

Министерство образования Российской Федерации

**Нижегородский государственный университет
им. Н.И.Лобачевского**

Радиофизический факультет

Кафедра математики

ПРЕДСТАВЛЕНИЕ ДАННЫХ ЦИФРОВЫХ ЭВМ

**Методическая разработка
для студентов радиофизического факультета ННГУ**

Нижний Новгород, 2001

УДК 681.3

Представление данных цифровых ЭВМ : Методическая разработка для студентов радиофизического факультета ННГУ / Сост. В. А. Савин. - Нижний Новгород : ННГУ, 2001. - 58 с.

Рассмотрены вопросы представления различных типов данных на современных цифровых ЭВМ, обсуждаемые в процессе преподавания общего курса “Вычислительные машины и программирование” на радиофизическом факультете ННГУ уже на протяжении ряда лет и нуждающиеся в более солидном методическом обеспечении.

Описаны нестандартные ситуации, незнание которых ведет к потере видения целей и задач курса и препятствует последующему изучению методов вычислений и математического моделирования на ЭВМ. Приведены соответствующие примеры и необходимые пояснения.

Составитель

В. А. Савин

Рецензенты:

А. И. Гавриков

А. В. Жидков

Нижегородский государственный университет
им. Н.И.Лобачевского, 2001

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	4
1. ОБЩИЕ ВОПРОСЫ ПРЕДСТАВЛЕНИЯ ЧИСЛОВЫХ ДАННЫХ.....	6
1.1. Счисление.....	6
1.2. Представление целых чисел в позиционных системах счисления.....	9
1.3. Представление действительных чисел в позиционных системах счисления.....	11
1.4. Округление чисел и погрешности.....	13
2. ДВОИЧНОЕ ПРЕДСТАВЛЕНИЕ ДАННЫХ ЭВМ.....	16
2.1. Двоичное представление целочисленных данных.....	17
2.2. Двоичное представление вещественных данных.....	31
2.2.1. Точность вещественных данных.....	33
2.2.2. Вещественные данные DEC PDP-11..	39
2.2.3. Вещественные данные IBM PC.....	42
2.3. Двоичное представление символьных данных.....	54
2.4. Двоичное представление логических данных.....	56
ЛИТЕРАТУРА.....	57

ВВЕДЕНИЕ

Решение практических задач с использованием вычислительных средств и оргтехники подразумевает адекватное восприятие их возможностей.

Современным цифровым ЭВМ (в силу их аппаратной реализации) доступен единственный язык - язык двоичных чисел. Любой объект, с которым оперирует машина, является записанным по определенным правилам двоичным кодом. Детальное изучение правил записи и выполнения машинных команд составляет предмет курса по архитектуре ЭВМ и выходит за рамки курса вычислительных машин и программирования, являющегося базовым курсом в системе математической подготовки на радиофизическом факультете ННГУ. Вместе с тем, правила двоичного представления данных стали за последнее время настолько похожими для разных ЭВМ, что изложение общих подходов, лежащих в их основе, не вызывает теперь непреодолимых трудностей. Появилась возможность сосредоточить внимание на чрезвычайно актуальных и практически значимых вопросах двоичного кодирования данных, оставаясь в рамках общеобразовательного курса.

Первым в ряду таких вопросов стоит вопрос о диапазоне значений используемых числовых данных. Несведущего пользователя может весьма удивить отрицательный результат сложения существенно положительных величин, выполненный его программой с выбранным “на авось” типом ячейки памяти для суммирования.

Другим важнейшим вопросом является вопрос о точности выполнения операций с приближенно представимыми на ЭВМ вещественными числами. Относительный характер

ошибки представления чисел, ограниченное количество значащих цифр, округление чисел, отбрасывание значащих разрядов - все это оказывается довольно трудным для понимания начинающих программистов.

С момента появления такой отрасли знания, как методы вычислений на ЭВМ, бытует утверждение, что эквивалентные с точки зрения классической математики действия вовсе не являются таковыми с точки зрения математики вычислительно - машинной. Помочь начинающим пользователям также проникнуться этой мыслью - одна из главных задач данной методической разработки.

Серьезному и вдумчивому специалисту в ходе постановки и решения практической задачи необходимо учитывать особенности выполнения операций на цифровой ЭВМ, проводить обоснованный выбор типов и размеров используемых данных, анализировать полученный результат.

Только такой подход позволяет в конечном итоге добиваться получения компактной, эффективной и корректно работающей вычислительно - машинной программы.

В процессе изложения материала обсуждаются основные типы данных ANSI - стандартов алгоритмических языков высокого уровня FORTRAN и C, как наиболее идеологически близких и полезных будущим радиофизикам - исследователям. На этих же языках написаны иллюстрирующие примеры, ориентированные на использование практически любого современного компилятора, работающего под управлением операционных систем UNIX, MS-DOS, MS-WINDOWS на ЭВМ класса IBM PC.

1. ОБЩИЕ ВОПРОСЫ ПРЕДСТАВЛЕНИЯ ЧИСЛОВЫХ ДАННЫХ

1.1. Счисление

Понятие числа неразрывно связано с развитием математики и составляет ее основу [1 , 2]. Потребность в счете и последовавший затем переход от использования именованных чисел к отвлеченным числам, счет группами (парами, десятками, дюжинами, то есть прообразами так называемых узловых чисел), возникновение арифметических операций привели, в конечном итоге, к появлению того, что в настоящее время определяется понятием натурального числа. Все народы, обладавшие письменностью, владели этим понятием и пользовались той или иной системой счисления.

Понятие счисления (нумерации) включает совокупность приемов представления натуральных чисел. Одновременно с формированием систем счисления шло совершенствование принципов записи чисел. Наиболее ранней и примитивной является словесная запись чисел. Более совершенной формой представления чисел стали цифры - специальные условные знаки для обозначения и записи чисел. Цифровое представление чисел происходит на основе цифровых последовательностей (слов, кортежей). Они строятся путем упорядочения некоторого количества знаков из конечного множества этих знаков, образующих алфавит системы счисления.

В любой системе счисления некоторые символы (слова или знаки) служат для обозначения определенных чисел, называемых узловыми. Остальные числа (алгоритмические) получаются в результате каких-либо операций из узловых чисел. Системы счисления различаются выбором узловых

чисел и способами образования алгоритмических, а с появлением письменных обозначений числовых символов, системы счисления стали различаться характером числовых знаков и принципами их записи.

Различают непозиционные и позиционные системы счисления. Для первых каждый числовой знак в записи любого числа в ней имеет одно и то же значение. Для вторых значение числового знака зависит от его расположения в записи числа. Все известные позиционные системы счисления - аддитивно-мультипликативные.

В Древнем Египте было несколько разных систем счисления. Примерно за 2,5 - 3 тысячи лет до н.э. существовала иероглифическая нумерация, где имелись специальные знаки для обозначения единиц десятичных разрядов вплоть до 10^7 . Другие числа изображались путем комбинации этих знаков. Основной арифметической операцией было сложение, то есть имела место непозиционная десятичная аддитивная система счисления.

За 2 тысячи лет до н.э. в древнем Вавилоне употребляли шестидесятиричную нумерацию с позиционным принципом записи чисел. Узловыми являлись числа **1**, **10**, **60**. Вавилонские цифры представляли собой клинописные знаки. Порядок следования разрядов чисел совпадал с ныне принятым. На ее основе, вероятнее всего, среднеазиатские ученые не позднее 10 века создали позиционную шестидесятиричную систему счисления. Она особенно широко применялась в астрономических вычислениях и таблицах, а следы ее дошли до нашего времени в измерении времени и углов.

Наиболее долговечной из древнейших цифровых систем

оказалась римская нумерация, возникшая около 5 века до н.э. и применяемая иногда и в настоящее время. Узловыми числами в ней являлись числа **1, 5, 10, 50, 100, 500, 1000**, обозначаемые соответственно знаками **I, V, X, L, C, D, M**. Система счисления является непозиционной, алгоритмические числа в ней получаются путем сложения и вычитания узловых: **I = 1, III = 3, IV = 4, VIII = 8, IX = 9,**

XXI = 21, XLVI = 46, CCVL = 245,

XDVII = 497, MMII = 2002.

Начало процесса формирования современной позиционной десятичной системы счисления относится к началу н.э., когда в Индии уже была широко распространена словесная позиционная десятичная нумерация. Там же, вероятно, не позднее 5 века появились прообразы современных цифр (включая нуль). С 8 века эта система стала распространяться по Арабскому Востоку, а после 10 века с ней познакомились европейцы. Всеобщее распространение новые цифры, занесенные в Европу арабами, получили начиная с 15 века, приобретя их сохранившееся поныне название - арабские. Это произошло в силу удобства записи чисел при помощи этих цифр в позиционной десятичной системе счисления. Начертание цифр претерпело со временем ряд крупных изменений.

Сам позиционный принцип записи оправдывается теоремой элементарной теории чисел, устанавливающей единственность и однозначность записи в ней любого натурального числа.

1.2. Представление целых чисел в позиционных системах счисления

В позиционной системе счисления в качестве базисного числа (базы, основания) выбирается некоторое натуральное число $r > 1$ и на его основе строится r -ичная система счисления. Для $r = 1$ системы счисления не существует.

Далее вводятся r основных знаков алфавита такой позиционной системы счисления, называемых ее цифрами:

$$R_0, R_1, \dots, R_{r-1}. \quad (1)$$

Для двоичной (бинарной, диадной) системы счисления с $r = 2$ в качестве цифр чаще всего используются 0 и 1 .

Для восьмеричной системы счисления с $r = 8$ это цифры $0, 1, 2, 3, 4, 5, 6, 7$.

Для десятичной системы счисления с $r = 10$ это наиболее привычные арабские цифры $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$.

Для шестнадцатеричной системы счисления с $r = 16$ это арабские цифры от 0 до 9 , а также латинские буквы $A(a), B(b), C(c), D(d), E(e), F(f)$.

Знаки алфавитов (цифры) соответствующих систем счисления служат для образования цифровых последовательностей нуля и натуральных чисел. Каждой цифровой последовательности только из одной цифры однозначно соответствует число нуль или одно из $r - 1$ первых натуральных чисел.

Число нуль и всякое натуральное число имеют строго однозначное цифровое представление в r -ичной системе счисления:

$$X_p X_{p-1} \dots X_1 X_0. \quad (2)$$

Здесь все цифры $0 \leq X_i \leq r - 1$ принадлежат множеству знаков алфавита рассматриваемой позиционной си-

стемы счисления (1) и являются позиционными коэффициентами, то есть участвуют в образовании искомого значения в соответствии со своей позицией в этой записи. Значение представляемого числа образуется аддитивно-мультипликативно:

$$\mathbf{X} = \mathbf{X}_p \mathbf{W}_p + \mathbf{X}_{p-1} \mathbf{W}_{p-1} + \dots + \mathbf{X}_1 \mathbf{W}_1 + \mathbf{X}_0 \mathbf{W}_0 . \quad (3)$$

Числовое значение каждого позиционного коэффициента \mathbf{X}_i добавляется в общее значение с учетом его позиционного значения (веса) $\mathbf{W}_i = \mathbf{r}^i$, поэтому аддитивный вклад каждой цифры в значение числа равен $\mathbf{X}_i \mathbf{r}^i$, $i = 0, 1, \dots, p$. Здесь p соответствует позиционному значению старшего значащего (отличного от нуля) позиционного коэффициента \mathbf{X}_p .

В качестве примера рассмотрим цифровое представление в различных позиционных системах счисления натурального числа **168** :

$$\begin{aligned} 168 &= 1 \cdot 10^2 + 6 \cdot 10^1 + 8 \cdot 10^0 = \\ &= 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = \\ &= 2 \cdot 8^2 + 5 \cdot 8^1 + 0 \cdot 8^0 = 10 \cdot 16^1 + 8 \cdot 16^0 . \end{aligned}$$

Таким образом $168_{10} = 10101000_2 = 250_8 = A8_{16} (a8_{16})$.

Запись любого числа в любой позиционной системе счисления (2) всегда может быть дополнена слева любым произвольным количеством незначащих нулей, что, естественно, ни в коей мере не нарушает однозначности цифрового представления этого числа:

$$8_{10} = 0008_{10} = 10_8 = 000010_8 = 1000_2 = 00001000_2 .$$

Однако из соображений удобства чаще всего пользуются минимально необходимым количеством позиций для записи разных чисел.

Последние рассуждения, среди прочего, прямо указывают на невозможность записи произвольного численного значения в ограниченное количество позиций в любой системе счисления: этих позиций может просто не хватить для записи старших значащих позиционных коэффициентов.

Дополнение алфавита позиционной системы счисления унарными знаками $+$ и $-$ позволяет наряду с натуральными числами и нулем записывать отрицательные числа, то есть получить расширенное множество целых (положительных, отрицательных и нуля) чисел.

1.3. Представление действительных чисел в позиционных системах счисления

Желание записывать как целую, так и дробную части значений действительных чисел в позиционной системе счисления ведет к необходимости распространения описанных выше правил на отрицательные значения показателей i весовых коэффициентов W_i .

Для выделения коэффициента X_0 с позиционным значением $W_0 = r^0$ необходим дополнительный знак дробности, размещаемый в цифровой последовательности непосредственно справа от этого коэффициента. В качестве такого знака обычно берется точка (иногда запятая).

Исходя из вышесказанного и в соответствии с выражением (2) некоторое действительное число может иметь следующее цифровое представление в r -ичной системе счисления:

$$X_p \dots X_0 \bullet X_{-1} \dots X_{-q} . \quad (4)$$

Значение представляемого числа образуется по аналогии с выражением (3) следующим образом:

$$\mathbf{X} = \mathbf{X}_p \mathbf{W}_p + \dots + \mathbf{X}_0 \mathbf{W}_0 + \mathbf{X}_{-1} \mathbf{W}_{-1} + \dots + \mathbf{X}_{-q} \mathbf{W}_{-q} . \quad (5)$$

Представление действительных чисел с дробной частью имеет ряд особенностей. Каждое число, точно представимое в произвольной позиционной системе счисления последовательностью цифр конечной длины, является рациональным числом. Наоборот, в каждой позиционной системе можно точно представить только некоторое подмножество рациональных чисел, определяемое выбором основания \mathbf{r} системы счисления.

Так рациональное число $7/3$ может быть точно представлено конечной последовательностью цифр в троичной ($\mathbf{r} = 3$) системе счисления, но это оказывается невозможным для двоичной и десятичной систем:

$$\frac{7}{3} = 2.1_3 \approx 10.010101010101 \dots_2 \approx 2.33333333 \dots_{10} .$$

Рациональное число $19/5$ может быть точно представлено конечной последовательностью цифр в десятичной системе счисления, но не в двоичной и троичной системах:

$$\frac{19}{5} = 3.8_{10} \approx 11.110011001100 \dots_2 \approx 10.21012101 \dots_3 .$$

Здесь работает следующее правило. Если \mathbf{a}/\mathbf{b} - рациональное число, где \mathbf{a} и \mathbf{b} - взаимно простые натуральные числа, то \mathbf{a}/\mathbf{b} может быть точно представлено в позиционной системе счисления с основанием \mathbf{r} тогда и только тогда, когда каждый простой множитель в разложении числа \mathbf{b} является простым множителем в разложении числа \mathbf{r} . Отсюда, в десятичной системе счисления точно представимы только

такие рациональные числа a/b , у которых b содержит лишь простые множители **2** и **5**.

Для двоичной системы точно представимы только числа, у которых $b = 2^n$, где n - неотрицательное целое число.

Запись точно представимого в заданной позиционной системе рационального числа (4) всегда может быть дополнена справа (в его дробной части) любым произвольным количеством нулей, которые не нарушают однозначности цифрового представления этого числа:

$$\frac{11}{8} = 1.375_{10} = 1.375000_{10} = 1.011_2 = 1.011000_2.$$

Подобно незначащим нулям в целой части числа (см. выше), незначащие нули в дробной части числа также обычно опускаются.

1.4. Округление чисел и погрешности

Не каждое рациональное и тем более иррациональное число можно представить в позиционной системе счисления в виде конечной цифровой последовательности. Поэтому чаще всего приходится иметь дело с приближенным значением представляемого действительного числа, содержащим ограниченное число цифр. Аналогичная ситуация возникает и тогда, когда точное число содержит конечное, но слишком большое количество цифр.

Простейшим способом получения приближенного значения числа является отбрасывание цифр справа в его точной записи (обрыв), начиная с некоторой позиции (разряда). При этом погрешность приближения $\Delta Z = A - Z$,

где \mathbf{Z} - точное число, а \mathbf{A} - его приближение, не превосходит по абсолютной величине единицы разряда последней сохраняемой в приближенной записи цифры.

Другим способом получения приближенного значения числа является его округление. Наиболее часто употребляются стандартные правила математического округления, при которых погрешность приближения не превосходит по абсолютной величине половины единицы разряда последней сохраняемой в приближенной записи цифры.

Так для несуществующего в виде периодической десятичной дроби иррационального числа $\pi \approx \mathbf{3.14159265} \dots_{10}$ отбрасывание правых разрядов после позиционного значения $\mathbf{W}_{-4} = 10^{-4}$ дает приближенное значение $\mathbf{3.1415}_{10}$. Его погрешность $|\Delta \mathbf{Z}| \leq \mathbf{W}_{-4}$, а само значение можно трактовать как результат округления с недостатком.

Математическое округление этого же числа в тех же разрядах дает результат округления с избытком $\mathbf{3.1416}_{10}$. Его погрешность $|\Delta \mathbf{Z}| \leq \mathbf{W}_{-4}/2$. Поскольку погрешность последнего результата не превосходит половины позиционного значения последней (самой правой) цифры, все его цифры являются верными.

Приближенное число обычно характеризуется количеством сохраненных разрядов в дробной части числа или количеством значащих цифр. К значащим цифрам относятся все цифры, кроме нулей слева. Так, например, действительные числа $\mathbf{270}$, $\mathbf{3.14}$, $\mathbf{0.00956}$ имеют по три значащие цифры.

При записи приближенных чисел все значащие цифры должны быть верными, если погрешность числа не указывается каким-либо иным способом.

Приближенные значения чисел появляются не только в результате обрыва и округления. Каждое измеряемое значение некоторой величины в общем случае также есть приближенное значение этой величины. Если A - приближенное значение числа Z , то величина $\Delta Z = A - Z$ называется истинной погрешностью числа A и совпадает с введенной ранее погрешностью приближения, а величина $\delta Z = \Delta Z / Z$ называется истинной относительной погрешностью числа A . Поскольку в большинстве случаев точное значение числа Z неизвестно, то неизвестными оказываются как истинная, так и истинная относительная погрешности числа A . Однако, чаще всего можно указать граничную величину истинной погрешности, то есть такое число $\Delta A > 0$, для которого выполняется неравенство $|A - Z| \leq \Delta A$, или $A - \Delta A \leq Z \leq A + \Delta A$. В этом случае величина ΔA называется пределом (границей) погрешности, или предельной абсолютной погрешностью, или сокращенно абсолютной погрешностью числа A . Величина $\delta A = \Delta A / A$ называется предельной относительной погрешностью, или сокращенно относительной погрешностью числа A .

2.ДВОИЧНОЕ ПРЕДСТАВЛЕНИЕ ДАННЫХ ЭВМ

Хранимые в памяти ЭВМ данные являются аппаратной комбинацией символов **0** и **1**, то есть числами, представленными в двоичной системе счисления. Современные алгоритмические языки высокого уровня содержат большое количество основных и производных типов данных, однако с точки зрения двоичной реализации на ЭВМ имеют место только три основных типа:

- **целочисленные данные**
(форма с фиксированной точкой);
- **вещественные данные**
(форма с плавающей точкой);
- **данные в форме алфавитно-цифровых символов.**

Правильная интерпретация ячейки памяти каждого типа позволяет корректно извлекать хранимую в ней информацию и обрабатывать ее соответствующим образом.

Наименьшей двоичной единицей информации является **1 бит** - единственный двоичный разряд. На большинстве современных ЭВМ минимальной адресуемой единицей памяти является **1 байт (1В)**, содержащий комбинацию фиксированного числа соседних битов. В настоящее время **1 байт** практически всегда равен **8 битам**.

Данные разных типов занимают различные по размеру ячейки памяти, но этот размер всегда кратен **1 байту**. В различных языковых системах существуют ячейки памяти размером в **1В, 2В, 4В, 6В, 8В, 16В** и т.д. Адрес такой ячейки в оперативной памяти ЭВМ (ОЗУ) определяется адресом ее начального байта, который, в свою оче-

редь, является, по сути, порядковым номером этого байта в ОЗУ. Для измерения величины ОЗУ и магнитных накопителей ЭВМ служат производные единицы:

$$1\text{KB} = 2^{10}\text{B} = 1024\text{B} \approx 10^3\text{B} ,$$

$$1\text{MB} = 2^{10}\text{KB} = 1048576\text{B} \approx 10^6\text{B} ,$$

$$1\text{GB} = 2^{10}\text{MB} = 1073741824\text{B} \approx 10^9\text{B} .$$

2.1. Двоичное представление целочисленных данных

Каждый отдельный бит может принимать значения **0** или **1** , то есть число его значений составляет $2^1 = 2$. Соответственно, двухбитовая конструкция может принимать $2^2 = 4$ разных значения, а четырехбитовая - $2^4 = 16$ разных значений. На примере этого последнего (в реальности отсутствующего на современных ЭВМ, виртуального) типа данных можно весьма удобно и неумолимо рассмотреть принятый в настоящее время способ формирования двоичных целых чисел.

Для представления целых чисел в памяти ЭВМ используется внутренняя форма записи с фиксированной точкой, когда самый младший (правый) двоичный разряд соответствует нулевой степени числа **2** .

Основываясь на материале 1-й главы, несложно получить все **16** возможных беззнаковых значений четырехбитового целого данного от **0000₂** до **1111₂** , которые можно трактовать как значения в диапазоне от **0₁₀** до **15₁₀** .

При выполнении вычислений на ЭВМ помимо неотрицательных требуются также и отрицательные целые числа. Для удобства выполнения операций представление це-

лых чисел организовано так, что при наличии **0** в самом старшем (левом) двоичном разряде число считается неотрицательным, а при наличии **1** - отрицательным.

В процессе развития цифровых ЭВМ существовали разные формы представления отрицательных целых чисел.

Знаковая форма фактически соответствует уже описанному способу кодирования знака числа:

$$\begin{aligned} 1_{10} &= 0001_2, \quad -1_{10} = 1001_2; \\ 7_{10} &= 0111_2, \quad -7_{10} = 1111_2. \end{aligned}$$

Форма дополнения до 1 подразумевает представление отрицательных целых чисел в дополнительном (инверсном) коде, когда каждый **0** в записи положительного числа заменяется на **1** в записи отрицательного числа, а **1** меняется на **0** :

$$\begin{aligned} 1_{10} &= 0001_2, \quad -1_{10} = 1110_2; \\ 7_{10} &= 0111_2, \quad -7_{10} = 1000_2. \end{aligned}$$

Обе рассмотренные формы представления отрицательных целых чисел обладают очень существенным с точки зрения организации вычислений на ЭВМ недостатком - наличием двух различных кодировок целочисленного нуля.

В знаковой форме они выглядят так:

$$+0_{10} = 0000_2, \quad -0_{10} = 1000_2.$$

А в форме дополнения до **1** - так:

$$+0_{10} = 0000_2, \quad -0_{10} = 1111_2.$$

Эту проблему удастся устранить использованием представления в **форме дополнения до 2**, получающейся из формы дополнения до **1** добавлением числа **1** к кодировке отрицательных целых чисел:

$$\begin{aligned} 1_{10} &= 0001_2 \Rightarrow 1110_2 \Rightarrow 1111_2 = -1_{10}; \\ 7_{10} &= 0111_2 \Rightarrow 1000_2 \Rightarrow 1001_2 = -7_{10}. \end{aligned}$$

В соответствии с описанными правилами теперь можно записать все **16** значений со знаком четырехбитового целого данного в диапазоне от -8_{10} до $+7_{10}$:

$$\begin{aligned} -8_{10} &= 1000_2; -7_{10} = 1001_2; -6_{10} = 1010_2; -5_{10} = 1011_2; \\ -4_{10} &= 1100_2; -3_{10} = 1101_2; -2_{10} = 1110_2; -1_{10} = 1111_2; \\ 0_{10} &= 0000_2; +1_{10} = 0001_2; +2_{10} = 0010_2; +3_{10} = 0011_2; \\ +4_{10} &= 0100_2; +5_{10} = 0101_2; +6_{10} = 0110_2; +7_{10} = 0111_2. \end{aligned}$$

Самым коротким из реально существующих на современных ЭВМ данных целого типа является **1В = 8 битам** . Байтовые данные могут принимать $2^8 = 256_{10}$ разных значений. Это соответствует значениям в диапазоне от 0_{10} до $2^8 - 1 = 255_{10}$ в беззнаковом варианте и значениям в диапазоне от $-2^7 = -128_{10}$ до $2^7 - 1 = 127_{10}$ в варианте с учетом знака.

Следующим по возрастанию объема занимаемой памяти данным целого типа является **2В = 16 битам** . Двухбайтовые данные могут принимать $2^{16} = 65536_{10}$ разных значений. Это соответствует значениям в диапазоне от 0_{10} до $2^{16} - 1 = 65535_{10}$ в беззнаковом варианте и значениям в диапазоне от $-2^{15} = -32768_{10}$ до $2^{15} - 1 = 32767_{10}$ в варианте с учетом знака.

На большинстве современных ЭВМ длинные данные целого типа занимают ячейки в **4В = 32 битам** . Четырехбайтовые данные могут принимать $2^{32} = 4294967296_{10}$ разных значений. Это соответствует значениям в диапазоне от 0_{10} до $2^{32} - 1 = 4294967295_{10}$ в беззнаковом варианте и значениям в диапазоне от $-2^{31} = -2147483648_{10}$ до $2^{31} - 1 = 2147483647_{10}$ в варианте с учетом знака.

Зная порядок расположения байт в ячейке интересующей ЭВМ, можно написать программу, позволяющую получить

битовое представление произвольного целого числа. Примеры таких программы на языках FORTRAN и C для ЭВМ IBM PC приведены ниже. Следует заметить, что для C - варианта программы использование нестандартной функции ltoa() компилятора Borland C++ объясняется лишь желанием продемонстрировать короткий и простой способ решения такой задачи в ущерб его общности и мобильности.

```

PROGRAM    I__BIT
INTEGER * 1  I1, II(2), IL(4)
INTEGER * 2  I2
INTEGER * 4  I4
EQUIVALENCE ( I2, II(1) ), ( I4, IL(1) )
10 WRITE(*,15)
15 FORMAT(/,' INTEGER VALUE : ', $)
   READ(*, *, ERR = 10)  I4
   WRITE(*,35) I4
35 FORMAT(14X,'INTEGER*4 =', I11)
   CALL BIT( 4, IL )
   IF( I4.LT.-32768 .OR. I4.GT.32767 ) GOTO 10
   I2 = I4
   WRITE(*,55) I2
55 FORMAT(19X,'INTEGER*2 =', I6, /, 18(' '), $)
   CALL BIT( 2, II )
   IF( I4 .LT. -128 .OR. I4 .GT. 127 ) GOTO 10
   I1 = I4
   WRITE(*,75) I1
75 FORMAT(21X,'INTEGER*1 =', I4, /, 27(' '), $)
   CALL BIT( 1, I1 )
   GOTO 10
END

```

```

SUBROUTINE BIT( NB, IB )
INTEGER      I, J, JB, KB, MB, NB
INTEGER*1    IB(8), IC(8)
DO  I = NB, 1, -1
    JB = IB(I)
    IF( JB .LT. 0 ) JB = JB + 256
    DO  J = 1, 8
        MB = 2 ** ( 8 - J )
        KB = JB / MB
        JB = JB - KB * MB
        IC(J) = KB
    END DO
    WRITE(*,'(1X, 8I1, $)')( IC(J), J = 1, 8 )
END DO
WRITE(*,*)
RETURN
END

INTEGER VALUE : 1
            INTEGER*4 =          1
00000000 00000000 00000000 00000001
            INTEGER*2 =          1
            00000000 00000001
            INTEGER*1 =          1
                    00000001

INTEGER VALUE : -1
            INTEGER*4 =          -1
11111111 11111111 11111111 11111111
            INTEGER*2 =          -1
            11111111 11111111
            INTEGER*1 =          -1
                    11111111

```

```

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
char* ltoa( long, char*, int );
int main( void ) {
    long l; int k, n; char s[33];
    scanf( "%ld", &l );
    ltoa( l, s, 2 ); n = strlen( s );
    for( k=0; k<32-n; k++ ) printf( "0" );
    for( k=0; k<n; k++ ) printf( "%c", s[k] );
    printf( "\n" );
    return 0;
}
1
00000000000000000000000000000001
-1
11111111111111111111111111111111

```

Приведем в качестве иллюстрации двоичное представление ранее не рассматривавшихся положительных и отрицательных значений целых степеней двойки по мере увеличения объема занимаемой памяти:

$$\begin{aligned}
 +2^3 &= +8_{10} = 00001000_2; & +2^4 &= +16_{10} = 00010000_2; \\
 -2^3 &= -8_{10} = 11111000_2; & -2^4 &= -16_{10} = 11110000_2; \\
 +2^5 &= +32_{10} = 00100000_2 & +2^6 &= +64_{10} = 01000000_2; \\
 -2^5 &= -32_{10} = 11100000_2 & -2^6 &= -64_{10} = 11000000_2; \\
 &+2^7 &= +128_{10} &= 00000000\ 10000000_2; \\
 &-2^7 &= -128_{10} &= 11111111\ 10000000_2; \\
 &+2^8 &= +256_{10} &= 00000001\ 00000000_2; \\
 &-2^8 &= -256_{10} &= 11111111\ 00000000_2; \\
 &+2^9 &= +512_{10} &= 00000010\ 00000000_2; \\
 &-2^9 &= -512_{10} &= 11111110\ 00000000_2;
 \end{aligned}$$

$$\begin{aligned}
+2^{10} &= +1024_{10} = 00000100\ 00000000_2; \\
-2^{10} &= -1024_{10} = 11111100\ 00000000_2; \\
+2^{11} &= +2048_{10} = 00001000\ 00000000_2; \\
-2^{11} &= -2048_{10} = 11111000\ 00000000_2; \\
+2^{12} &= +4096_{10} = 00010000\ 00000000_2; \\
-2^{12} &= -4096_{10} = 11110000\ 00000000_2; \\
+2^{13} &= +8192_{10} = 00100000\ 00000000_2; \\
-2^{13} &= -8192_{10} = 11100000\ 00000000_2; \\
+2^{14} &= +16384_{10} = 01000000\ 00000000_2; \\
-2^{14} &= -16384_{10} = 11000000\ 00000000_2; \\
+2^{15} &= +32768_{10} = 00000000\ 00000000\ 10000000\ 0..0_2; \\
-2^{15} &= -32768_{10} = 11111111\ 11111111\ 10000000\ 0..0_2; \\
+2^{16} &= +65536_{10} = 00000000\ 00000001\ 00000000\ 0..0_2; \\
-2^{16} &= -65536_{10} = 11111111\ 11111111\ 00000000\ 0..0_2; \\
+2^{17} &= +131072_{10} = 00000000\ 00000010\ 00000000\ 0..0_2; \\
-2^{17} &= -131072_{10} = 11111111\ 11111110\ 00000000\ 0..0_2; \\
+2^{18} &= +262144_{10} = 00000000\ 00000100\ 00000000\ 0..0_2; \\
-2^{18} &= -262144_{10} = 11111111\ 11111100\ 00000000\ 0..0_2; \\
+2^{19} &= +524288_{10} = 00000000\ 00001000\ 00000000\ 0..0_2; \\
-2^{19} &= -524288_{10} = 11111111\ 11111000\ 00000000\ 0..0_2; \\
+2^{20} &= +1048576_{10} = 00000000\ 00010000\ 00000000\ 0..0_2; \\
-2^{20} &= -1048576_{10} = 11111111\ 11110000\ 00000000\ 0..0_2; \\
+2^{21} &= +2097152_{10} = 00000000\ 00100000\ 00000000\ 0..0_2; \\
-2^{21} &= -2097152_{10} = 11111111\ 11100000\ 00000000\ 0..0_2; \\
+2^{22} &= +4194304_{10} = 00000000\ 01000000\ 00000000\ 0..0_2; \\
-2^{22} &= -4194304_{10} = 11111111\ 11000000\ 00000000\ 0..0_2; \\
+2^{23} &= +8388608_{10} = 00000000\ 10000000\ 00000000\ 0..0_2; \\
-2^{23} &= -8388608_{10} = 11111111\ 10000000\ 00000000\ 0..0_2; \\
+2^{24} &= +16777216_{10} = 00000001\ 00000000\ 00000000\ 0..0_2; \\
-2^{24} &= -16777216_{10} = 11111111\ 00000000\ 00000000\ 0..0_2;
\end{aligned}$$

$$\begin{aligned}
+2^{25} &= +33554432_{10} = 00000010\ 00000000\ 00000000\ 0..0_2; \\
-2^{25} &= -33554432_{10} = 11111110\ 00000000\ 00000000\ 0..0_2; \\
+2^{26} &= +67108864_{10} = 00000100\ 00000000\ 00000000\ 0..0_2; \\
-2^{26} &= -67108864_{10} = 11111100\ 00000000\ 00000000\ 0..0_2; \\
+2^{27} &= +134217728_{10} = 00001000\ 00000000\ 00000000\ 0..0_2; \\
-2^{27} &= -134217728_{10} = 11111000\ 00000000\ 00000000\ 0..0_2; \\
+2^{28} &= +268435456_{10} = 00010000\ 00000000\ 00000000\ 0..0_2; \\
-2^{28} &= -268435456_{10} = 11110000\ 00000000\ 00000000\ 0..0_2; \\
+2^{29} &= +536870912_{10} = 00100000\ 00000000\ 00000000\ 0..0_2; \\
-2^{29} &= -536870912_{10} = 11100000\ 00000000\ 00000000\ 0..0_2; \\
+2^{30} &= +1073741824_{10} = 01000000\ 00000000\ 00000000\ 0..0_2; \\
-2^{30} &= -1073741824_{10} = 11000000\ 00000000\ 00000000\ 0..0_2.
\end{aligned}$$

Ограниченность диапазона значений целочисленных данных прямо следует из ограниченности длины их двоичного представления. Битовая длина ячейки памяти, занимаемая каждым из рассмотренных выше производных типов целочисленных данных, определяет лишь разные границы этого диапазона. Отсюда вытекает главная проблема выполнения операций с данными целого типа - возможность переполнения знакового разряда.

Так при выполнении операций сложения перенос единицы в старший (знаковый) разряд двоичного представления изменяет знак результата:

r=10		r=2		r=10		r=2			
-----		-----		-----		-----			
3		0011		00000011	3		0011		00000011
+					+				
4		0100		00000100	5		0101		00000101
-----		-----		-----		-----		-----	
+7		0111		00000111	+8		1000		00001000

Приведенный пример демонстрирует, что результат операции $3_{10} + 4_{10} = 7_{10}$ корректен как для байтовых, так и для четырехбитовых целочисленных данных. Одновременно результат операции $3_{10} + 5_{10} = 8_{10}$ корректен для байтовых, но некорректен и равен -8_{10} для четырехбитовых целочисленных данных, поскольку в последнем случае вообще не существует двоичного представления правильного результата.

Проблемы знакового разряда можно продемонстрировать и в случае выполнения операций вычитания.

При этом оказывается, что кодировка отрицательных целых чисел в форме дополнения до **2** весьма удобна при условии определенной интерпретации ситуаций выхода за пределы знакового разряда:

r=10	r=2	r=10	r=2	r=10	r=2
----- -----		----- -----		----- -----	
4	0100	-4	1100	3	0011
+		+		+	
-3	1101	-3	1101	-4	1100
----- -----		----- -----		----- -----	
+1	1 0001	-7	1 1001	-1	1111

Рассмотрим в качестве нового примера следующие операции:

r=10	r=2	r=10	r=2
----- -----		----- -----	
7	0111 00000111	-8	1000 11111000
+		+	
1	0001 00000001	-1	1111 11111111
----- -----		----- -----	
+8	1000 00001000	-9	1 0111 1 11110111

Теперь результаты операций

$7_{10} + 1_{10} = 8_{10}$ и $-8_{10} - 1_{10} = -9_{10}$ корректны для байтовых, но некорректны и равны -8_{10} и $+7_{10}$ соответственно для четырехбитовых целочисленных данных.

Аналогично можно продемонстрировать возможность переполнения знакового разряда при выполнении операций сложения для байтовых, двухбайтовых и четырехбайтовых целочисленных данных:

r=10		r=2		r=10		r=2
-----		-----		-----		-----
127		01111111		32767		01111111 11111111
+				+		
1		00000001		1		00000000 00000001
-----		-----		-----		-----
+128		10000000		+32768		10000000 00000000

Здесь результаты операций

$127_{10} + 1_{10} = 128_{10}$ и $32767_{10} + 1_{10} = 32768_{10}$ некорректны и равны соответственно -128_{10} и -32768_{10} .

r=10		r=2
-----		-----
2147483647		01111111 11111111 11111111 11111111
+		
1		00000000 00000000 00000000 00000001
-----		-----
+2147483648		10000000 00000000 00000000 00000000

Здесь результат операции равен -2147483648_{10} , хотя $2147483647_{10} + 1_{10} = 2147483648_{10}$.

Для байтовых, двухбайтовых и четырехбайтовых целочисленных данных можно продемонстрировать проблемы знакового разряда и в случае выполнения операций вычитания:

r=10		r=2		r=10		r=2
-----		-----		-----		-----
-128		10000000		-32768		10000000 00000000
+				+		
-1		11111111		-1		11111111 11111111
-----		-----		-----		-----
-129		1 01111111		-32769		1 01111111 11111111

Здесь результаты операций

$-128_{10} - 1_{10} = -129_{10}$ и $-32768_{10} - 1_{10} = -32769_{10}$ некорректны и равны соответственно $+127_{10}$ и $+32767_{10}$.

r=10		r=2
-----		-----
-2147483648		10000000 00000000 00000000 00000000
+		
-1		11111111 11111111 11111111 11111111
-----		-----
-2147483649		1 01111111 11111111 11111111 11111111

Здесь результат операции равен $+2147483647_{10}$, хотя $-2147483648_{10} - 1_{10} = -2147483649_{10}$.

В качестве наиболее наглядного примера обоснованного выбора размеров ячеек памяти целочисленных данных может служить задача вычисления количества “счастливых” билетов с шестизначными номерами от **000000** до **999999**. Напомним, что “счастливым” считается билет, у которого

Далее приводятся тексты программ на языках FORTRAN и С, реализующих один из возможных способов решения поставленной задачи.

```
I2_SUM = -10284
I4_SUM = 55252
```

```

#include <stdio.h>
int main( void ) {
    short i, j, k, l, m, n;
    int    int__sum = 0;
    long   long_sum = 0L;
    for( i=0; i<=9; i++ ) {
        for( j=0; j<=9; j++ ) {
            for( k=0; k<=9; k++ ) {
                for( l=0; l<=9; l++ ) {
                    for( m=0; m<=9; m++ ) {
                        for( n=0; n<=9; n++ ) {
                            if( i+j+k == l+m+n ) {
                                int__sum++;
                                long_sum++; } } } } } } }
    printf("int__sum = %d\n", int__sum);
    printf("long_sum = %ld\n", long_sum);
    return 0;
}

int__sum = -10284
long_sum = 55252

```

Результаты выполнения программ могут показаться странными лишь на первый взгляд. Правильный ответ 55252_{10} получается для четырехбайтовой целочисленной ячейки суммирования. Двухбайтовая же ячейка слишком коротка для хранения корректного результата проводимого суммирования. На начальном этапе суммирования в ней достигается результат $+32767_{10}$, затем происходит переполнение знакового разряда и получается число -32768_{10} , после чего дальнейшее суммирование останавливается на значении

-10284_{10} , что конечно же неверно.

Весьма интересны иллюстрации к выполнению операций умножения в двоичной системе счисления:

r=10	r=2	r=10	r=2
---- -----		----- -----	
7	00000111	17	0000000000010001
* *		* *	
3	00000011	13	0000000000001101
---- -----		----- -----	
21	00000111	51	0000000000010001
+		+ +	
0 0000111		17	00 00000000010001
-----		----- +	
0 00010101		221	000 0000000010001

		000 0000000011011101	

Здесь получение результата сводится к сложению сдвинутых в позиции единиц второго сомножителя битовых записей первого сомножителя.

2.2. Двоичное представление вещественных данных

Для двоичного кодирования на ЭВМ вещественных данных необходимо определить правила записи:

- **знака числа;**
- **значащей части числа;**
- **знака порядка числа;**
- **значения порядка числа.**

Подобно целочисленным данным, при $S = 0$ в самом старшем (левом) двоичном разряде, вещественное число считается неотрицательным, а при $S = 1$ - отрицательным. Очевидно, что поскольку для записи вещественных чисел используется знаковая форма, то каждому положительному числу соответствует равное ему по модулю отрицательное число, а самому большому положительному числу соответствует самое малое отрицательное. Поэтому для рассмотрения диапазона значений вещественных чисел достаточно рассмотреть лишь поведение их модулей в областях малых и больших значений.

В силу того, что для модуля числа всегда справедливо отношение $0 \leq |X|$, будем интересоваться наименьшим близким к нулю, но отличным от него, значением модуля вещественного числа.

Отдельным вопросом остается вопрос о том, что считать за вещественный нуль на цифровой ЭВМ для различных способов кодирования вещественных данных.

Для представления значащей части вещественных чисел в памяти ЭВМ используется внутренняя форма записи с фиксированной точкой, когда самый старший (левый) двоичный разряд этой значащей части соответствует -1 -й сте-

пени числа 2 . Такая система записи позволяет изображать лишь беззнаковые числа, значения которых меньше единицы, то есть по сути мантиссы чисел:

$$M = 0 \bullet m_{-1} m_{-2} \dots m_{1-q} m_{-q} . \quad (6)$$

Порядки вещественных чисел являются целыми числами, способ записи которых рассмотрен уже достаточно подробно. Знак порядка может быть учтен различным образом. Архитектура современных ЭВМ подразумевает запись порядка как беззнакового целого числа длиной в $k = p + 1$ двоичных разрядов:

$$N = n_p n_{p-1} \dots n_1 n_0 . \quad (7)$$

Истинное значение порядка получается его смещением на некоторую величину K , соответствующую примерно середине диапазона значений этого порядка.

Являясь, по сути, записью в форме с фиксированной точкой, порядок, тем не менее, обозначает форму записи с плавающей точкой для вещественного числа, поскольку определяет коэффициент в виде основания системы счисления 2 в степени порядка, на который нужно домножить мантиссу и получить истинное значение записываемого числа:

$$X = (1 - 2 \cdot S) \cdot M \cdot 2^{N-K} . \quad (8)$$

В двоичной записи вещественного числа содержатся мантисса и порядок числа со своими знаками, занимая определенные биты:

$$S n_p n_{p-1} \dots n_1 n_0 m_{-1} m_{-2} \dots m_{1-q} m_{-q} . \quad (9)$$

Такая форма записи является **ненормализованной** и предполагает значение мантиссы $0 \leq M < 1$. Ненормализован-

ная форма записи (9) необходима в ходе процесса выполнения арифметических операций, в частности, для выравнивания порядков при операциях сложения и вычитания.

Вместе с тем, для двоичной системы счисления существует уникальная возможность записывать вещественное число в **нормализованной** форме, когда старший разряд мантиисы всегда является значащим (равным единице), а производимое при этом смещение учитывается значением порядка числа. Значение мантиисы в таком случае попадает в диапазон $0.5 \leq M < 1$.

Запись нормализованного числа подразумевает наличие скрытого -1 - го разряда мантиисы, поскольку его отсутствие не нарушает однозначности двоичного представления:

$$S \ n_p \ n_{p-1} \dots n_1 \ n_0 \ m_{-2} \ m_{-3} \dots m_{1-q} \ m_{-q} . \quad (10)$$

Нормализованная форма записи (10) более экономична и позволяет хранить большее количество значащих цифр при фиксированной длине мантиисы.

2.2.1. Точность вещественных данных

Вопросы точности двоичного представления вещественных чисел можно весьма удобно и неумолительно рассмотреть на примере в реальности отсутствующего на современных ЭВМ виртуального двухбайтового (шестнадцатибитового) типа данных [3].

Будем использовать ненормализованную форму записи (9) при $q = 10$, $k = p + 1 = 5$, $K = 16$.

В этом случае максимальный порядок 11111_2

соответствует значению $31_{10} - 16_{10} = +15_{10}$,

порядок 10001_2 - значению $17_{10} - 16_{10} = +1_{10}$,

порядок 1000_2 - значению $16_{10} - 16_{10} = 0_{10}$,
порядок 01111_2 - значению $15_{10} - 16_{10} = -1_{10}$,
минимальный порядок 00000_2 - значению -16_{10} .

Так, например, числу $-0.75_{10} = -(0.5_{10} + 0.25_{10}) =$
 $= -(1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} + 0 \cdot 2^{-5} + 0 \cdot 2^{-6} + \dots)$
соответствует шестнадцатибитовое машинное представление
1 10000 1100000000 .

Числу $0.40625_{10} = 0.25_{10} + 0.125_{10} + 0.03125_{10} =$
 $= 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} + 1 \cdot 2^{-5} + 0 \cdot 2^{-6} + 0 \cdot 2^{-7} + \dots =$
 $= (1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} + 0 \cdot 2^{-5} + 0 \cdot 2^{-6} + \dots) \cdot 2^{-1}$
можно поставить в соответствие машинное представление в
виде либо **0 10000 0110100000**, либо **0 01111 1101000000**.
Последний пример весьма наглядно иллюстрирует возмож-
ность нормализации двоичного представления чисел в форме
с плавающей точкой.

Для демонстрации проблемы точности вычислений при
использовании вещественных чисел рассмотрим некоторое
из них, например, $46.5_{10} = 32_{10} + 8_{10} + 4_{10} + 2_{10} + 0.5_{10} =$
 $= 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + \dots =$
 $= (1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} +$
 $+ 1 \cdot 2^{-5} + 0 \cdot 2^{-6} + 1 \cdot 2^{-7} + 0 \cdot 2^{-8} + \dots) \cdot 2^6$,

а двоичное представление имеет вид **0 10110 1011101000** .
Поставим цель изменить значения младших значащих раз-
рядов этого числа. Для этого вполне подходят операции
сложения и вычитания. В качестве второго операнда для
таких арифметических операций возьмем, например, число
 $0.25_{10} = 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + \dots = (1 \cdot 2^{-1} + 0 \cdot 2^{-2} + \dots) \cdot 2^{-1} =$
 $= (1 \cdot 2^{-8} + 0 \cdot 2^{-9} + \dots) \cdot 2^6$.

Его допускающая нормализацию форма двоичного представ-
ления имеет вид **0 01111 1000000000** , однако нас в боль-

шей степени интересует ненормализованное двоичное представление **0 10110 0000000100**, поскольку арифметические операции сложения и вычитания можно производить лишь над операндами с выравненными порядками:

r=10	r=2		r=10	r=2
----- -----			----- -----	
46.5 0 10110 1011101000			46.75 0 10110 1011101100	
+			-	
0.25 0 10110 0000000100			46.5 0 10110 1011101000	
----- -----			----- -----	
46.75 0 10110 1011101100			0.25 0 10110 0000000100	

Аналогичные действия проведем с вдвое меньшим числом
 $0.125_{10} = 1 \cdot 2^{-3} + 0 \cdot 2^{-4} + \dots = (1 \cdot 2^{-1} + 0 \cdot 2^{-2} + \dots) \cdot 2^{-2} =$
 $= (1 \cdot 2^{-9} + 0 \cdot 2^{-10} + \dots) \cdot 2^6 :$

r=10	r=2		r=10	r=2
----- -----			----- -----	
46.5 0 10110 1011101000			46.625 0 10110 1011101010	
+			-	
0.125 0 10110 0000000010			0.125 0 10110 0000000010	
----- -----			----- -----	
46.625 0 10110 1011101010			46.5 0 10110 1011101000	

Последнее корректное сложение с еще вдвое меньшим числом

$0.0625_{10} = 1 \cdot 2^{-4} + 0 \cdot 2^{-5} + \dots = (1 \cdot 2^{-1} + 0 \cdot 2^{-2} + \dots) \cdot 2^{-3} =$
 $= (1 \cdot 2^{-10} + 0 \cdot 2^{-11} + \dots) \cdot 2^6$

выглядит следующим образом:

r=10		r=2
-----		-----
46.5		0 10110 1011101000
+		
0.0625		0 10110 0000000001
-----		-----
46.5625		0 10110 1011101001

Однако, дальнейшее уменьшение вдвое второго слагаемого $0.03125_{10} = 1 \cdot 2^{-5} + 0 \cdot 2^{-6} + \dots = (1 \cdot 2^{-1} + 0 \cdot 2^{-2} + \dots) \cdot 2^{-4} = (1 \cdot 2^{-11} + 0 \cdot 2^{-12} + \dots) \cdot 2^6$

приводит к вытеснению значащего разряда за пределы отведенного под мантиссу битового поля и оказывается эквивалентным операции сложения с нулем, не изменяя значения первого слагаемого:

r=10		r=2
-----		-----
46.5		0 10110 1011101000 (0)
+		
0.03125		0 10110 0000000000 (1)
-----		-----
46.53125		0 10110 1011101000 (1)

Последний результат демонстрирует дискретный характер двоичного представления вещественных данных на цифровой ЭВМ, вытекающий из конечности длины этого представления. Мантиссы чисел 46.5_{10} и 46.5625_{10} отличаются на значение самого младшего (правого) двоичного разряда и не допускают возможности записи любого вещественного числа из промежутка между ними.

То же самое можно сказать о паре чисел 46.5_{10}

и $46.4375_{10} = 46.5_{10} - 0.0625_{10}$, двоичное представление последнего из которых имеет вид **0 10110 1011100111**.

Число $2^{-4} = 0.0625_{10}$ будет являться дискретом для всех чисел, двоичный порядок которых $N - K = 6$, то есть чисел в диапазоне от $2^5 = 32_{10}$ или **0 10110 1000000000** до $2^6 - 2^{-4} = 63.9375_{10}$ или **0 10110 1111111111**.

Приведем ранее рассмотренный фрагмент этого диапазона:

---	+	-----	+	-----	+	-----	+	-----	+	---	>
46.25		46.3125		46.375		46.4375		46.5			

---	+	-----	+	-----	+	-----	+	-----	+	---	>
46.5		46.5625		46.625		46.6875		46.75			

В соответствии с положениями раздела 1.4 настоящей разработки, каждое число рассматриваемого диапазона определяется с точностью до половины дискрета и, следовательно, предельная абсолютная погрешность этих чисел $\Delta X = 2^{-5}$. Предельная относительная погрешность в этом случае не превышает $\delta X = 2^{-5}/2^5 = 2^{-10} = 1/1024 \approx 10^{-3}$.

Одновременно эта величина соответствует весовому значению самого младшего (правого) двоичного разряда мантисы $W_{-q} = 2^{-10}$. Таким образом получается, что длина двоичной мантисы определяет относительную погрешность представления вещественных данных, которая в данном случае составляет **10** точных двоичных цифр или **3** точные десятичные цифры, что подтверждается приведенными выше примерами.

Нетрудно убедиться, что относительная погрешность представления вещественных данных на цифровой ЭВМ есть

величина постоянная, в то время как абсолютная погрешность зависит от значения числа и меняется в два раза при изменении на единицу значения порядка.

Число $2^{-10} = 0.0009765625_{10}$ будет являться дискретом для всех чисел, двоичный порядок которых

$N - K = 0$, то есть чисел в диапазоне от $2^{-1} = 0.5_{10}$ или $0\ 10000\ 1000000000$ до $2^0 - 2^{-10} = 0.9990234375_{10}$ или $0\ 10000\ 1111111111$.

Число $2^{-9} = 0.001953125_{10}$ будет являться дискретом для всех чисел, двоичный порядок которых $N - K = 1$, то есть чисел в диапазоне от $2^0 = 1_{10}$ или $0\ 10001\ 1000000000$ до $2^1 - 2^{-9} = 1.998046875_{10}$ или $0\ 10001\ 1111111111$.

Шестнадцатибитовый тип данных не может устраивать реальных пользователей ЭВМ как по точности, так и по диапазону $|X| < 2^{15} = 32768_{10}$ представления вещественных чисел. Поэтому на практике используются более длинные двоичные представления чисел с плавающей точкой.

2.2.2. Вещественные данные DEC PDP-11

ЭВМ PDP-11 фирмы DEC используют нормализованную форму (10) записи вещественных данных [3].

2.2.2.1. Числа одинарной степени точности размещаются в четырехбайтовых (тридцатидвухбитовых) ячейках памяти при $q = 24$, $k = p + 1 = 8$, $K = 128$.

В этом случае максимальный порядок 11111111_2 соответствует значению $255_{10} - 128_{10} = +127_{10}$,
порядок 10000001_2 - значению $129_{10} - 128_{10} = +1_{10}$,
порядок 10000000_2 - значению $128_{10} - 128_{10} = 0_{10}$,
порядок 01111111_2 - значению $127_{10} - 128_{10} = -1_{10}$,
минимальный порядок 00000000_2 - значению -128_{10} .

Так, например, числу $-0.75_{10} = -(0.5_{10} + 0.25_{10}) =$
 $= -(1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} + 0 \cdot 2^{-5} + 0 \cdot 2^{-6} + \dots)$
ставится в соответствие четырехбайтовое нормализованное
представление **1 10000000 100000000000000000000000** .

Числу $0.40625_{10} = 0.25_{10} + 0.125_{10} + 0.03125_{10} =$
 $= (1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} + 0 \cdot 2^{-5} + 0 \cdot 2^{-6} + \dots) \cdot 2^{-1}$
ставится в соответствие четырехбайтовое нормализованное
представление **0 01111111 101000000000000000000000** .

Запись мантиисы нормализованного числа фактически содержит **23** двоичных разряда. Тем не менее, весовое значение самого младшего (правого) двоичного разряда мантиисы $W_{-q} = 2^{-24}$. Таким образом, относительная погрешность представления вещественных данных одинарной степени точности на ЭВМ рассматриваемого типа составляет $\delta X = 2^{-24} \approx 5.96046447754 \cdot 10^{-8}$, что соответствует **7** точным десятичным цифрам.

Число $2^{-24} \approx 5.96 \cdot 10^{-8}$ будет являться дискретом для всех чисел, двоичный порядок которых $N - K = 0$, то есть чисел в диапазоне от $2^{-1} = 0.5_{10}$ или

0 10000000 000000000000000000000000

до $2^0 - 2^{-24} \approx 0.9999999404_{10}$ или

0 10000000 111111111111111111111111 .

Число $2^{-23} \approx 1.192 \cdot 10^{-7}$ будет являться дискретом для всех чисел, двоичный порядок которых $N - K = 1$, то есть чисел в диапазоне от $2^0 = 1_{10}$ или

0 10000001 000000000000000000000000

до $2^1 - 2^{-23} \approx 1.9999998808_{10}$ или

0 10000001 111111111111111111111111 .

Рассмотрим диапазон значений, которые могут принимать вещественные числа одинарной степени точности.

Число $2^{103} \approx 0.10141205 \cdot 10^{32}$ будет являться дискретом для всех чисел с максимально возможным порядком $N - K = 127$, то есть чисел в диапазоне

от $2^{126} \approx 0.85070592 \cdot 10^{38}$ или

0 11111111 000000000000000000000000

до $2^{127} - 2^{103} \approx 1.7014117332 \cdot 10^{38}$ или

0 11111111 111111111111111111111111 .

Последнее приведенное вещественное число является абсолютным максимумом для рассматриваемого способа двоичного представления. Превышение его значения диагностируется ЭВМ как ошибка переполнения.

Архитектура ЭВМ рассматриваемого класса устроена таким образом, что если $N = 0$ или $N - K = -128$, то число считается равным нулю ($X = 0.0$) вне зависимости от того, чему равны знак числа S и мантисса числа

М. Таким образом, по сути дела, вводится машинный вещественный нуль. Поэтому минимальным отличным от нуля вещественным числом одинарной степени точности будет число с порядком $N - K = -127$ и минимальным значением нормализованной мантииссы $M = 0.5_{10}$, то есть число $2^{-128} \approx 0.2938735877 \cdot 10^{-38}$ или

0 00000001 000000000000000000000000 .

Число $2^{-151} \approx 0.35032462 \cdot 10^{-45}$ будет являться дискретом для всех чисел с минимальным значащим порядком $N - K = -127$, то есть чисел в диапазоне от 2^{-128} до $2^{-127} - 2^{-151} \approx 0.5877471404 \cdot 10^{-38}$ или

0 00000001 111111111111111111111111 .

Диапазон значений вещественных чисел одинарной степени точности, таким образом, может быть охарактеризован соотношением $2^{-128} \leq |X| < 2^{127}$.

2.2.2.2. Числа двойной степени точности размещаются в восьмибайтовых (шестидесятичетырехбитовых) ячейках памяти при $q = 56$, $k = p + 1 = 8$, $K = 128$.

Запись порядка таких чисел аналогична числам с одинарной степенью точности, поэтому аналогичны их диапазоны значений.

Запись мантииссы нормализованного числа двойной степени точности фактически содержит **55** двоичных разрядов. Тем не менее, весовое значение самого младшего (правого) двоичного разряда мантииссы $W_{-q} = 2^{-56}$. Таким образом, относительная погрешность представления вещественных данных двойной степени точности на ЭВМ рассматриваемого типа составляет $\delta X = 2^{-56} \approx 0.1388 \cdot 10^{-16}$, что соответствует **16** точным десятичным цифрам.

Вещественное число $1/3$, таким образом, в машинном представлении с одинарной степенью точности представляет собой $\approx 0.33333333_{10}$, а в представлении с двойной степенью точности $\approx 0.3333333333333333_{10}$, что, как нетрудно заметить, далеко не одно и то же.

2.2.3. Вещественные данные IBM PC

ЭВМ IBM PC помимо нормализованной формы (10) записи вещественных данных, для близких по модулю к нулю чисел используют и ненормализованную форму (9). Следует отметить, что излагаемый в данном разделе материал несколько отличается от такового в книге [4], поскольку там допускается отступление от общепринятого понятия мантиссы, что методически спорно.

2.2.3.1. Числа одинарной степени точности размещаются в четырехбайтовых (тридцатидвухбитовых) ячейках памяти при $q = 24$, $k = p + 1 = 8$, $K = 126$.

Для ЭВМ IBM PC состоящий только из одних единиц порядок 11111111_2 рассматривается как сигнал переполнения, поэтому максимально возможный порядок 11111110_2 соответствует значению $254_{10} - 126_{10} = +128_{10}$,
 порядок 10000000_2 - значению $128_{10} - 126_{10} = +2_{10}$,
 порядок 01111111_2 - значению $127_{10} - 126_{10} = +1_{10}$,
 порядок 01111110_2 - значению $126_{10} - 126_{10} = 0_{10}$,
 порядок 01111101_2 - значению $125_{10} - 126_{10} = -1_{10}$,
 минимальный порядок 00000000_2 - значению -126_{10} .

Так, например, числу $-0.75_{10} = -(0.5_{10} + 0.25_{10}) = -(1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} + 0 \cdot 2^{-5} + 0 \cdot 2^{-6} + \dots)$

ставится в соответствие четырехбайтовое нормализованное представление **1 01111110 100000000000000000000000** .

Числу $0.40625_{10} = 0.25_{10} + 0.125_{10} + 0.03125_{10} =$
 $= (1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} + 0 \cdot 2^{-5} + 0 \cdot 2^{-6} + \dots) \cdot 2^{-1}$
 ставится в соответствие четырехбайтовое нормализованное представление **0 01111101 101000000000000000000000** .

Запись мантииссы нормализованного числа фактически содержит **23** двоичных разряда. Тем не менее, весовое значение самого младшего (правого) двоичного разряда мантииссы $W_q = 2^{-24}$. Таким образом, относительная погрешность представления вещественных данных одинарной степени точности на ЭВМ рассматриваемого типа составляет $\delta X = 2^{-24} \approx 5.96046447754 \cdot 10^{-8}$, что соответствует **7** точным десятичным цифрам.

Число $2^{-24} \approx 5.96 \cdot 10^{-8}$ будет являться дискретом для всех чисел, двоичный порядок которых $N - K = 0$, то есть чисел в диапазоне от $2^0 = 0.5_{10}$ или

0 01111110 000000000000000000000000

до $2^0 - 2^{-24} \approx 0.9999999404_{10}$ или

0 01111110 111111111111111111111111 .

Число $2^{-23} \approx 1.192 \cdot 10^{-7}$ будет являться дискретом для всех чисел, двоичный порядок которых $N - K = 1$, то есть чисел в диапазоне от $2^0 = 1_{10}$ или

0 01111111 000000000000000000000000

до $2^1 - 2^{-23} \approx 1.9999998808_{10}$ или

0 01111111 111111111111111111111111 .

Рассмотрим диапазон значений, которые могут принимать вещественные числа одинарной степени точности.

Число $2^{104} \approx 0.202824096 \cdot 10^{32}$ будет являться дис-

кретом для всех чисел с максимально возможным порядком $N - K = 128$, то есть чисел в диапазоне

от $2^{127} \approx 1.7014118346 \cdot 10^{38}$ или

0 11111110 000000000000000000000000

до $2^{128} - 2^{104} \approx 3.4028234664 \cdot 10^{38}$ или

0 11111110 111111111111111111111111.

Последнее приведенное вещественное число является абсолютным максимумом для рассматриваемого способа двоичного представления. Превышение его значения диагностируется ЭВМ как ошибка переполнения.

Архитектура ЭВМ рассматриваемого класса устроена таким образом, что если $N = 0$ или $N - K = -126$, то число записывается в ненормализованной форме (9).

Поэтому минимальными вещественными числами одинарной степени точности в нормализованной форме (10) будут числа с порядком $N = +1$ или $N - K = -125$.

Число $2^{-149} \approx 1.401298464 \cdot 10^{-45}$ будет являться дискретом для всех чисел со значащим порядком $N - K = -125$, то есть чисел в диапазоне от $2^{-126} \approx 1.17549435 \cdot 10^{-38}$

или **0 00000001 000000000000000000000000**

до $2^{-125} - 2^{-149} \approx 2.35098856 \cdot 10^{-38}$

или **0 00000001 111111111111111111111111**.

Это же самое число 2^{-149} будет минимальным отличным от нуля вещественным числом одинарной степени точности, поскольку оно имеет минимальный порядок $N - K = -126$ и минимальное значение ненормализованной мантиссы:

0 00000000 000000000000000000000001.

Наконец, то же самое число 2^{-149} будет являться дискретом для всех ненормализованных чисел с минимальным поряд-


```

DO  I = 1, ISUM
    RSUM = RSUM + F()
END DO
I = RSUM
WRITE(*,66)  ISUM, I
66  FORMAT(I9,' =?', I9)
STOP
END

```

```

REAL*4  FUNCTION  F()
F = 1.0
RETURN
END

```

```

#include<math.h>
#include<stdio.h>
float  f( void ) { return 1.0; }
int main( void ) {
    long  l, lsum;
    float fsum;
    lsum = pow( 2L, 24L );
    printf( "2**24 = %8ld\n", lsum );
    fsum = 0.0;
    lsum = 17000000L;
    for( l=1L; l<=lsum; l++ ) fsum += f();
    printf( "%8ld =? %8.0f\n", lsum, fsum );
    return 0;
}

```

```

2**24 = 16777216
17000000 =? 16777216

```

Результаты выполнения программ целиком укладываются в рамки представлений о конечной длине значащей части вещественного числа.

Достигнув значения $2^{24} = 16\,777\,216_{10}$ или

0 10010111 000000000000000000000000

сумма перестает меняться, так как при прибавлении к ней еще одной единицы, в силу выравнивания порядков, происходит выход значащей части этой единицы за пределы длины мантиссы. Вполне возможно было бы получить значение

$2^{24} + 2^1 = 16\,777\,218_{10}$ или

0 10010111 000000000000000000000001

и это демонстрирует путь выхода из ситуаций подобного рода, когда вначале вычисляются частичные суммы, а затем они объединяются во все более крупные, пока не достигается конечный корректный результат. Важно при этом на каждом этапе решения задачи оставаться в рамках разрядной сетки используемой ЭВМ.

2.2.3.2. Числа двойной степени точности размещаются в восьмибайтовых (шестидесятичетырехбитовых) ячейках памяти при $q = 53$, $k = p + 1 = 11$, $K = 1022$.

На ЭВМ IBM PC состоящий только из одних единиц порядок **1111111111₂** рассматривается как сигнал переполнения, поэтому максимально возможный порядок **11111111110₂** соответствует значению $2046_{10} - 1022_{10} = +1024_{10}$,
 порядок **10000000000₂** - значению $1024_{10} - 1022_{10} = +2_{10}$,
 порядок **01111111111₂** - значению $1023_{10} - 1022_{10} = +1_{10}$,
 порядок **01111111110₂** - значению $1022_{10} - 1022_{10} = 0_{10}$,
 порядок **01111111101₂** - значению $1021_{10} - 1022_{10} = -1_{10}$,
 минимальный порядок **00000000000₂** - значению -1022_{10} .

Так, например, числу $-0.75_{10} = -(0.5_{10} + 0.25_{10}) =$
 $= -(1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} + 0 \cdot 2^{-5} + 0 \cdot 2^{-6} + \dots)$
 ставится в соответствие восьмибайтовое нормализованное
 представление **1 0111111110 100000000000000000...0**.

Числу $0.40625_{10} = 0.25_{10} + 0.125_{10} + 0.03125_{10} =$
 $= (1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} + 0 \cdot 2^{-5} + 0 \cdot 2^{-6} + \dots) \cdot 2^{-1}$
 ставится в соответствие восьмибайтовое нормализованное
 представление **0 0111111101 101000000000000000...0**.

Запись мантиисы нормализованного числа фактически
 содержит **52** двоичных разряда. Тем не менее, весовое зна-
 чение самого младшего (правого) двоичного разряда ман-
 тиссы $W_q = 2^{-53}$. Таким образом, относительная по-
 грешность представления вещественных данных двойной
 степени точности на ЭВМ рассматриваемого типа составля-
 ет $\delta X = 2^{-53} \approx 1.11022302463 \cdot 10^{-16}$, что соответствует
15 точным десятичным цифрам.

Число $2^{-53} \approx 1.11 \cdot 10^{-16}$ будет являться дискретом
 для всех чисел, двоичный порядок которых $N - K = 0$, то
 есть чисел в диапазоне от $2^{-1} = 0.5_{10}$ или
0 0111111110 000000000000000000000000000000...0
 до $2^0 - 2^{-53} \approx 0.999999999999999889_{10}$ или
0 0111111110 111111111111111111111111111111...1.

Число $2^{-52} \approx 2.22 \cdot 10^{-16}$ будет являться дискретом
 для всех чисел, двоичный порядок которых $N - K = 1$, то
 есть чисел в диапазоне от $2^0 = 1_{10}$ или
0 0111111111 000000000000000000000000000000...0
 до $2^1 - 2^{-52} \approx 1.999999999999999778_{10}$ или
0 0111111111 111111111111111111111111111111...1.

Рассмотрим диапазон значений, которые могут прини-

Число $2^{971} \approx 199.584030953472 \cdot 10^{290}$ будет являться дискретом для всех чисел с максимально возможным порядком $N - K = 1024$, то есть чисел в диапазоне от $2^{1023} \approx 0.898846567431157954 \cdot 10^{308}$ или
0 11111111110 000000000000000000000000000000...0
до $2^{1024} - 2^{971} \approx 1.79769313486231571 \cdot 10^{308}$ или
0 11111111110 111111111111111111111111111111...1.

Последнее приведенное вещественное число является абсолютным максимумом для рассматриваемого способа двоичного представления. Превышение его значения диагностируется ЭВМ как ошибка переполнения.

Число $\mathbf{2^{-1074} \approx 0.494065645841246544 \cdot 10^{-323}}$ будет
являться дискретом для всех чисел со значащим порядком
N – K = -1021, то есть чисел в диапазоне
от $\mathbf{2^{-1022} \approx 2.22507385850720138 \cdot 10^{-308}}$ или
0 00000000001 0000000000000000000000000000...0
до $\mathbf{2^{-1021} - 2^{-1074} \approx 4.45014771701440227 \cdot 10^{-308}}$ или
0 00000000001 1111111111111111111111111111...1.

Это же самое число 2^{-1074} будет минимальным отличным от нуля вещественным числом двойной степени точности, поскольку оно имеет минимальный порядок $N - K = -1022$ и минимальное значение ненормализованной мантиссы: 0 000000000000 0 ... 00000000000000000000000000000001.

до $\mathbf{2^{-1022} - 2^{-1074} \approx 2,22507385850720089 \cdot 10^{-308}}$ или
0 000000000000 111111111111111111111111111111 ... 1.

Подобно примеру с данными одинарной степени точности, можно показать, что при суммировании единиц двойной степени точности результат не может превышать $2^{53} = 9\,007\,199\,254\,740\,992_{10}$.

Следует заметить, что для C - варианта программы использование нестандартной функции ltoa() компилятора Borland C++ объясняется лишь желанием продемонстрировать короткий и простой способ решения такой задачи в ущерб его общности и мобильности.

```

PROGRAM    R__BIT
INTEGER * 1  II(4), IL(8)
R E A L * 4  R
R E A L * 8  D
EQUIVALENCE ( R, II(1) ), ( D, IL(1) )
11 WRITE(*,25)
25 FORMAT(' VALUE : ', $)
READ(*, *, ERR = 11) D
IF( DABS( D ) .LE. 3.4028235D+38 ) THEN
    R = D
    WRITE(*,55)
55  FORMAT(' REAL*4 = ', $)
    CALL BIT( 4, II )
END IF
WRITE(*,85)
85  FORMAT(' REAL*8 = ', $)
    CALL BIT( 8, IL )
GOTO 11
END
SUBROUTINE BIT( NB, IB )
INTEGER      I, J, JB, KB, MB, NB
INTEGER*1    IB(8), IC(8)
DO  I = NB, 1, -1
    JB = IB(I)
    IF( JB .LT. 0 ) JB = JB + 256
    DO  J = 1, 8
        MB = 2 ** ( 8 - J )
        KB = JB / MB
        JB = JB - KB * MB
        IC(J) = KB
    END DO
END DO

```

```
VALUE : -1.3333333333333333  
REAL*4 = 10111111101010101010101010101011  
REAL*8 = 101111111111  
010101010101010101010101010101010101010101010101010101010101
```

52

```
void lout() {  
    int k, n; char s[33];  
        ltoa( *l, s, 2 ); n = strlen( s );  
        for( k=0; k<32-n; k++ ) printf( "0" );  
        for( k=0; k<n; k++ ) printf( "%c", s[k] );  
    return;  
}  
  
value : -1.3333333333333333  
float : 10111111101010101010101010101011  
double: 101111111111  
01010101010101010101010101010101010101010101010101010101
```

2.3. Двоичное представление символьных данных

Алфавитно-цифровые символы хранятся в памяти ЭВМ в виде **байт**, позволяющих различать **256** уникальных цифровых представления каждого отдельного символа.

Большинство современных ЭВМ используют код **ASCII**.

Приведем краткую информацию о его основной таблице - семибитовом подмножестве полного набора символов.

Первые **32** комбинации со значениями кодов от **0₁₀** до **31₁₀** представляют собой управляющие символы. Далее, начиная с пробела (SPACEBAR), следуют печатные символы:

ASCII	r = 2	r = 10	ASCII	r = 2	r = 10
	00100000	32	4	00110100	52
!	00100001	33	5	00110101	53
"	00100010	34	6	00110110	54
#	00100011	35	7	00110111	55
\$	00100100	36	8	00111000	56
%	00100101	37	9	00111001	57
&	00100110	38	:	00111010	58
'	00100111	39	;	00111011	59
(00101000	40	<	00111100	60
)	00101001	41	=	00111101	61
*	00101010	42	>	00111110	62
+	00101011	43	?	00111111	63
,	00101100	44	@	01000000	64
-	00101101	45	A	01000001	65
.	00101110	46	B	01000010	66
/	00101111	47	C	01000011	67
0	00110000	48	D	01000100	68
1	00110001	49	E	01000101	69
2	00110010	50	F	01000110	70
3	00110011	51	G	01000111	71

ASCII	r = 2	r = 10	ASCII	r = 2	r = 10
H	01001000	72	d	01100100	100
I	01001001	73	e	01100101	101
J	01001010	74	f	01100110	102
K	01001011	75	g	01100111	103
L	01001100	76	h	01101000	104
M	01001101	77	i	01101001	105
N	01001110	78	j	01101010	106
O	01001111	79	k	01101011	107
P	01010000	80	l	01101100	108
Q	01010001	81	m	01101101	109
R	01010010	82	n	01101110	110
S	01010011	83	o	01101111	111
T	01010100	84	p	01110000	112
U	01010101	85	q	01110001	113
V	01010110	86	r	01110010	114
W	01010111	87	s	01110011	115
X	01011000	88	t	01110100	116
Y	01011001	89	u	01110101	117
Z	01011010	90	v	01110110	118
[01011011	91	w	01110111	119
\	01011100	92	x	01111000	120
]	01011101	93	y	01111001	121
^	01011110	94	z	01111010	122
_	01011111	95	{	01111011	123
`	01100000	96		01111100	124
a	01100001	97	}	01111101	125
b	01100010	98	~	01111110	126
c	01100011	99	DEL	01111111	127

2.4. Двоичное представление логических данных

В некоторых языках программирования существует специально выделенный логический тип данных.

Так в языке FORTRAN можно использовать байтовые (**LOGICAL * 1**), двухбайтовые (**LOGICAL * 2**) и четырехбайтовые (**LOGICAL * 4**) логические данные.

На ЭВМ IBM PC булево значение **.FALSE.** кодируется в каждом из этих случаев так:

00000000₂ ;
00000000 00000000₂ ;
00000000 00000000 00000000 00000000₂ .

Соответственно булево значение **.TRUE.** кодируется так:

00000001₂ ;
00000000 00000001₂ ;
00000000 00000000 00000000 00000001₂ .

Как альтернатива такому жесткому определению выступает правило языка C, где логическое значение “**Ложь**” представляется целым нулевым значением, а значение “**Истина**” представляется любым ненулевым значением.

ЛИТЕРАТУРА

- 1. Математическая энциклопедия** / Гл. ред. И.М.Виноградов, т. 1 - 5. - М. : Советская Энциклопедия, 1985.
- 2. Бронштейн И.Н., Семендяев К.А.** Справочник по математике для инженеров и учащихся втузов. - М. : Наука, 1986.
- 3. Экхауз Р., Моррис Л.** Мини - ЭВМ : Организация и программирование / Пер. с англ. - М. : Финансы и статистика, 1983.
- 4. Морс С.П., Алберт Д.Д.** Архитектура микропроцессора 80286 / Пер. с англ. - М. : Радио и связь, 1990.

ПРЕДСТАВЛЕНИЕ ДАННЫХ ЦИФРОВЫХ ЭВМ

Методическая разработка
для студентов радиофизического факультета ННГУ

Составитель
Владимир Александрович Савин

Подписано к печати Формат 60x84 $\frac{1}{16}$.
Печать офсетная. Бумага оберточная. Усл. печ. л. , .
Тираж экз. Заказ Бесплатно.

Нижегородский государственный университет
им. Н.И.Лобачевского. 603600, Н.Новгород, пр. Гагарина, 23.

Типография ННГУ. 603000, Н.Новгород,
ул. Б.Покровская, 37.
