{0000}\ \underline{0000}\ \underline{0}110\ 0100_2 = 0064_{16}.$$

2.2. Знаковые типы данных

В *знаковом целом типе* могут быть представлены положительные и отрицательные числа, а также ноль. Положительные числа и ноль в знаковом типе хранятся точно так же, как в беззнаковом – в двоичном виде. Знаковые отрицательные числа хранятся в компьютере в *дополнительном коде*. Рассмотрим следующий алгоритм.

Алгоритм 1. Первый способ перевода отрицательного числа в дополнительный код.

1. Перевести модуль отрицательного числа в двоичный код.
2. Дополнить двоичное число незначащими нулями до требуемой разрядности (в зависимости от используемого типа данных).
3. Проинвертировать полученное двоичное число (т. е. нули становятся единицами, а единицы – нулями).
4. К полученному двоичному числу арифметически прибавить 1.

Пример. Пусть имеется десятичное отрицательное число –100. В каком машинном формате его можно представить?

Решение. Наше число отрицательное, значит, его нужно перевести в дополнительный код (алгоритм 1):

1) $|-100| = 100_{10} = 110\ 0100_2$.

2) На предыдущем шаге получилось число разрядностью 7 бит. Но такого типа данных нет, поэтому используем 8 бит, добавляя слева ноль:

$$\underline{\mathbf{0}}110\ 0100_2.$$

3) Инвертируем: $0110\ 0100_2 \rightarrow 1001\ 1011_2$.

4) К полученному числу прибавляем 1:

$$\begin{array}{rr} & 1001\ 1011 \\ + & \qquad\quad 1 \\ \hline & 1001\ 1100 \end{array}$$

Ответ: $1001\ 1100_2 = 9C_{16}$.

Задание. Самостоятельно найдите машинное представление числа –100 в 16-разрядном знаковом типе и в 32-разрядном знаковом типе.

Ответ: $FF9C_{16}$, $FFFF\ FF9C_{16}$.

Задание. Самостоятельно найдите машинное представление числа –1000 в 16-разрядном знаковом типе.

Ответ: $F828_{16}$.

Существует ещё один алгоритм перевода в дополнительный код, связанный с названием кода: «дополнительный» означает, что число в дополнительном коде дополняет модуль исходного отрицательного числа до 2^n, где n – разрядность используемого типа данных.

Алгоритм 2. Второй способ перевода отрицательного числа в дополнительный код.

Обозначим n – разрядность используемого типа данных, X – отрицательное число.

1. Найдем разность:

$$y = 2^n - |X|.$$

2. Переведем y в двоичную систему счисления.

Пример. Найти дополнительный код числа –100 для 8-разрядного типа данных.

Решение. Воспользуемся алгоритмом 2.

1) $y = 2^n - |X| = 2^8 - |-100| = 256 - 100 = 156_{10}$.

2) $156_{10} = 1001\ 1100_2$.

Ответ: дополнительный код числа –100 в 8-разрядного типе данных равен $1001\ 1100_2 = 9C_{16}$.

Алгоритм 3. Первый способ перевода числа из дополнительного кода в десятичный вид.

1. Инвертируем двоичное число.
2. Прибавляем арифметически к нему единицу.
3. Переводим полученное двоичное число в десятичную систему счисления и приписываем знак «минус».

Пример. Дано двоичное число в дополнительном коде $1001\ 1100_2$. Требуется перевести его в десятичную систему счисления.

Решение.

1) Инвертируем число: $1001\ 1100_2 \rightarrow 0110\ 0011_2$.

2) Прибавляем к полученному числу единицу:

$$\begin{array}{rr} & 0110\ 0011 \\ + & \qquad\quad 1 \\ \hline & 0110\ 0100 \end{array}$$

3) Переводим число в десятичную систему счисления: $0110\ 0100_2 = 100_{10}$ и приписываем знак минус.

Ответ: -100_{10}.

По аналогии с получением дополнительного кода существует второй способ нахождения десятичного числа по дополнительному коду.

Алгоритм 4. Второй способ перевода числа из дополнительного кода в десятичный вид.

Обозначим n – разрядность используемого типа данных, X – искомое отрицательное число.

1. Переводим двоичное число в десятичную систему счисления. Полученное число обозначаем y.
2. Находим разность: $|X| = 2^n - y$.
3. Приписываем к полученному результату знак «минус».

Пример. Найти десятичное представление двоичного числа в дополнительном коде $1001\ 1100_2$.

Решение. Воспользуемся алгоритмом 4.

1) $1001\ 1100_2 = 156_{10}$.

2) $|X| = 256 - 156 = 100_{10}$.

3) Приписываем знак «минус».

Ответ: –100.

2.3. Определение машинного представления десятичного числа

Сейчас можно обобщить наши сведения и рассмотреть алгоритм нахождения внутреннего представления числа в памяти компьютера.

Алгоритм 5. Определение машинного представления десятичного числа (рис. 1).

1. Выбрать тип данных, который будет использоваться для машинного представления.
2. Если выбран беззнаковый тип, то определяем знак десятичного числа. Переход на п.4.
3. Если выбран знаковый тип, то определяем знак десятичного числа. Переход на п.6
4. Если знак десятичного числа «минус», то представление такого числа в беззнаковом типе невозможно. Завершение работы. Иначе переход на п.5.
5. Если знак десятичного числа «плюс» или число равно 0, то переводим десятичное число в двоичный вид и дополняем нулями до требуемой разрядности. Завершение работы.
6. Если знак десятичного числа «минус», то число должно быть представлено в дополнительном коде и нужно использовать алгоритм 1 или 2 для перевода из десятичной системы счисления в дополнительный код. Завершение работы. Иначе переход на п.5.

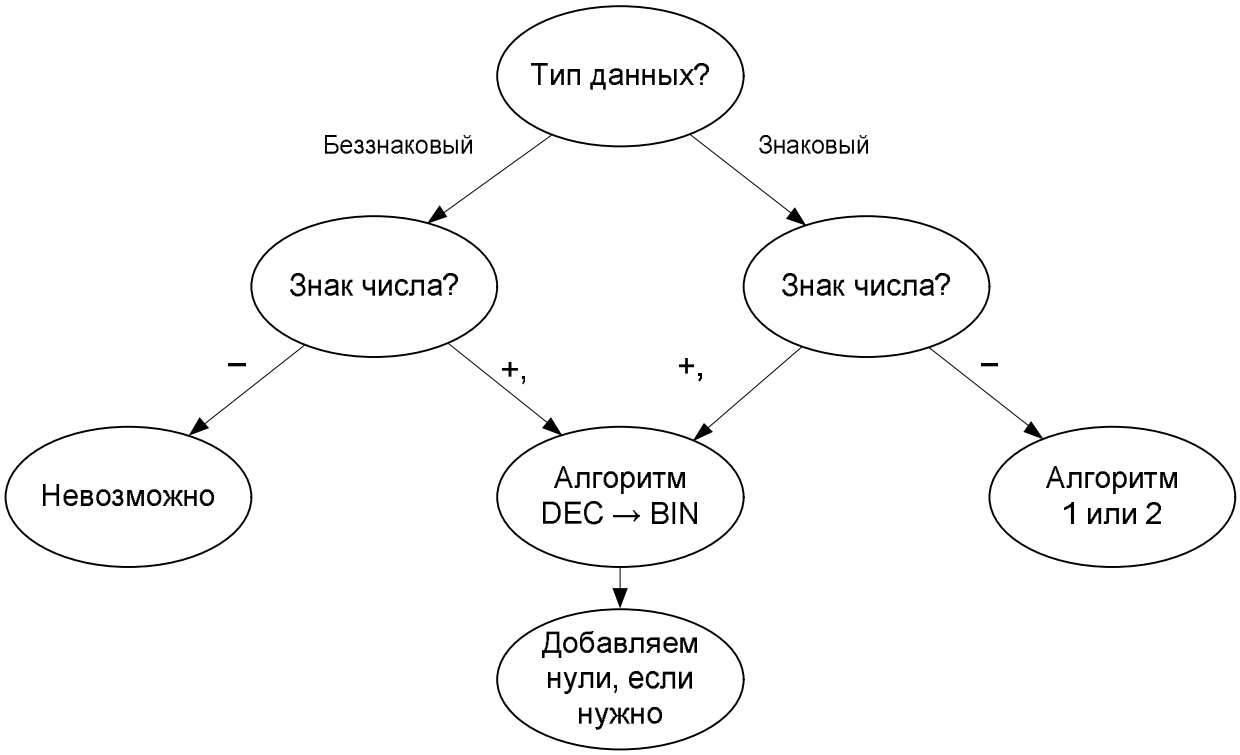


Рис. 1. Алгоритм 5 – определение машинного представления десятичного числа

2.4. Определение десятичного вида числа по его машинному представлению

Выше мы рассматривали проблему представления целых чисел в памяти компьютера. А что если стоит обратная задача: по машинному представлению числа определить его десятичную форму?

Оказывается, просто по двоичному виду числа в памяти компьютера невозможно определить, какое число там представлено. Мало того, нельзя даже сказать, какого рода данные хранятся – числа, символы, графика или звук. Это является следствием одного из принципов фон Неймана: все данные хранятся в памяти компьютера в двоичном виде.

Чтобы правильно интерпретировать двоичное число, представленное в памяти компьютера, *нужно обязательно знать тип данных*. Так, например, число $FC18_{16}$ в типе `SmallInt` интерпретируется как –1000, так как тип знаковый. С другой стороны, то же самое число в беззнаковом типе `Word` будет интерпретироваться как 64536. То есть один и тот же двоичный код может представлять разные числа, в зависимости от заданного типа.

Введем одно важное правило: если число представлено в памяти в знаковом типе, то старший (крайний левый) бит отвечает за знак числа: если

старший бит равен 0, то число положительное, если 1, то отрицательное. Поэтому в знаковых типах старший бит называется *Signum* (лат. *знак*).

Рассмотрим следующий алгоритм.

Алгоритм 6. Определение десятичного вида числа по его машинному представлению (см. рис. 2).

1. Определить тип данных, который использовался для машинного представления.
2. Если тип беззнаковый, то просто переводим из двоичного вида в десятичный.
3. Если тип знаковый, то проверяем значение старшего бита.
4. Если старший бит равен 0, то просто переводим из двоичного вида в десятичный.
5. Если старший бит равен 1, то число представлено в дополнительном коде и нужно использовать алгоритм 3 или 4 для перевода из дополнительного кода в десятичный вид.

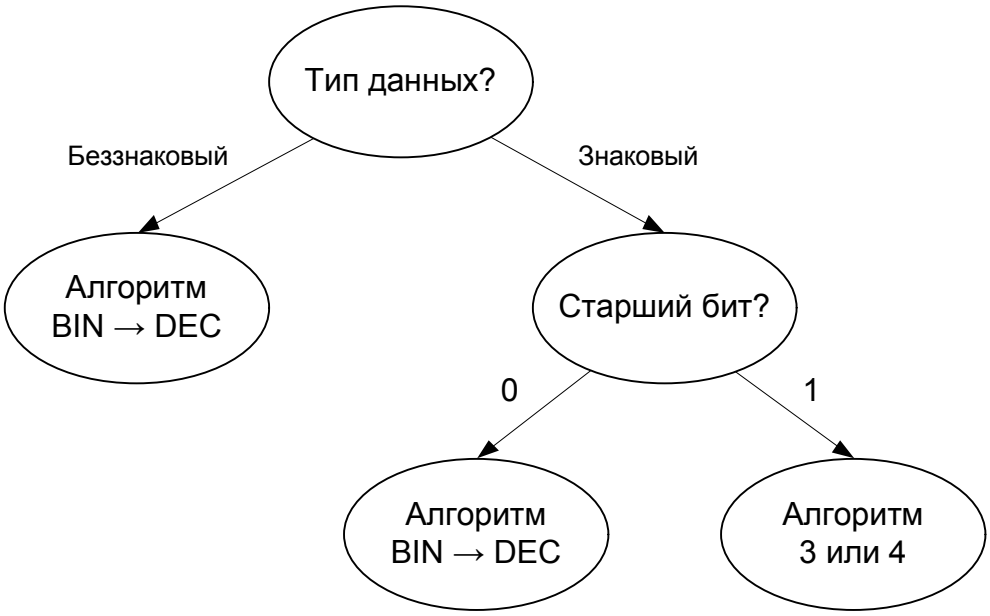


Рис. 2. Алгоритм 6 – определение десятичного вида числа по его машинному представлению

Каким образом определить используемый тип данных? Это можно сделать либо на основании априорной информации (например, если программе-отладчику, которой нужно отобразить значение ячейки памяти,

доступен исходный код отлаживаемой программы), либо при помощи косвенный данных (например, какие машинные команды используются для работы с интересующей ячейкой памяти – некоторые машинные команды имеют знаковые и беззнаковые варианты).

2.5. Диапазоны типов данных

Чтобы уметь правильно работать с типом данных, нужно точно знать его диапазон. Если, например, при сложении двух чисел результат окажется за пределами диапазона используемого типа, значение, которое сохранится в памяти, будет неверным. Такие ситуации надо уметь отслеживать и предупреждать.

Для определения диапазона целого типа данных (т. е. нахождения минимального и максимального значений, представимых в этом типе) достаточно знать его разрядность n и является тип знаковым или беззнаковым.

Независимо от «знаковости» типа, общее количество чисел, представимых в этом типе, равно 2^n:

$$\textbf{\textit{общее количество чисел в типе}} = 2^n.$$

Для беззнаковых типов минимальное значение представляется всеми нулями в двоичной записи числа и всегда равно 0.

Чтобы получить максимальное значение, нужно заполнить все разряды единицами и перевести полученное двоичное число в десятичный вид. Например, для 8-разрядного беззнакового типа максимальное число будет равно $1111\ 1111_2 = 255_{10}$. В общем случае, максимальное число, представимое в n разрядах, будет равно $2^n–1$. Таким образом, получаем:

$$\textbf{\textit{диапазон беззнакового типа}}: [0\ ..\ 2^n–1].$$

Для вывода формулы диапазона знаковых типов рассмотрим некоторый 4-разрядный тип данных. Отложим $2^4 = 16$ двоичных чисел, представимых в этом типе, на числовом круге (см. рис. 3). Каждому двоичному числу снаружи круга припишем его десятичный эквивалент в том случае, если бы тип был беззнаковый, и внутри круга, если тип знаковый. Десятичное представление чисел можно находить, пользуясь алгоритмом 6 (см. рис. 2).

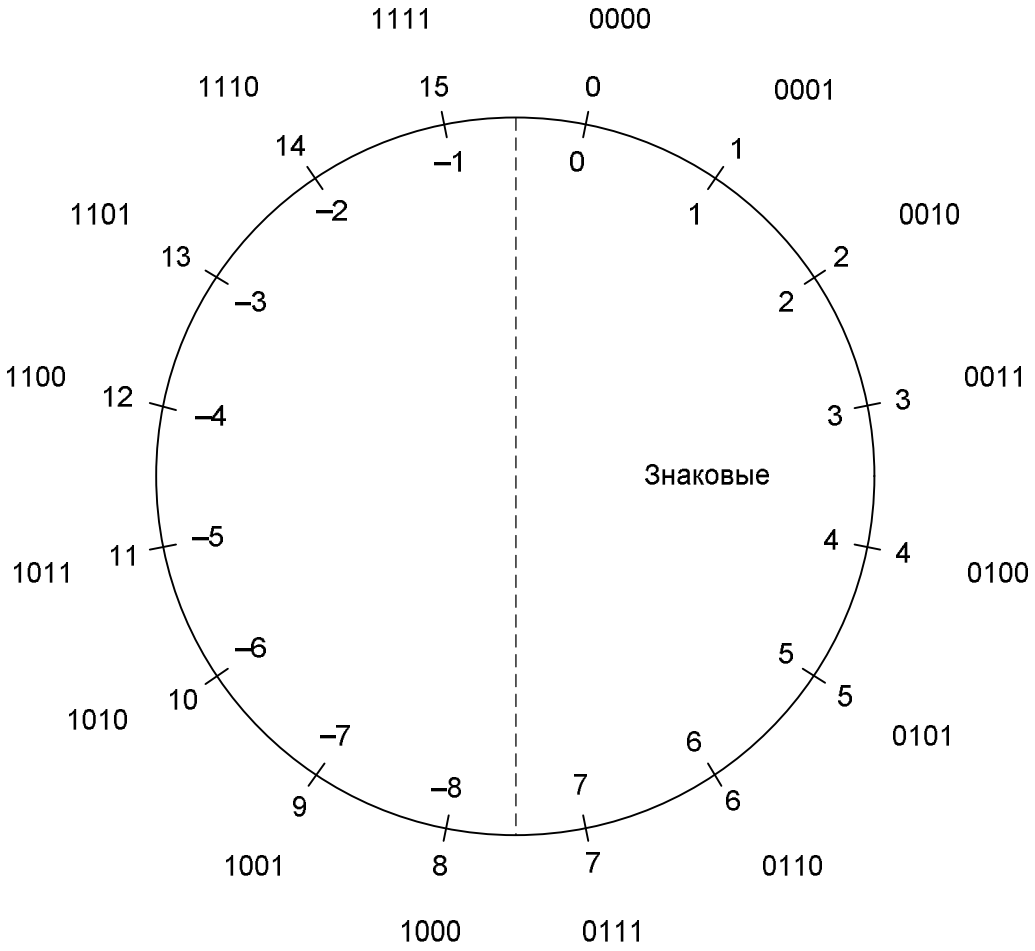


Рис. 3. Числовой круг для 4-разрядных типов

Смотря на рис. 3, можно заметить, что круг для знакового типа поделен на две равные части – неотрицательные числа (справа, там где старший бит равен 0) и отрицательные числа (слева, старший бит равен 1). Таким образом, диапазон знакового типа делится пополам (количество чисел в каждой $2^n / 2$) – одна половина отводится на положительные числа и ноль, другая – на отрицательные числа:

диапазон знакового типа: $[-2^{n-1} .. 2^{n-1}-1]$.

Заметьте, что половина положительных десятичных чисел из всего диапазона беззнакового типа совпадает по своему внутреннему представлению в памяти компьютера с отрицательными числами.

2.6. Целочисленные типы данных в Паскале

В таблице ниже представлены целочисленные типы, используемые в среде программирования PascalABC.Net. Заметим, что поскольку среда программирования 32-разрядная (предназначена для работы на 32-разрядных процессорах), то базовый целый тип Integer является 4-байтовым.

Таблица 1. Целые типы данных в PascalABC.Net

Тип	Разрядность	Знак	Диапазон значений
`Byte`	8	Беззнаковый	0..255
`ShortInt`	8	Знаковый	–128..127
`Word`	16	Беззнаковый	0..65 535
`SmallInt`	16	Знаковый	–32 768..32 767
`Longword`, `Cardinal`	32	Беззнаковый	0..4 294 967 295
`Integer`, `Longint`	32	Знаковый	–2 147 483 648..2 147 483 647
`Uint64`	64	Беззнаковый	-2^{63} .. $2^{63}-1$
`Int64`	64	Знаковый	0 .. $2^{64}-1$

Задание. Что получится в результате выполнения программы на Паскале? Самостоятельно постройте числовой круг для 8-разрядных типов и объясните на его основе ответ.

```
var x: Byte;
begin
  x := 200 + 200;
  writeln('x=', x);
end.
```

3. Битовые операции

Битовые (поразрядные) операции – это операции, выполняемые отдельно с каждым битом машинного представления числа. К битовым операциям относят применение к битам логических функций AND, OR, XOR, NOT, а также команды сдвига всех битов числа влево (SHL) или вправо (SHR).

Выполнение битовой операции (AND, OR, XOR) над двумя операндами приведет к тому, что соответствующая логическая функция будет обрабатывать по два бита из каждого операнда: первый с первым, второй со вторым и т. д.

При использовании битовых операций вводится понятие *маски* (mask) – числа, которое применяется для выделения нужных бит в переменной.

Битовая операция AND (И, логическое умножение), примененная к двум битам, дает в результате 1, только если оба бита были равны 1. Операция AND используется чаще всего для проверки значений отдельных бит. В этом случае в двоичной записи маски все биты, кроме проверяемого, обнуляют. Также AND применяется для обнуления бит. При этом все биты в маске, кроме обнуляемого, устанавливаются в 1.

	1	0	1	1	0	0	1	0
AND	1	0	0	0	1	0	1	1
	1	0	0	0	0	0	1	0

Пример. Определить значение 5-го бита (нумерация с 0) во введенном с клавиатуры числе.

```
program Test_AND_1;
var a, b, mask : integer;
begin
  writeln('Введите число:');
  readln(a);
  mask := 32;     // Маска - двоичное число 0010 0000
                  // (единица в 5-м бите)
  b := a AND mask;          // Накладываем маску на a,
                            // результат присваиваем b
  if b <> 0 then writeln('Бит 5 = 1')
            else writeln('Бит 5 = 0');
end.
```

Пример. Обнулить 3-й бит во введенном с клавиатуры числе.

```
program Test_AND_2;
var a, b, mask1, mask2 : integer;
begin
  writeln('Введите число:');
  readln(a);

  // Проверка 3-го бита
  mask1 := 8;         // Маска - двоичное число 0000 1000
                      // (единица в 3-м бите)
  b := a AND mask1;   // Накладываем маску на a,
                      // результат присваиваем b

  if b <> 0 then writeln('Бит 3 = 1')
            else writeln('Бит 3 = 0');

  // Обнуление 3-го бита
  // (независимо от результатов проверки)
  mask2 := 247;             // Маска - двоичное число
                            // 1111 0111 (ноль в 3-м бите)
  a := a AND mask2;         // Накладываем маску на a
                            // результат присваиваем b
  writeln('Бит 3 обнулен!');

  // Проверка 3-го бита
  b := a AND mask1;         // Накладываем маску на a
                            // результат присваиваем b

  if b <> 0 then writeln('Бит 3 = 1')
            else writeln('Бит 3 = 0');
end.
```

Битовая операция OR (ИЛИ, логическое сложение), примененная к двум битам, дает в результате 1, если хотя бы один бит был равен 1. Операция OR используется для установки отдельных бит в единицу, при этом в маске устанавливаемые биты также должны быть равны единице.

$$\begin{array}{cc} & 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0 \\ \text{OR} & 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1 \\ \hline & 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1 \end{array}$$

Пример. Установить в единицу 6-й бит во введенном с клавиатуры числе.

```
program Test_OR;
var a, b, mask : integer;
begin
  writeln('Введите число:');
  readln(a);

  // Проверка 6-го бита
  mask := 64;     // Маска - двоичное число 0100 0000
                  // (единица в 6-м бите)
  b := a AND mask;          // Накладываем маску на a,
                            // результат присваиваем b
  if b <> 0 then writeln('Бит 6 = 1')
            else writeln('Бит 6 = 0');

  // Установка 6-го бита
  // (независимо от результатов проверки)
  a := a OR mask;           // Накладываем маску на a,
                            // результат присваиваем a
  writeln('Бит 6 установлен!');

  // Проверка 3-го бита
  b := a AND mask;          // Накладываем маску на a,
                            // результат присваиваем b
  if b <> 0 then writeln('Бит 6 = 1')
            else writeln('Бит 6 = 0');
end.
```

Битовая операция XOR (Исключающее ИЛИ, сложение по модулю 2), примененная к двум битам, дает в результате 1, если биты не были равны друг другу. Операция XOR используется для инвертирования значений отдельных бит, соответствующие биты в маске должны быть установлены в 1. Также можно применять XOR для обнуления переменных – с этой целью выполняют операцию XOR переменной с самой собой (заметим, что такой способ обнуления работает быстрее обычного присваивания переменной нуля).

$$\begin{array}{rcccccccc} & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ \text{XOR} & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ \hline & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}$$

Пример. Проинвертировать значение 4-го бита во введенном с клавиатуры числе.

```
program Test_XOR_1;
var a, b, mask : integer;
begin
  writeln('Введите число:');
  readln(a);

  // Проверка 4-го бита
  mask := 16;     // Маска - двоичное число 0001 0000
                  // (единица в 4-м бите)
  b := a AND mask;         // Накладываем маску
  if b <> 0 then writeln('Бит 4 = 1')
            else writeln('Бит 4 = 0');

  // Инверсия 4-го бита
  // (независимо от результатов проверки)
  a := a XOR mask;          // Накладываем маску на a,
                            // результат присваиваем a
  writeln('Бит 4 инвертирован!');

  // Проверка 4-го бита
  b := a AND mask;         // Накладываем маску
  if b <> 0 then writeln('Бит 4 = 1')
            else writeln('Бит 4 = 0');
end.
```

Битовая операция NOT (НЕ, инверсия), примененная к одному биту, дает в результате бит с противоположным значением. Операция NOT применяется, когда необходимо инвертировать все биты числа.

NOT 10110010 = 01001101.

Операции сдвига влево (Shift Left – SHR) и *вправо* (Shift Right – SHR) выполняют смещение всех битов числа на заданное количество разрядов влево или вправо. Применяются эти операции для быстрого умножения и деления целых чисел на степени двойки, а также для вспомогательных действий при других битовых операциях.

Пример. Сдвиг числа на 3 разряда влево:

10110010 SHL 3 = 10010000.

Пример. Сдвиг числа на 2 разряда вправо:

10110010 SHR 2 = 00101100.

Пример. Написать программу умножения и деления введенного числа на степени двойки.

```
program shl_shr;
var a, proiz, r: integer;
begin
  writeln('Введите число от 1 до 1000:');
  readln(a);

  // Умножение
  r := 1;
  proiz := 1;
  while (proiz <= 100000) do
  begin
    proiz := a shl r;     // Сдвиг на r разрядов влево =
                          // умножение на 2^r
    writeln(a, ' * 2^', r, ' = ', proiz);
    inc(r);               // Увеличиваем r на 1
  end;
  writeln;

  // Деление
  r := 1;
  proiz := 1;
  while (proiz > 0) do
  begin
    proiz := a shr r;    // Сдвиг на r разрядов вправо =
                         // деление на 2^r
    writeln(a, ' / 2^', r, ' = ', proiz);
    inc(r);              // Увеличиваем r на 1
  end;
end.
```

Пример. Узнать значение определенного бита в числе. Число и номер бита вводятся с клавиатуры.

```
program CheckBit;
var a, b, bit, mask : integer;
begin
  writeln('Введите число:');
  readln(a);
  writeln('Введите номер бита (0-31):');
  readln(bit);
  mask := 1 shl bit;      // Формируем маску –
                  // сдвигаем 1 на bit разрядов влево
  b := a AND mask;        // Накладываем маску
  if b <> 0 then writeln('Bit ', bit, ' = 1')
  else writeln('Bit ', bit, ' = 0');
end.
```

Следует отметить, что во многих языках программирования присутствуют как битовые операции, так и логические[1]. Важно понимать разницу между этими двумя видами операций. Битовые операции работают с отдельными битами чисел, выполняя свою функцию поразрядно, а логические операции обрабатывают число как единое целое, считая его значение либо *true* (истиной), *либо* false (ложью).

В некоторых языках, например, в Паскале, они обозначаются одинаково (AND, OR, XOR, NOT), в других применяются разные обозначение, например в Си для битовых операций используются одинарные символы «&», «|», а для логических – двойные «&&», «| |». Каким образом компилятор Паскаля определяет, какой из двух видов операций сейчас используется? В соответствии с типами операндов. Если операнды целого типа (например, `integer`), то применяется битовая операция, если операнды имеют тип `boolean`, то используется логическая операция.

[1] Операции AND, OR, XOR, NOT могут быть как битовыми, так и логическими, операции сдвига всегда битовые.

4. Вещественные числа

4.1. Стандарт IEEE 754-2008

Для представления вещественных чисел в памяти компьютера разработан стандарт IEEE 754-2008[1], в котором описаны форматы представления чисел, алгоритмы округления, операции над числами, обработка исключений (деление на ноль, переполнение и др.).

В этом стандарте используется форма представления вещественных чисел с *плавающей запятой*. В этой форме вещественное число X записывается в следующем виде:

$$X = \pm m \cdot q^{\pm p},$$

где m – мантисса числа (лат. *добавление*, *прибавка*), q – основание системы счисления, p – порядок.

Пример. Представить число $123{,}45_{10}$ в форме с плавающей запятой.

$$123{,}45 = 12{,}345 \cdot 10^1 = 1{,}2345 \cdot 10^2.$$

В этом примере числа 12,345 и 1,2345 являются мантиссой, а 1 и 2 – порядком.

Из примера видно, что десятичная запятая может перемещаться («плавать») с одновременным изменением порядка: если запятая перемещается влево на n разрядов, то порядок нужно увеличить на n единиц, если запятая перемещается вправо на n разрядов, то порядок нужно уменьшить на n единиц.

Для представления чисел в памяти основание системы счисления принимается равным 2.

Пример. Представить число $10101{,}101_2$ в форме с плавающей запятой.

$$10101{,}101 = 101{,}01101 \cdot 2^2 = 1010110{,}1 \cdot 2^{-2}.$$

[1] IEEE (Institute of Electrical and Electronics Engineers) – Институт инженеров по электронике и электротехнике, некоммерческая международная ассоциация, одной из функцией которой является разработка стандартов в области информатики.

Для определенности вещественные числа в компьютерах представляются в *нормализованном виде*, т. е. запятая расположена справа от первой ненулевой цифры.

Пример.
$0{,}00101_2$ – ненормализованное двоичное число,
$0{,}0101 \cdot 2^{-1}$ – ненормализованное двоичное число,
$1{,}01 \cdot 2^{-3}$ – нормализованное двоичное число.

В двоичном нормализованном представлении числа старший бит, расположенный слева от запятой, всегда равен 1, поэтому его можно не хранить (т. н. «скрытая единица»), а диапазон нормализованных мантисс получается следующим: $1 \le m < 2$.

Рассмотрим основные форматы представления вещественных чисел, описанные в стандарте IEEE 754-2008. Они отличаются размером ячейки памяти, отводимой для одного числа:

1) формат одинарной точности (single precision, 4 байт);
2) формат двойной точности (double precision, 8 байт);
3) формат расширенной точности (extended precision, 10 байт).

В *формате одинарной точности* под число отводится 32 бита, которые распределены следующим образом:

Рис. 4. Формат одинарной точности

В этом формате указывается только знак всего числа: если бит знака равен 0, число положительное, если 1 – отрицательное. Знак порядка не указывается, так как применяется *смещенный порядок sp*, который всегда неотрицателен. Смещенный порядок получается прибавлением к исходному порядку константы 127:

$$sp = p + 127. \qquad (1)$$

При 8-битной ширине поля смещенного порядка диапазон порядков:

$$127 \le p \le 128.$$

Таким образом, для представления вещественного числа X в формате одинарной точности используется следующая формула (в десятичном виде):

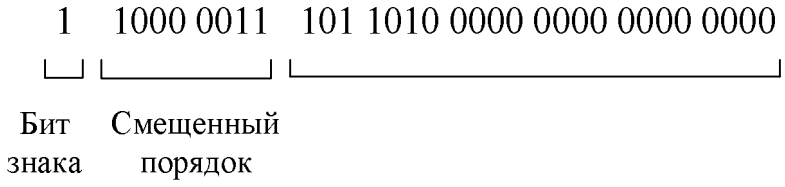
$$X = \pm 1,m \cdot 2^{sp-127}. \quad (2)$$

Алгоритм 7. Перевод десятичного вещественного числа в формат одинарной точности.

1. Перевести вещественное число в двоичный вид.
2. Нормализовать полученное двоичное число.
3. Отбросить скрытую единицу.
4. Определить мантиссу в двоичном виде.
5. Определить порядок в десятичном виде.
6. Вычислить смещенный порядок по формуле (1) и перевести его в двоичный вид.
7. Определить значение старшего бита, соответствующего знаку числа.
8. Записать ответ в соответствии с рисунком 4.

Пример. Представить в формате одинарной точности число $-27{,}25_{10}$.

1) Переводим десятичное число в двоичный вид: $27{,}25_{10} = 11011{,}01_2$.
2) Нормализуем двоичное число: $11011{,}01_2 = 1{,}101101 \cdot 2^4$.
3) Отбрасываем скрытую единицу.
4) Мантисса: $m = 101101_2$.
5) Порядок: $p = 4_{10}$.
6) Находим смещенный порядок: $sp = 4 + 127 = 131_{10} = 1000\ 0011_2$.
7) Поскольку знак «минус», то старший бит равен 1.
8) Ответ в двоичном виде:



В шестнадцатеричном виде: $-27{,}25_{10} = \text{C1DA } 0000_{16}$.

Алгоритм 8. Перевод машинного представления нормализованного вещественного числа одинарной точности в десятичный вид.

1) Отделить старший бит в двоичном представлении и определить знак числа: «плюс», если старший бит равен 1, «минус», если старший бит равен 0.
2) Отделить двоичное представление смещенного порядка (8 бит).
3) Перевести смещенный порядок в десятичный вид.
4) Вычислить порядок искомого числа по формуле:

$$p = sp - 127.$$

5) Выделяем мантиссу, прибавляем к ней слева скрытую единицу и двоичную запятую.
6) Переводим полученную мантиссу в десятичный вид.
7) Записываем ответ по формуле (2).

Пример. Найти вещественное число, соответствующее машинному представлению: 1100 0001 1101 1010 0000 0000 0000 0000.

1) Старший бит равен 1, следовательно, знак числа «минус».
2) Выделяем смещенный порядок: $sp = 1000\ 0011_2$.
3) Переводим смещенный порядок в десятичный вид:

$$sp = 1000\ 0011_2 = 131_{10}.$$

4) Вычисляем порядок: $p = 131 - 127 = 4$.
5) Выделяем мантиссу, прибавляем к ней слева скрытую единицу и двоичную запятую: $m = 1{,}101101_2$.
6) Переводим мантиссу в десятичный вид:

$$m = 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} + 1 \cdot 2^{-6} = 1{,}703125_{10}.$$

7) Ответ: $-1{,}703125 \cdot 2^4 = -27{,}25_{10}$.

В других форматах стандарта IEEE 754-2008 следующие значения ширины полей:

– *формат двойной точности*: под все число отводится 64 бита (8 байт), из них 1 бит – знак, 11 бит – смещенный порядок, 52 бита – мантисса, смещенный порядок вычисляется по формуле:

$$sp = p + 1023;$$

– *формат расширенной точности*: под все число отводится 80 бит (10 байт), из них 1 бит – знак, 15 бит – смещенный порядок, 64 бита – мантисса, смещенный порядок вычисляется по формуле:

$$sp = p + 16383.$$

4.2. Особые числа

В стандарте IEEE 754-2008 определены т. н. особые числа, которые представляют ноль, бесконечность, неопределенный результат операции и числа, близкие к нулю.

1. *Ноль*.

В отличие от машинного представления целых чисел, при хранении вещественных чисел существует два варианта нуля – положительный и отрицательный (все примеры далее будем приводить только для одинарного формата).

Положительный ноль:

0 0000 0000 $000\ 0000\ 0000\ 0000\ 0000\ 0000_2$ = 0000 0000

Бит знака — Смещенный порядок — Мантисса

Отрицательный ноль:

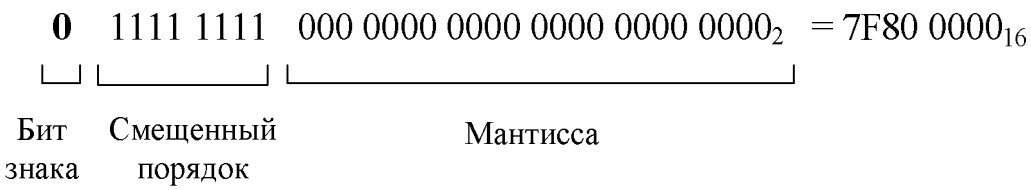
1 0000 0000 $000\ 0000\ 0000\ 0000\ 0000\ 0000_2$ = 8000 0000

Бит знака — Смещенный порядок — Мантисса

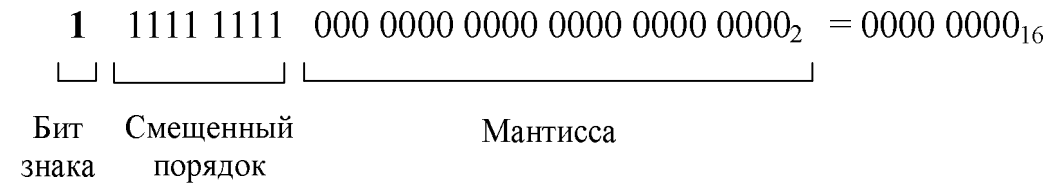
2. *Бесконечность*.

Вариантов представления бесконечности также два, положительный и отрицательный. В этом представлении в смещенном порядке содержатся все единицы.

Положительная бесконечность:



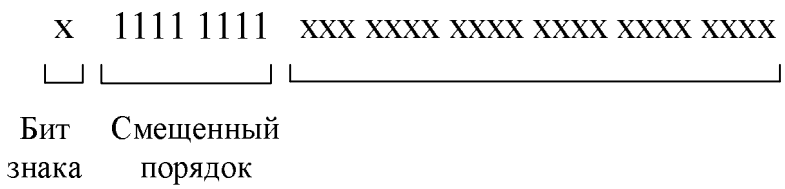
Отрицательная бесконечность:



3. *NaN (Not a Number)*.

Особые числа NaN (дословный перевод «не-числа») используются для представления неопределенных или некорректных результатов, например результатов операций деления ноля на ноль, умножения нуля на бесконечность, извлечения квадратного корня из отрицательного числа.

Число будет являться NaN, если в поле смещенного порядка хранятся все единицы (исключая случай, когда в мантиссе все нули):

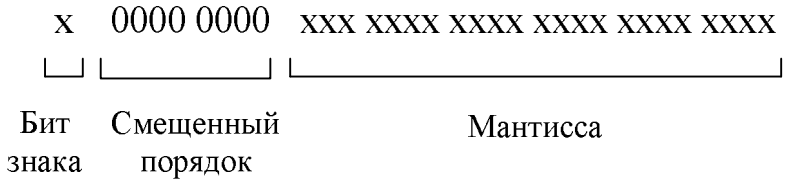


Символ *x* на рисунке говорит о том, что данный бит не имеет значения.

4. *Денормализованные числа*.

Особенность представления вещественных чисел со скрытой единицей в том, что имеется довольно большой разрыв между нулем и ближайшим к нему числом. В некоторых случаях это обстоятельство может приводить к ошибкам при работе с малыми величинами. Чтобы снизить вероятность таких ошибок, были введены т. н. денормализованные числа, при хранении которых скрытая единица не используется.

Признаком того, что число денормализованное, является наличие всех нулей в поле смещенного порядка (за исключением случая всех нулей в мантиссе):



Алгоритм 9. Перевод машинного представления денормализованного вещественного числа одинарной точности в десятичный вид.

1) Отделить старший бит в двоичном представлении и определить знак числа: «плюс», если старший бит равен 1, «минус», если старший бит равен 0.
2) Отделить двоичное представление смещенного порядка (8 бит) – для денормализованного числа должен быть равен $0000\ 0000_2$.
3) Порядок искомого числа должен быть равен –127.
4) Выделяем мантиссу, двоичной запятой отделяем крайний слева бит (ноль или единицу)
5) Переводим полученную мантиссу в десятичный вид.
6) Записываем ответ по формуле (2).

Заметим, что алгоритм 9 отличается от алгоритма 8 тем, что скрытая единица не добавляется.

Алгоритм 10. Перевод машинного представления произвольного вещественного числа одинарной точности в десятичный вид (см. рис. 5).

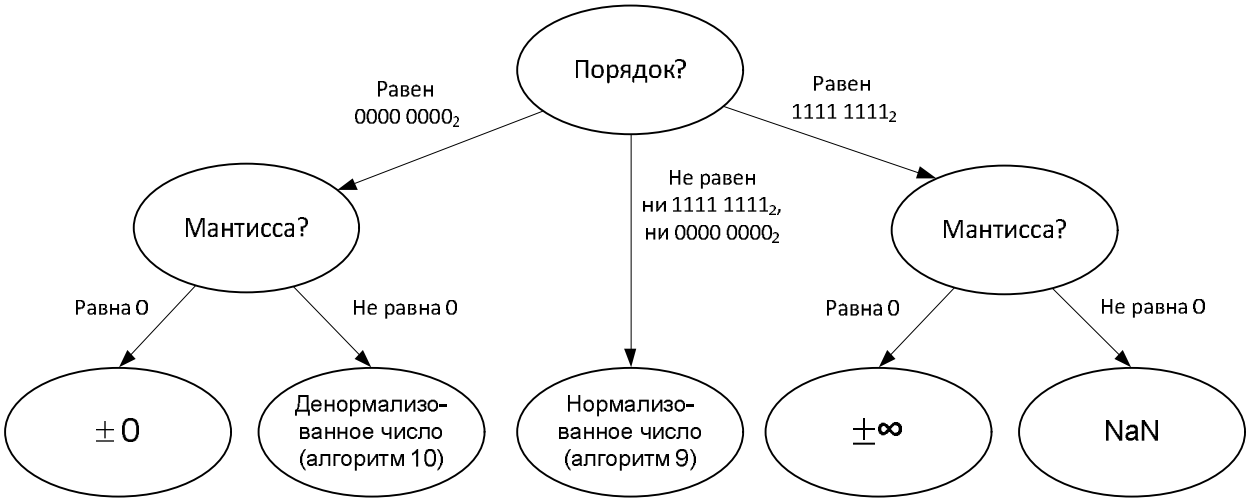


Рис. 5. Алгоритм 10 – перевод машинного представления произвольного вещественного числа одинарной точности в десятичный вид

4.3. Диапазоны типов данных

Сразу же выделим принципиальное отличие машинного представления целых и вещественных чисел: если мы указываем диапазон целых чисел, то подразумеваем, что *все* числа из данного диапазона имеют свое машинное представление. В то же время, как известно, любой диапазон вещественных чисел содержит бесконечное множество значений, тогда как любая ячейка памяти – конечна и поэтому не может представлять бесконечное множество значений. Таким образом, в машинный диапазон вещественных чисел входят *не все* числа, а только их конечное подмножество.

Для определения диапазона вещественных чисел нужно найти минимальные и максимальные (по модулю) нормализованные и денормализованные числа.

1) Минимальное нормализованное число:

x 0000 0001 000 0000 0000 0000 0000 0000

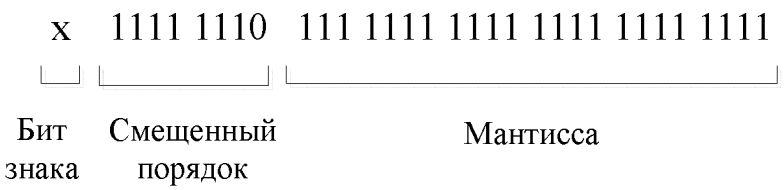
Бит знака — Смещенный порядок

В шестнадцатеричном виде (знак «+»): 0080 0000.

Находим значение этого числа по алгоритму 8:

$1{,}0 \cdot 2^{1-127} = 1{,}0 \cdot 2^{-126} = 1{,}17549435 \cdot 10^{-38}$.

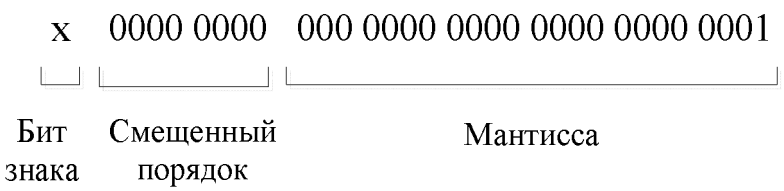
2) Максимальное нормализованное число:



В шестнадцатеричном виде (знак «+»): 7F7F FFFF.

Значение этого числа:
$1{,}111\ 1111\ 1111\ 1111\ 1111\ 1111_2 \cdot 2^{254-127} =$
$= 1{,}99999988_{10} \cdot 2^{127} = 3{,}40282347 \cdot 10^{+38}$.

3) Минимальное денормализованное число:

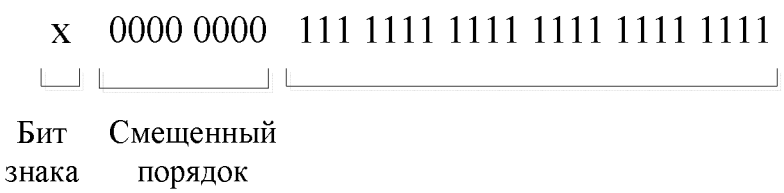


В шестнадцатеричном виде (знак «+»): 0000 0001.

Находим значение этого числа по алгоритму 9 для денормализованных чисел:

$0{,}00\ 0000\ 0000\ 0000\ 0000\ 0001_2 \cdot 2^{-127} = 1{,}0 \cdot 2^{-22} \cdot 2^{-127} = 1.40129846 \cdot 10^{-45}$.

4) Максимальное денормализованное число:



В шестнадцатеричном виде (знак «+»): 007F FFFF.

Находим значение числа:

$1{,}11\ 1111\ 1111\ 1111\ 1111\ 1111_2 \cdot 2^{-127} = 1{,}99999976 \cdot 2^{-127} = 1.17549421 \cdot 10^{-38}$.

Таким образом, диапазон вещественных чисел одинарной точности можно представить следующей схемой:

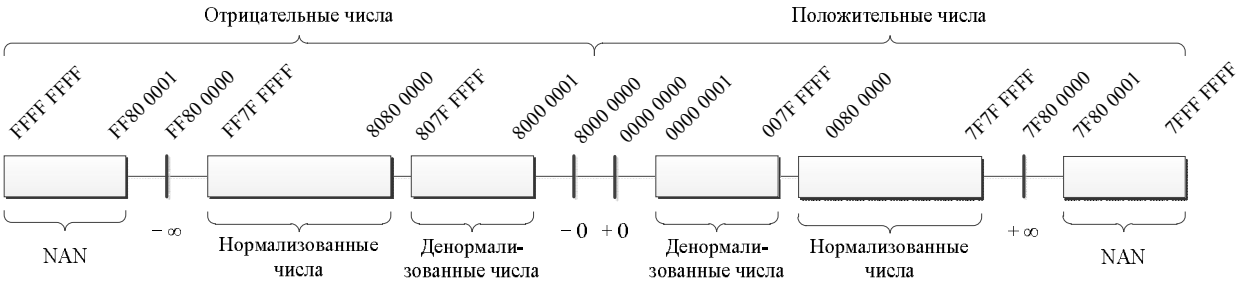


Рис. 6. Диапазон вещественных чисел с одинарной точностью

На рис. 6 можно увидеть, что если результат некоторой операции получится по модулю меньше, чем минимальное денормализованное число (но больше нуля), то в машинном формате представить его будет невозможно и вместо правильного ответа будет ноль. Такая ситуация называется *потерей значимости*. Для операций с целыми типами такая ситуация невозможна.

Отметим ещё одну особенность формата вещественных чисел – *неравномерность расположения* на числовой прямой. Числа, которые могут быть представлены в машинном формате, меньшие по модулю, располагаются чаще, чем большие по модулю.

В заключение приведем диапазоны (только положительные числа) для вещественных типов, используемых в PascalABC.Net (табл. 2).

Таблица 2. Вещественные типы данных в PascalABC.Net

Тип данных	Разрядность, байт	Диапазон
`Single`	4	$[1{,}4\cdot10^{-45} .. 3{,}4\cdot10^{+38}]$
`Real`, `Double`	8	$[5{,}0\cdot10^{-324} .. 1{,}7\cdot10^{+308}]$

5. Задания для самостоятельного решения

1. Назовите основные ячейки памяти и запишите минимальные и максимальные числа (в двоичном, десятичном и шестнадцатеричном форматах), представимые в каждой из них.

2. Что такое разрядность процессора?

3. Запишите машинное представление в беззнаковом 8-разрядном типе следующих десятичных чисел:

0, 10, 50, 200, 255.

4. Запишите машинное представление в беззнаковом 16-разрядном типе следующих десятичных чисел:

10, 200, 500, 1000, 10000.

5. Запишите машинное представление в знаковом 8-разрядном типе следующих десятичных чисел:

0, 1, –1, 10, –10, 50, –50, 127, –128.

6. Запишите машинное представление в знаковом 16-разрядном типе следующих десятичных чисел:

10, –10, –200, 200, –1000, 1000.

7. Определите десятичное число по его машинному представлению:

$1100\ 1010_2$.

Тип данных неизвестен, поэтому должно быть два ответа – для беззнакового типа и для знакового.

8. Определите десятичное число по его машинному представлению:

$1001\ 1101\ 1010\ 1011_2$.

Тип данных неизвестен, поэтому должно быть два ответа – для беззнакового типа и для знакового.

9. Перечислите типы данных для целых чисел в PascalABC.Net. Укажите их диапазоны.

10. Запишите результаты следующих битовых операций:

1011 0111 AND 1101 0011

1011 0111 OR 1101 0011

1011 0111 XOR 1101 0011

$1011\ 0111_2$ SHL 3_{10}

$1011\ 0111_2$ SHR 3_{10}

11. Запишите машинное представление десятичных чисел в вещественном формате одинарной точности:

1; 10; 10000; –1,015625.

12. Найдите десятичные числа по их машинному представлению в вещественном формате одинарной точности:

6DDC0000, AADBB000, FFEEC374, 0068B5C1.

Литература

1. Андреева Е. В., Босова Л. Л., Фалина И. Н. Математические основы информатики. Элективный курс: Учебное пособие. – М.: БИНОМ. Лаборатория знаний, 2005.

2. Петцольд Ч. Код. – М.: Русская редакция, 2001.

3. Цилькер Б. Я., Орлов В. А. Организация ЭВМ и систем. – СПБ.: Питер, 2004.

4. IEEE Computer Society. IEEE Standard for Floating-Point Arithmetic. IEEE Standard 754-2008, August 2008.