КОДИРОВАНИЕ

Распространено ошибочное мнение, что ЭВМ — быстродействующий арифмометр. В действительности же она вычислительная в том смысле, что оперирует непосредственно только с числами. А нам уже известно, что эти операции не всегда сводятся к арифметическим. Числа – удобная форма представления информации, природа которой может быть самой различной: от состояний датчиков в системах управления до сообщений на привычном нам естественном языке.

Как представить всю эту информацию с помощью чисел, каковы наиболее удобные формы представления самих чисел, наконец, как придать обработанной информации удобную для использования форму — тема этой главы. В ней будут рассмотрены *кодирование* (представление дискретной информации в стандартных символических формах) и *декодирование*, т. е. обратный процесс.

Системы счисления

То, что любая информация может быть выражена числами, выдвигает на первый план проблему кодирования самих чисел. Примечательно, что эта проблема весьма тщательно разработана всем ходом развития нашей цивилизации. «Все есть число»,— говорили пифагорийцы, подчеркивая практическую роль чисел в человеческой деятельности. Известно множество способов представления чисел. В любом случае число изображается символом или группой символов (словом) некоторого алфавита. Будем называть такие символы *цифрами*, символические изображения чисел — *кодами*, а правила их получения — *системами счисления* (кодирования).

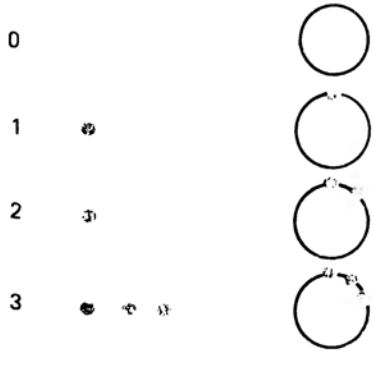
Простейшая и самая древняя система счисления использует для записи любых чисел всего один символ (рис. 3.1). Длина записи числа при таком кодировании прямо связана с его величиной, что роднит этот способ с геометрическим представлением чисел в виде отрезков. Сами того не сознавая, этим кодом пользуются малыши, показывая пальцами свой возраст. С ним же тесно связан *код «1 из n»* (рис. 3.2). Изображение чисел точками на оси, широко используемое в математике, фактически представляет их запись в коде «1 из *n*». Показания стрелочных часов, измерительных приборов определяются одной позицией из *n* возможных. Нажатием кнопки в лифте указывается нужный номер этажа в коде «1 из *n*».

Если символ в записи чисел играет роль, зависящую от его позиции, то системы счисления, обладающие этим свойством, называются позиционными. Из всех позиционных систем наибольший практический интерес представляют те, в которых получение кода числа (кодирование) и обратный процесс (декодирование), описываются несложными правилами, например сводятся к арифметическим действиям.

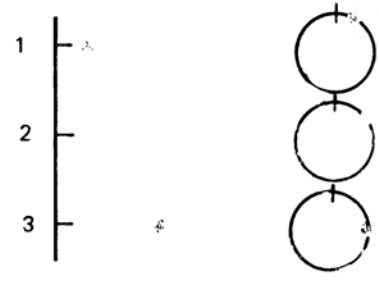
Простейшие системы такого рода называются *взвешенными*. В них процедура декодирования осуществляется следующим образом: каждая цифра кода умножается на коэффициент *(вес)* позиции, и все результаты складываются (рис. 3.3). Веса отдельных позиций обычно убывают, если читать цифры кода слева направо. Левая цифра кода числа обычно читается и произносится первой, и если она самая «весомая», то уже по ней можно примерно оценить величину всего числа. Поэтому крайняя левая позиция кода числа называется старшей, а крайняя правая — младшей.

Рис. 3.1

При изображении чисел в простейшем коде вместо точки может быть использован произвольный символ, например * или 1.



Если задано «начало отсчета» — место, начиная с которого пишутся символы простейшего кода, то все его символы, кроме последнего, можно опустить. Величина числа, таким образом, определяется позицией, которую занимает единственный символ по отношению к началу записи.



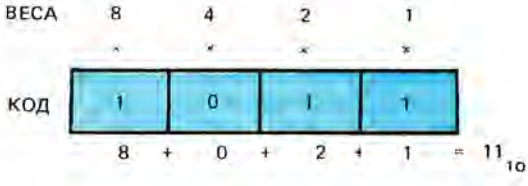


Рис. 3.3. При декодировании кода 1011 в системе с весами 8—4—2—1 получается число 11. Говоря строго, 11 — это тоже код. Обычно мы отождествляем сами числа с их записями в десятичной системе счисления (кодирования!), к которым привыкли с детства. Заметим, что и все необходимые для декодирования вычисления проведены по правилам обычной, т. е. десятичной арифметики.

Веса отдельных позиций можно выбрать совершенно произвольно, получая таким образом различные системы счисления. Не все из них одинаково удобны. С особенностями выбора весов познакомимся при рассмотрении десятичной системы счисления.

3.3. Как мы считаем?

Изобретение десятичной системы счисления приписывают древним арабам, развитие — индусам. Появление ее в Европе датируется примерно 1200 г. н. э. Десятичными цифрами выражаются время, номера домов и телефонов, цены, бюджет, показания приборов, на них базируется метрическая система мер.

В чем же секрет такого успеха десятичной системы, конкуренции с которой не выдержало подавляющее большинство остальных систем?

Процесс декодирования десятичных чисел мы обычно осуществляем «автоматически». Запись любого числа, скажем 1987, мы расшифровываем так: одна тысяча, плюс 9 сотен, плюс 8 десятков, плюс 7 единиц. Следовательно, наша система — взвешенная, а веса в ней — последовательные целые степени числа 10 ($1 = 10^0$, $10 = 10^1$, $100 = 10^2$, $1000 = 10^3$, ...). Главная особенность этого выбора весов состоит в том, что между всеми возможными числами и кодами существует взаимно однозначное соответствие, а арифметические операции над числами производятся при помощи небольшого числа несложных правил.

Не будем перечислять эти правила, они известны любому первокласснику. В их основе две таблицы — умножения и сложения для всех возможных комбинаций одноразрядных чисел (цифр) и *правило переноса:* если в результате сложения двух цифр получается число, которое больше или равно 10, то его можно записать только с помощью двух цифр с соответствующими весами.

Именно в простоте арифметических действий, операций кодиро-

вания и декодирования кроется успех десятичной системы счисления. Правила обращения с десятичными числами, изучаемые еще в детском возрасте, в результате повседневной практики усваиваются настолько крепко, что мы пользуемся ими скорее подсознательно. По этой причине многие люди даже не догадываются о существовании других систем счисления.

3.3. А почему десятичная?

Попробуем обобщить наши представления о десятичной системе счисления. Любой код, обозначаемый последовательностью произвольных цифр, например 748, интерпретируется нами как число, равное

$$7 \cdot 10^2 + 4 \cdot 10^1 + 8 \cdot 10^0.$$

Естественным обобщением такого представления чисел является кодирование дробей: запись вида 0,365 означает не что иное, как

$$\frac{365}{1000} = \frac{3}{10} + \frac{6}{100} + \frac{5}{1000}.$$

Итак, в общем случае символическая запись

$$a_n \ldots a_1 a_0,\ a_{-1} a_{-2} \ldots a_{-m}$$

расшифровывается как

$$a_n \cdot 10^n + \ldots + a_1 \cdot 10 + a_0 + a_{-1} \cdot 10^{-1} + \ldots a_{-m} \cdot 10^{-m},$$

а система счисления называется *позиционной с основанием 10*, или просто десятичной.

Очевидно, что имеют полное право на существование позиционные системы с произвольным основанием, а в некоторых отношениях, если отвлечься от привычки, они могут оказаться и удобнее десятичных.

В системе счисления с произвольным основанием b запись

$$a_n \ldots a_1 a_0,\ a_{-1} a_{-2} \ldots a_{-m}$$

соответствует десятичному числу, которое находится как

$$a_n \cdot b^n + \ldots a_1 \cdot b + a_0 + a_{-1} \cdot b^{-1} + a_{-2} \cdot b^{-2} + \ldots a_{-m} \cdot b^{-m}$$

по правилам обычной (десятичной) арифметики.

Так,

$$123_8 = 1 \cdot 8^2 + 2 \cdot 8 + 3 = 83$$
$$1010_2 = 1 \cdot 2^3 + 1 \cdot 2^1 = 10,$$

где индексы в записи чисел означают используемую систему счисления.

Обратный перевод обычно осуществляют т. н. *методом деления*, в ходе которого нужно выписать остатки от последовательного деления десятичного числа на величину основания b и прочитать их, начиная с последнего. Так, производя последовательное деление числа 19 сначала на 8, а затем на 2:

	остаток
$19 : 8 = 2$	3
$2 : 8 = 0$	2

	остаток
$19 : 2 = 9$	1
$9 : 2 = 4$	1
$4 : 2 = 2$	0
$2 : 2 = 1$	0
$1 : 2 = 0$	1,

находим, что $19 = 23_8 = 10011_2$.

Для перевода десятичных дробей (чистых, т. е. без целой части) используют *метод умножения*. При этом исходная дробь умножается на величину b. Целая часть результата представляет собой первую цифру после запятой в искомой дроби. Дробная часть результата снова умножается на b и т. д. Так, например,

$$0{,}15625 \cdot 8 = 1{,}25, \qquad 0{,}75 \cdot 2 = 1{,}5,$$
$$0{,}25 \cdot 8 = 2{,}0; \qquad 0{,}5 \cdot 2 = 1{,}0$$

и, следовательно, $0{,}15625 = 0{,}12_8$, а $0{,}75 = 0{,}11_2$.

С целью перевода в двоичную систему десятичных чисел не очень большой длины можно просто подобрать различные степени числа 2, дающие в сумме исходное число. Так, $10{,}25 = 2^3 + 2^1 + 2^{-2} = 1010{,}01_2$.

Арифметические операции над числами в недесятичных системах счисления производятся по тем же правилам, что и над обычными числами. Небольшая тренировка позволяет обходиться и без таблиц сложения и умножения: все необходимые варианты легко воспроизводятся в уме. Главное при этом — помнить, что перенос в следующий разряд возникает тогда, когда результат действия над двумя цифрами равен или больше основания. Ниже приведены примеры выполнения четырех арифметических действий над числами в восьмеричной системе счисления.

$$\begin{array}{r} 364 \\ +\,525 \\ \hline 1111 \end{array} \qquad \begin{array}{r} 525 \\ -\,364 \\ \hline 141 \end{array} \qquad \begin{array}{r} 364 \\ \times\,525 \\ \hline 2304 \\ +\;750 \\ 2304 \\ \hline 242404 \end{array} \qquad \begin{array}{r|l} 525 & 37 \\ \cline{2-2} -\,37 & 13 \\ \cline{1-1} 135 & \\ -\,135 & \\ \cline{1-1} 0 & \end{array}$$

Рекомендуем читателю внимательно проанализировать «необычные» ситуации и убедиться в правильности выполненных операций путем перевода исходных чисел и результатов в десятичную систему.

3.4. Алфавит для ЭВМ, или если бы первоклассником была машина

Двумя основными функциями ЦА являются хранение и преобразование символов (см. гл. 2). Обсудим особенности реализации этих функций в различных системах счисления. Прежде всего заметим, что в качестве *запоминающего элемента* (ЗЭ) для хране-

ния любой цифры позиционной системы счисления с основанием b может служить произвольное устройство, имеющее ровно b устойчивых состояний, каждое из которых ставится в соответствие определенной цифре. С этой целью могут использоваться обычные механические переключатели, имеющие нужное число положений. В арифмометрах роль ЗЭ играют вращающиеся шестеренки, в которых фиксируется ровно десять устойчивых положений. Недостаток всех механических ЗЭ — низкое быстродействие.

Создание электронных ЗЭ, имеющих много устойчивых состояний, затруднено. В то же время можно предложить немало способов для реализации ЗЭ с двумя устойчивыми состояниями: ток в катушке идет или не идет, конденсатор заряжен или разряжен и т. п. Правда, эти состояния не вполне устойчивы: возбужденный в катушке ток в конце концов прекращается, а конденсатор — разряжается. Следовательно, ЗЭ на их основе могут хранить информацию лишь ограниченное время. Этого недостатка лишены, например, ЗЭ из ферритовых колец. Зафиксировав такое колечко в пространстве, его можно намагнитить в направлении по часовой стрелке или наоборот. Любое такое состояние сохраняется неограниченно долго без притока энергии извне.

Итак, если состояния несложных электронных ЗЭ соответствуют двум символам, то для хранения информации целесообразно ее кодирование в двоичной системе счисления. Может показаться, что при таком выборе выигрыш в простоте ЗЭ достигается слишком дорогой ценой. Ведь запись произвольного числа в двоичной системе по сравнению с любой другой будет самой длинной, т. е. для хранения числа потребуется очень много ЗЭ. На самом деле все обстоит не так уж плохо. Различие в требуемом числе ЗЭ чисто количественное, а сложность ЗЭ в недвоичных системах счисления возрастает качественно. Даже если предположить, что сложность ЗЭ прямо пропорциональна основанию системы счисления, то затраты на оборудование для хранения двоичных чисел едва ли не минимальны. В этом смысле теоретически наиболее выгодна троичная система счисления.

С двоичной системой связаны и другие серьезные преимущества. Она обеспечивает максимальную помехоустойчивость в процессе передачи информации как между отдельными узлами автоматического устройства, так и на большие расстояния. В ней предельно просто выполняются арифметические действия, ведь таблицы сложения и умножения занимают всего лишь восемь строк:

$$\begin{array}{ll} 0+0=0 & 0\cdot 0=0 \\ 0+1=1 & 0\cdot 1=0 \\ 1+0=1 & 1\cdot 0=0 \\ 1+1=10 & 1\cdot 1=1 \end{array}$$

Нетрудно себе представить, как упростилось бы обучение арифметике малышей, если бы мы пользовались двоичной систе-

мой счисления. Наконец, в двоичной системе наиболее просто описываются и реализуются логические операции.

Благодаря таким особенностям двоичная система стала стандартом при построении ЭВМ, однако уместно заметить, что она издавна была предметом пристального внимания. Вот что писал выдающийся французский математик Пьер Симон Лаплас (1749—1807) об отношении к двоичной (бинарной) системе уже упоминавшегося великого немецкого математика Г. Ф. Лейбница: «В своей бинарной арифметике Лейбниц видел прообраз творения. Ему представлялось, что единица представляет божественное начало, а нуль — небытие, и что высшее существо создает все сущее из небытия точно таким же образом, как единица и нуль в его системе выражают все числа». Оставим в стороне свойственные тому времени религиозные заблуждения и согласимся с удивительной универсальностью алфавита, состоящего всего из двух символов.

3.5. Бит, байт, слово

Сопоставим ряды натуральных чисел в двоичной и десятичной системе счисления, ограничившись четырьмя разрядами для записи двоичных чисел.

десятичное число	двоичное число	десятичное число	двоичное число
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

Приведенное соответствие можно построить двумя способами: переводя десятичные числа в двоичные или же последовательно прибавляя единицу к предыдущему числу столбца. Нетрудно увидеть следующую закономерность: добавление единицы к двоичному числу всегда изменяет его младшую цифру на противоположную, а каждая последующая цифра изменяется лишь в том случае, когда предыдущая изменила свое значение с 1 на 0. Именно в этом случае происходит перенос в старший разряд и именно эта особенность используется при построении *двоичных счетчиков* (см. гл. 6).

Запись двоичных чисел в примере может показаться несколько странной: мы пишем 0001, 0010, 0100 и т. п. вместо 1, 10, 100, сохраняя «незначащие» нули в старших разрядах. Этим подчеркивается параллельный характер обработки информации в ЭВМ. Человек осуществляет арифметические операции над числами последовательно, цифра за цифрой; когда при этом встречаются нули в старших разрядах, то всегда можно разобраться, «значащие» ли они, и в противном случае закончить вычисления

или их этап. ЭВМ производит операции над числами параллельно сразу во всех разрядах, поэтому во всех разрядах всегда должно быть что-то записано, даже если это «незначащий» нуль.

Уже отмечалось, что совокупность символов, обрабатываемых автоматом параллельно, называется *словом*. Слово, с которым оперирует ЭВМ, даже если это закодированная информация нечисловой природы, всегда можно интерпретировать как двоичное число. Один разряд двоичного числа называется *битом* (сокращение английского словосочетания Binary digIT — двоичная цифра). Большие ЭВМ оперируют словами в 32 бита и более, а для большинства микропроцессоров характерная длина слов — 16 бит, хотя до сих пор распространены и 8-разрядные процессоры, а совсем недавно стали появляться 32-разрядные.

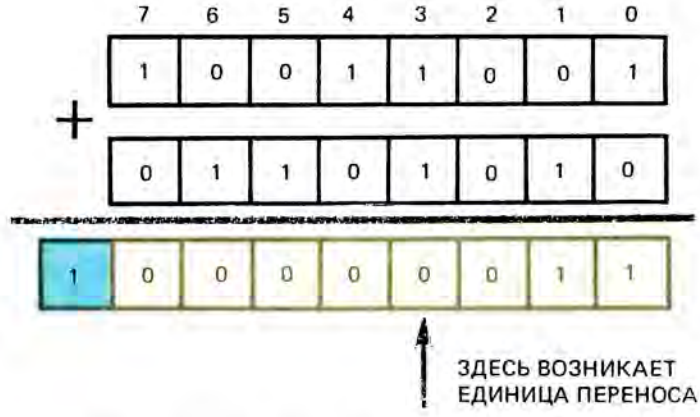
Может показаться, что конечный диапазон представления чисел в ЭВМ ограничивает ее вычислительные возможности. На самом деле короткая длина слова приводит только к снижению быстродействия ЭВМ: обработка достаточно больших чисел ведется последовательно-параллельным способом. При этом сами числа представляются несколькими машинными словами, а для выполнения операций над ними необходимо составить специальные программы.

Для ЭВМ и микропроцессоров характерна возможность производить действия и над укороченными операндами. Это во многом обусловлено тем, что для кодирования каждого символа текстовой информации чаще всего требуется совокупность из 8 бит, называемая *байтом*. Эффективность обработки текстов повышается, если в машинном слове укладывается целое число байтов и обеспечена возможность отдельной работы с каждым из них.

3.6 Особенности машинной арифметики

Представление чисел словами конечной длины накладывает особый отпечаток на характер арифметических действий. В качестве примера рассмотрим выполнение сложения в ЭВМ, оперирующей байтами (рис. 3.4). Возникающая в этом примере единица переноса из старшего разряда не «вписывается» в слово, содержащее результат. Такая ситуация называется *переполнением разрядной сетки* машины. Она может привести к получению неверного результата, поэтому УУ строится таким образом, что в случае переполнения всегда происходит изменение его состояния. Внутреннее состояние обычно отражается т. н. *словом состояния процессора* (ССП). ССП содержит информацию об особых результатах выполнения каждой команды и хранится в специальном регистре УУ. Отдельный его бит, обозначаемый буквой *C* (от англ. *Carry* — перенос), устанавливается в единичное состояние как раз при наличии переноса из старшего разряда.

Обычно при переполнении управление передается участку программы, который использует для представления результата

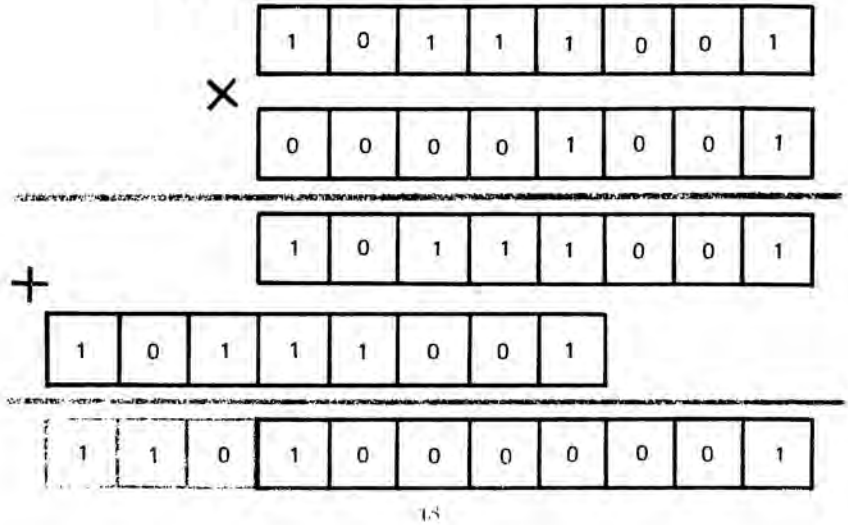


При сложении чисел 153_{10} и 106_{10}, представленных байтами, перенос, возникающий в третьем разряде, «путешествует» влево и оказывается за пределами байта результата. Аналогичная ситуация может возникнуть и при любой другой (конечной) длине слова. Если не принять специальных мер, единица переноса из старшего разряда будет утеряна, что приведет к ошибочному результату.

более, чем одно слово, или сигнализирует о возникшей ситуации человеку.

При умножении (рис. 3.5) результат может занимать несколько дополнительных разрядов. Многие ЭВМ вообще не имеют команды умножения. Оно осуществляется специальными стандартными программами, содержащими команды сложения и сдвига. При

В ходе умножения двоичных представлений чисел 185_{10} и 9_{10} сразу три разряда результата не укладываются в однобайтовом слове.



выполнении каждой из таких команд переполнение может анализироваться по С-биту.

Умножение производится значительно эффективнее, если ЭВМ обладает *командами расширенной арифметики* (РА). Одна из команд РА выполняет операцию умножения двух операндов длиной в машинное слово с размещением результата в двух машинных словах. В этом случае говорят, что результат имеет *двойную точность*. Двух слов всегда хватает для представления произведения одно-словных операндов, поэтому после выполнения такой команды бит переполнения принимает нулевое состояние.

Команда РА для выполнения деления использует два слова для представления делимого и по одному — для делителя, частного и остатка.

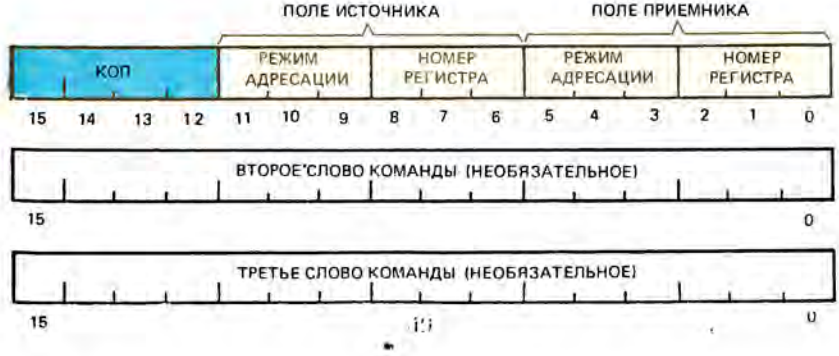
РА и особые команды для работы над т. н. *числами с плавающей запятой* значительно повышают вычислительные возможности ЭВМ.

3.7. Как кодируются команды?

Считывая очередную команду программы из памяти, процессор ЭВМ должен получить исчерпывающие сведения о том, как эту команду выполнять. В их числе — тип операции, число операндов с подробными указаниями на способ их адресации, число слов памяти, занимаемых текущей командой.

Кодирование команд рассмотрим на конкретном примере шестнадцатиразрядных ЭВМ типа СМ-4, ДВК и им подобных. Процессоры этого семейства допускают явную адресацию в одной команде двух операндов. Типичные двухадресные команды — сложение, вычитание, пересылка, сравнение. Каждая такая команда в общем случае занимает три последовательных слова ОЗУ (рис. 3.6).

Рис. 3.6. Первое слово двухадресной команды содержит код операции (КОП) и номера РОН с указаниями на используемый способ адресации для каждого из операндов. Один из них называется операндом-источником. Двоичный номер соответствующего ему РОН помещается в битах 6...8. Вместе с битами 9...11, определяющими один из восьми возможных способов адресации, они образуют т. н. поле источника. Биты 0...5, составляющие поле приемника, выполняют аналогичные функции для второго операнда (приемника).



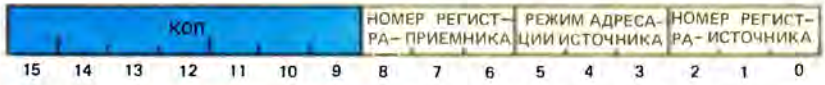


Рис. 3.7. Регистровые команды имеют семибитный КОП. В их числе упоминавшиеся в предыдущем параграфе команды РА, для которых биты 0...5 указывают на РОН и характер адресации источника, а биты 6...8 определяют регистр-приемник. В случае команды деления (КОП-0111001) в регистр-приемник (его номер обязан быть четным) и следующий за ним по счету РОН помещается 32-разрядное делимое, а после выполнения операции частное и остаток. Для некоторых других команд этой группы биты 0...5 исполняют особые функции.

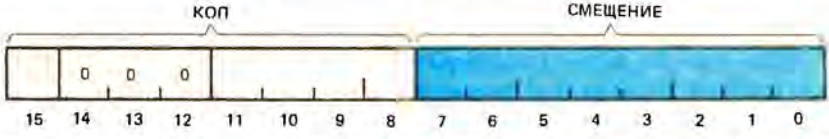


Рис. 3.8. КОП команд перехода по условию занимает весь старший байт. Биты 8...11 кодируют условие перехода в зависимости от значения отдельных битов ССП. При выполнении заданного условия осуществляется передача управления к команде, отстоящей от текущей на число слов, определяемое смещением.

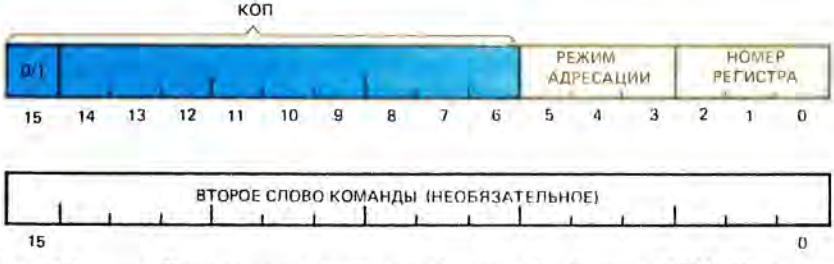


Рис. 3.9. Одноадресные команды обладают 10-битным КОП. Поля «режим» и «номер регистра» определяют характер адресации (аналогично двухоперандным командам).

Названия «источник» и «приемник» для операндов соответствуют их роли при выполнении команды пересылки, которая «копирует» *операнд-источник* в регистр или ячейку ОЗУ, занимаемую *операндом-приемником*. Операнд-источник сохраняется неизменным и при выполнении большинства других команд, результат которых располагается на месте операнда-приемника.

Необязательные второе и третье слова двухадресных команд могут нести дополнительную информацию об операндах при некоторых способах адресации.

Команды кодируются сложнее, чем числа. Отдельные биты чисел используются ЭВМ совершенно одинаково. Напротив, группы битов *(поля)* и даже отдельные биты в записи команд могут играть совершенно различную роль. Так, бит 15 в КОП для боль-

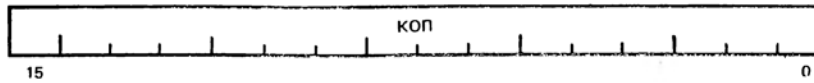


Рис. 3.10. Команды управления состоят только из КОП. Одна из таких команд с КОП=000...00 приводит к прекращению выполнения программы, т. е. к остановке процессора

шинства команд играет роль указателя длины операндов: если он равен нулю, команда оперирует со словами, в противном случае — с байтами. Оставшиеся три бита интерпретируются как двоичный номер, указывающий на конкретную операцию. Поскольку три бита дают $2^3 = 8$ различных двоичных чисел, то имеется возможность кодирования шестнадцати различных команд (по восемь), оперирующих с байтами и словами.

Может сложиться впечатление, что рассматриваемые процессоры рассчитаны на выполнение только шестнадцати команд. В действительности их число во много раз больше благодаря механизму т. н. *кодов расширения*. Дело в том, что необходимое на практике число двухадресных команд меньше шестнадцати. Поэтому некоторые из шестнадцати возможных комбинаций четырех старших битов — коды расширения — указывают процессору на то, что соответствующие команды не двухадресные и имеют совершенно иной формат.

В числе таких кодов — 0000, 1000, 0111, 1111. Некоторые из этих кодов приводят к группе т. н. регистровых команд (рис. 3.7.). Аналогично получается группа команд условных переходов, формат которых приведен на рисунке 3.8.

Некоторые из кодов расширения команд перехода приводят к группе одноадресных команд, формат которых представлен на рисунке 3.9. В нее входит, например, *команда очистки* содержимого адресуемой ячейки памяти, т. е. установки всех ее битов в нулевое состояние.

Наконец, коды расширения одноопернадных команд приводят к группе команд, использующих для записи КОП все биты командного слова (рис. 3.10). Такие команды не имеют операндов, они используются для управления работой процессора, в частности для изменения состояния ССП.

3.8. Зачем нужны иные системы кодирования?

Десятичная система счисления малоприемлема для ЭВМ, а двоичное кодирование неудобно для человека, занимающегося созданием программ. Усилия специалистов по вычислительной технике привели к тому, что большинство программ для современных ЭВМ пишутся не в двоичной и даже не в десятичной системе счисления, а на *языках высокого уровня*, значительно более удобных для человека. Ведутся работы в направлении обеспечения диалога с ЭВМ на естественном, человеческом языке, т. е. работы по автоматическому переводу информации, курсирующей между

человеком и ЭВМ в процессе их диалога. Значительную роль в обеспечении такого перевода играют сами ЭВМ как универсальные устройства обработки информации.

Тем не менее знание особенностей двоичного представления чисел и команд необходимо не только разработчикам и обслуживающему персоналу ЭВМ, но и программистам. Понимание тонкостей *машинного языка* позволяет получить более эффективные программы, даже если они пишутся на языке высокого уровня.

А нельзя ли найти удобную для человека форму восприятия двоичной информации? На первый взгляд кажется, что для этой цели достаточно снабдить ЭВМ устройством для преобразования двоичных чисел в десятичные, и наоборот. Однако при кодировании команд отдельные биты могут играть определенную, индивидуальную роль. В процессе перевода кода команды в десятичное число значения этих битов становятся трудноосязаемыми, поэтому запись десятичного кода нужной команды сразу, минуя ее двоичное представление, становится весьма трудоемкой. Напротив, чтобы выяснить смысл команды, представленной десятичным числом, придется обязательно перевести его в двоичную систему счисления.

В определенной мере указанных недостатков лишены *двоично-десятичные системы кодирования*. В них каждой десятичной цифре ставится в соответствие двоичный код, состоящий чаще всего из четырех символов. В двоично-десятичном коде десятичного числа вместо каждой десятичной цифры последовательно записываются соответствующие им четверки битов. Для обратного перевода двоичная запись разбивается на группы по четыре бита, и каждая группа заменяется соответствующей десятичной цифрой. Отдельный бит двоично-десятичного кода влияет при этом только на единственный знак десятичного эквивалента.

Однако двоично-десятичному кодированию свойственна избыточность. Оно требует больше места в памяти ЭВМ, поэтому применение двоично-десятичных кодов ограничивается устройствами ввода-вывода данных.

3.9. Эсперанто для человека и ЭВМ

Если произвольное двоичное число разбить на группы по n битов, то каждой такой группе можно поставить в соответствие цифру системы счисления с основанием $b = 2^n$. В частности, любой тройке битов будет соответствовать единственная *восьмеричная* ($8 = 2^3$) цифра, и, наоборот, произвольной восьмеричной цифре можно поставить в соответствие единственную тройку битов (триаду). Этим определяется простая взаимосвязь между записями чисел в двоичной и восьмеричной системах счисления.

С целью перевода в восьмеричную систему двоичное число разбивается на триады (справа налево). Если при этом в крайней слева группе окажется менее трех двоичных цифр, недостающие позиции заполняются нулями. Ниже приведены таблица соответ-

ствия двоичных триад восьмеричным цифрам и примеры перевода из двоичной системы в восьмеричную, и наоборот.

Двоичная триада	Восьмеричный эквивалент
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

$110\ 010\ 111\ 101_2 = 6275_8$;
$1304_8 = 001\ 011\ 000\ 100_2$;
$1\ 011\ 010\ 011\ 110\ 100_2 = 132364_8$;
$0\ 111\ 001\ 101\ 100\ 111_2 = 071547_8$.

Два последних примера характерны для ЭВМ, оперирующих шестнадцатиразрядными словами. Восьмеричные записи таких слов всегда содержат шесть цифр, старшая из которых может принимать лишь значения 0 и 1.

Совершенно аналогично, на основе таблицы соответствия четверок битов (тетрад) и *шестнадцатеричных* цифр, устанавливается взаимосвязь между любыми двоичными словами и их шестнадцатеричными эквивалентами:

Двоичная тетрада	Шестнадцатеричный эквивалент
. . .	...
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	*A*
1 0 1 1	*B*
1 1 0 0	*C*
1 1 0 1	*D*
1 1 1 0	*E*
1 1 1 1	*F*

$1101\ 1110\ 0101\ 1010_2 = \mathrm{DE5A}_{16}$;
$1101111100_2 = \mathrm{37C}_{16}$;
$\mathrm{3A}_{16} = 0011\ 1010_2 = 111010_2$.

Запись двоичных чисел в любой из этих форм компактнее, проще воспринимается, а значит, достаточно удобна для человека. Взаимосвязь отдельных битов двоичного числа и символов используемой формы локализована: значение отдельного бита влияет на единственный символ, и наоборот, каждый символ связан только с небольшой группой из трех или четырех битов. Наконец, алгоритмы перевода двоичного числа в одну из рассмотренных форм (и наоборот) очень просты и для человека, и для ЭВМ.

Восьмеричная или шестнадцатеричная системы счисления могут выступать в качестве простейшего языка общения человека с ЭВМ, достаточно близкого как к привычной для человека десятичной системе счисления, так и к двоичному «языку» машины. Запись чисел в системах счисления с основанием $b = 2^n$ можно рассматри-

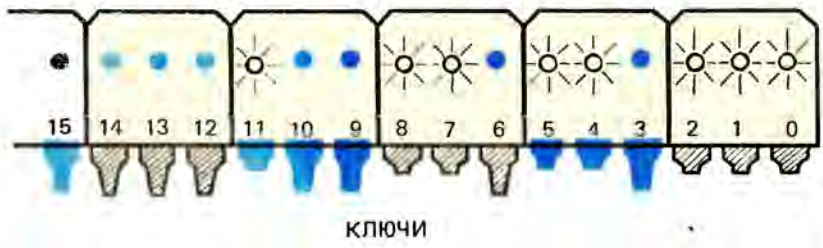


Рис. 3.11 Пульт шестнадцатиразрядной ЭВМ использует двоичную систему: 0 — ключ (тумблер) опущен, лампочка не горит, 1 — ключ поднят, лампочка горит. Группировка этих элементов в триады (ключи обычно имеют чередующуюся от группы к группе окраску) существенно упрощает как восприятие двоичной информации в восьмеричной форме, так и ввод восьмеричных чисел в двоичном виде.

вать как своеобразные формы представления двоичной информации, а при $n = 3, 4$ они близки к соответствующим десятичным числам с точки зрения компактности.

По этой причине использование восьмеричных и шестнадцатеричных кодов машинных команд, адресов ОЗУ, операндов широко распространено как на этапах составления несложных программ для микро-ЭВМ, их отладки, ручного ввода-вывода данных, так и на этапах разработки, создания, настройки и обслуживания вычислительных систем. Не случайно пульт управления большинства ЭВМ предполагает ввод, вывод (или индикацию) машинных слов именно в одной из таких систем счисления, чаще в восьмеричной (рис. 3.11).

На этом рисунке не показаны вспомогательные органы, которые индицируют состояние ЭВМ и ее узлов, инициируют ввод в машину адреса ячейки ОЗУ или ее содержимого и т. п.

3.10. Представление отрицательных чисел дополнительными кодами

Одной из причин появления в математике отрицательных чисел была потребность сделать всегда выполнимой операцию вычитания. Выражение $2-5=$, бессмысленное на множестве натуральных чисел, принимает вполне определенное значение, равное -3, на множестве целых чисел. Сама запись отрицательных, как и любых других чисел, по существу представляет собой некоторый код, расшифровываемый по особым правилам. Так,— 2 есть число, равное 2 по величине (модулю), но противоположное ему по знаку.

Операции над отрицательными числами определены так, чтобы они подчинялись тем же законам, что и натуральные числа. Как следствие из сказанного, для любого числа $y+(-y)=0$, а опе-

рацию вычитания двух чисел всегда можно свести к сложению согласно известной формуле $x - y = x + (-y)$. К сожалению, в конкретных вычислениях не обойтись без операции вычитания: несмотря на то что $2 - 5 = 2 + (-5)$, мы находим разность $5 - 2 = 3$ и присваиваем ей знак минус, потому что в исходном выражении из меньшего (по модулю) числа вычиталось большее.

Однако свести операцию вычитания к сложению все-таки можно благодаря явлению переполнения. Это реализуется при помощи специального способа кодирования отрицательных чисел — представления их в так называемом *дополнительном коде.*

Сущность способа нетрудно понять, изучая работу *реверсивного счетчика.* Подходящий счетчик имеется, например, в большинстве магнитофонов и служит в качестве индикатора расхода ленты. Его показания увеличиваются либо уменьшаются в зависимости от направления движения ленты. Интересующая нас особенность состоит в том, что при возрастании показаний счетчика до максимального, например до 999, следующими его состояниями будут 000, 001 и т. д. Происходит то, что мы называем переполнением разрядной сетки: $999 + 1 = 1000$, но для изображения единицы переноса в нашем счетчике не хватает одного разряда. При обратном движении ленты показания счетчика убывают, а после состояния 000 следует 999, 998, ... Отвлекаясь от конкретного устройства счетчика, логику изменения его состояний можно представить, как показано на рис. 3.12.

Но ведь последовательное вычитание единицы из некоторого числа, после достижения нуля, должно давать отрицательные числа: -1, -2, ... Нельзя ли связать каким-либо образом состояние 999 с числом -1? Может быть следует рассматривать 999 как искомый код числа -1?

Проверим нашу гипотезу: $001 + 999 = 1000$, а с учетом того, что единица переполнения теряется, получаем 000. Проверяя таким же образом соотношение $y + (-y) = 0$ для других y, можно убедиться, что 998, 997, ... так же подходят в качестве кодов отрицательных чисел -2, -3, ...

Попробуем теперь сложить положительное число, допустим 5 с кодом отрицательного числа -3, т. е. с 997. Получим $5 + 997 = 1002$, а с учетом потери единицы переполнения — число 2, т. е. правильный результат, который в обычной арифметике получается вычитанием: $5 - 3 = 2$.

Для того чтобы сформулировать точное определение дополнительного кода, осталось выяснить, какие коды в кольце, представленном на рисунке 3.12, соответствуют положительным, а какие отрицательным числам, иначе возникает неоднозначность в их интерпретации. Естественно следующее решение: считать половину состояний (от 0 до 499) — кодами нуля и положительных чисел, а оставшуюся половину (500...999) — кодами отрицательных чисел.

Таким образом, дополнительный код положительного числа совпадает с этим числом, а для отрицательного числа он равен дополнению его величины до числа $x_{\text{пер}}$, возникающего при пере-

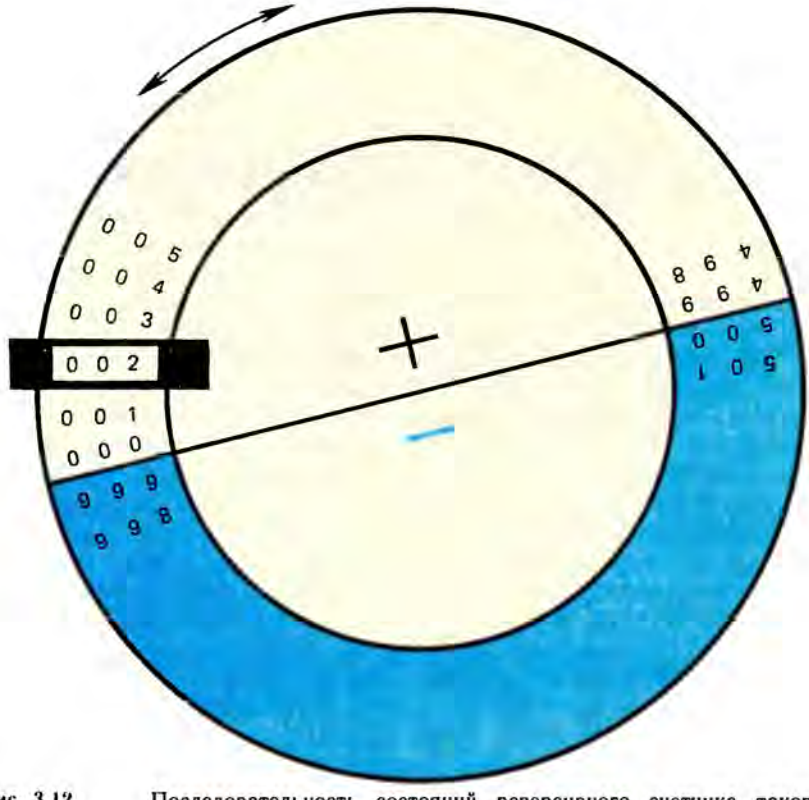


Рис. 3.12. Последовательность состояний реверсивного счетчика такова, что ее можно представить в виде замкнутого кольца из чисел. Если начальные показания счетчика 002, то в режиме вычитания следующими состояниями будут 001, 000, а затем 999, 998, ..., которые можно интерпретировать как коды отрицательных чисел -1, -2, ...

полнении используемой разрядной сетки. Очевидно, $x_{\text{пер}} = b^n$, где b — основание системы счисления, n — число разрядов. Дополнения приходится считать и при декодировании отрицательных чисел, коды которых лежат в пределах от $\frac{x_{\text{пер}}}{2}$ до $x_{\text{пер}} - 1$.

Находить дополнение числа x до $x_{\text{пер}}$, т. е. разность $b^n - x$, можно и другим способом. Обозначим через $\bar{x}$ число, полученное заменой каждой цифры числа x на ее дополнение до цифры $b - 1$. Очевидно, $\bar{x} + x = b^n - 1$. Так, $123_{10} + \overline{123}_{10} = 123 + 876 = 999$. Следовательно, $b^n - x = \bar{x} + 1$.

Такой способ особенно удобен в двоичной системе счисления, т. к. число x, называемое в этом случае *инверсией* x, получается в результате простой замены в числе x единиц нулями и наоборот. Следовательно, дополнения двоичных чисел можно находить, минуя операцию вычитания.

На практике важна еще одна особенность дополнительных ко-

дов двоичных чисел. Заметим, что число $\frac{x_{\text{пер}}}{2}$ состоит из единицы в старшем бите и нулей в последующих, поэтому все дополнительные коды отрицательных чисел имеют в старшем бите единицу, а положительных чисел — нуль. Говоря другими словами, старший бит дополнительного кода является признаком знака закодированного числа *(знаковым битом)*.

Итак, представление целых чисел дополнительным кодом позволяет обойтись без операции вычитания. Действительно, для любых чисел $x - y = x + (-y)$ и при условии, что x и $-y$ представлены дополнительным кодом, их сложение приведет к правильному результату. Сам переход к дополнительным кодам также не нуждается в операции вычитания, а может быть сведен к более простым операциям инверсии и добавления единицы *(инкрементации)*. По этой причине набор команд ЭВМ, состоящий из сложения, побитной инверсии и инкрементации, достаточен для выполнения всех арифметических действий, ведь умножение и деление могут быть сведены к сложению и вычитанию, что вытекает из известных всем алгоритмов выполнения этих действий.

Диапазон представления чисел n-разрядными двоичными кодами изменяется от $(0...2^n - 1)$ для кодирования неотрицательных чисел до $(-2^{n-1}...2^{n-1} - 1)$ для представления целых чисел дополнительным кодом.

3.11. Особенности арифметики в дополнительном коде

Не следует забывать, что явление переполнения разрядной сетки машины, открывая возможность удобного представления чисел в дополнительном коде, все же является и основным источником неприятностей в машинной арифметике. Ограниченная длина слова ЭВМ приводит к необходимости следить за тем, чтобы в разрядной сетке машины укладывались не только операнды, но и результат выполняемого действия. Как это делается в случае представления положительных чисел натуральным двоичным кодом, достаточно подробно разбиралось в 3.6. Нетрудно предположить, что с переходом к представлению чисел дополнительным кодом трудности могут только возрасти, хотя бы из-за уменьшения максимальной величины допустимых чисел. Для шестнадцатиразрядных ЭВМ диапазон представления чисел в натуральном коде — от 0 до 65535, а в дополнительном — от -32768 до $+32767$.

Для того чтобы выяснить, в каких случаях операции с числами в дополнительном коде могут привести к неверным результатам, рассмотрим несколько простых примеров:

$+\begin{array}{r}0011\\0010\end{array}$	$\begin{array}{r}0101\end{array}$	$+\begin{array}{r}3\\2\end{array}$	$\begin{array}{r}5\end{array}$	$+\begin{array}{r}0110\\1111\end{array}$	$\begin{array}{r}10101\end{array}$	$+\begin{array}{r}6\\-1\end{array}$	$\begin{array}{r}5\end{array}$
$+\begin{array}{r}1010\\1111\end{array}$	$\begin{array}{r}11001\end{array}$	$+\begin{array}{r}-6\\-1\end{array}$	$\begin{array}{r}-7\end{array}$				

Нет переносов в знаковый бит и из знакового бита. Результат верный.

Есть переносы в знаковый бит и из знакового бита. Результат верный.

Есть переносы в знаковый бит и из знакового бита. Результат верный.

0100	4	1010	−6	1100	−4
+0110	+6	+0100	+4	+1010	+−6
1010	−6	1110	−2	10110	6

Есть перенос в знаковый бит. Результат ошибочный. Правильный результат должен быть 10_{10}, но для его представления в дополнительном коде не хватает разрядов.

Переносов нет. Результат верный.

Есть перенос из знакового бита. Результат ошибочный (должно быть -10_{10}).

Анализ этих примеров позволяет сделать предположение, что признаком ошибочных ситуаций может служить наличие или отсутствие переносов в знаковый бит и из знакового бита в процессе сложения: результат ошибочен, если из двух этих переносов произошел только один. В остальных случаях, т. е. когда переносы вообще отсутствуют, или произошли сразу оба, сложение даст верный результат, т. е. представление чисел дополнительным кодом замкнуто относительно операции сложения. (Строгое доказательство этого утверждения может быть построено с учетом того, что запись двоичного числа в дополнительном коде можно интерпретировать как особый взвешенный код. Весовые коэффициенты этого кода для всех битов, кроме знакового, такие же, как у натурального двоичного кода, а знаковый бит имеет отрицательный вес, равный -2^{n-1}, где n — длина слова.)

Неблагоприятные ситуации, возникающие при вычислениях с числами в дополнительном коде, должны учитываться программистами. Обычно для этого в состав ССП ЭВМ вводят т. н. V-бит (от англ. *oVerflow* — переполнение), который автоматически устанавливается в единичное состояние, когда при выполнении текущей команды происходит перенос в старший (знаковый) бит или из него. Состояние этого бита может учитываться при выполнении команд условного перехода. Используя эти команды, можно избежать ошибок таким же образом, как это было рассмотрено в 3.6.

Отметим, что в состав ССП обычно вводят и другие биты, упрощающие работу с числами в дополнительном коде. Так, биты N и Z (от англ. *Negative* и *Zero*) устанавливаются, если результат предыдущей операции соответственно отрицательный или нулевой. Состояния этих битов также могут анализироваться командами перехода.

3.12. **Некоторые другие представления отрицательных чисел**

Представление отрицательных чисел в дополнительном коде наиболее распространено. Тем не менее, существуют ситуации, когда предпочтительнее использование других форм. В первую очередь это относится к представлению дробей, о чем пойдет речь в следующем параграфе. Здесь же мы кратко обсудим еще две распространенные формы записи отрицательных чисел.

1. Представление прямым кодом (в форме «знак-величина»). Оно соответствует привычной человеку форме записи чисел. Как и для дополнительного кода, первый бит n-разрядного слова знаковый. Остальные $n-1$ битов представляют величину (модуль) числа в двоичной системе счисления. Для выполнения арифметических операций с такими кодами нужны специальные программы, реализующие хорошо известные каждому человеку алгоритмы. Так, для операции сложения нужно проверить знаки обоих операндов. Если они одинаковы, то вычисляется сумма операндов и ей присваивается тот же знак. Если знаки противоположны, то из большего по модулю числа нужно вычесть меньшее и результату присвоить знак уменьшаемого. Тем не менее соответствующие этим алгоритмам программы не так уж просты. Знаковый бит должен анализироваться ими отдельно, а сами арифметические действия производиться только над остальными битами чисел. Это требует применения специальных команд. В сложении дополнительных кодов знаковый бит фигурирует наравне со всеми остальными битами, что приводит к более эффективной работе ЭВМ.

В системе прямых кодов есть интересная особенность, заключающаяся в существовании двух различных представлений нуля: 00...0 и 10...0, называемых *положительным и отрицательным нулем*. Нетрудно убедиться, что оба приведенных представления равноправны. Однако эта особенность является источником ошибок при составлении программ. Так, если приходится проверять результат каких-то вычислений на равенство нулю, то сравнение должно проводиться как с положительным, так и с отрицательным нулем, о чем легко забыть.

Важное достоинство представления целых чисел в форме «знак-величина» — простота операций кодирования и декодирования. Диапазон чисел, представленных прямым кодом, лежит в пределах от $-2^{n-1}+1$ до $2^{n-1}-1$. Для шестнадцатиразрядных ЭВМ эти границы равны -32767, $+32767$.

2. Представление *смещенным кодом* (с избытком 2^{n-1}). Диапазон представления чисел в этой системе $-2^{n-1} \leqslant x \leqslant 2^{n-1}-1$, т. е. такой же, как и в системе дополнительных кодов. Алгоритм кодирования основан на том, что число $x+2^{n-1}$, лежащее в пределах $0 \leqslant x+2^{n-1} \leqslant 2^n-1$, может быть представлено натуральным n-разрядным двоичным кодом. Число 2^{n-1} называется смещением. Для декодирования можно вычесть смещение из кода по правилам обычной арифметики. Интересно, что такая система идентична системе дополнительных кодов с точностью до инверсии старшего бита: в системе со смещением 0 означает отрицательное число, а 1 — положительное. На основании этой связи легко получить правила определения переполнения.

В таблице 3.1 приведена интерпретация четырехразрядных двоичных кодов в различных системах представления целых чисел.

Таблица 3.1

Двоичный код	Числа в различных представлениях			
	Натуральные числа	Дополнительное кодирование	Прямое кодирование	Кодирование со смещением
0 0 0 0	0	0	+0	−8
0 0 0 1	1	1	1	−7
0 0 1 0	2	2	2	−6
0 0 1 1	3	3	3	−5
0 1 0 0	4	4	4	−4
0 1 0 1	5	5	5	−3
0 1 1 0	6	6	6	−2
0 1 1 1	7	7	7	−1
1 0 0 0	8	−8	−0	0
1 0 0 1	9	−7	−1	1
1 0 1 0	10	−6	−2	2
1 0 1 1	11	−5	−3	3
1 1 0 0	12	−4	−4	4
1 1 0 1	13	−3	−5	5
1 1 1 0	14	−2	−6	6
1 1 1 1	15	−1	−7	7

3.13. Представление дробей. Числа с фиксированной и плавающей запятой

Нетрудно убедиться, что без каких-либо изменений в алгоритмах выполнения арифметических операций рассмотренные методы кодирования целых чисел пригодны для обозначения дробей. Так, если договориться, что в четырехразрядных дополнительных кодах двоичная запятая, отделяющая целую часть от дробной, расположена посередине, то целочисленное действие $5 + (-1) = 4$ можно интерпретировать как $1{,}25 + (-0{,}25) = 1{,}0$:

$$\begin{array}{r} 0101 \\ +\,1111 \\ \hline 10100 \end{array} \qquad \begin{array}{r} 5 \\ +\,-1 \\ \hline 4 \end{array} \qquad \begin{array}{r} 01{,}01 \\ +\,11{,}11 \\ \hline 101{,}00 \end{array} \qquad \begin{array}{r} 1{,}25 \\ +\,-0{,}25 \\ \hline 1{,}00 \end{array}$$

Таким образом, для представления дробей программист может мысленно располагать двоичную запятую в любой нужной ему позиции. Естественно, при выполнении сложения и вычитания двоичные запятые должны быть расположены одинаково в обоих операндах, таково же будет и положение запятой в результате. Представление дробей, при котором положение двоичной запятой задается неявно в определенном месте машинного слова, называется *представлением с фиксированной запятой*.

Неудобство такого представления проявляется при решении задач с величинами, которые могут сильно изменяться в сторону как очень малых, так и очень больших чисел. В конкретных физических, математических и других задачах диапазон изменения величин может составлять, например, от 10^{-30} до 10^{+30}. Можно убедиться, что в представлении с фиксированной запятой подошли

бы двоичные операнды длиной около 256 бит (32 байта), по 128 бит на целую и дробную части.

Однако работа ЭВМ с операндами такой длины была бы крайне неэффективной. На практике значительная часть из этих 256 бит оказалась бы равной нулю. Для очень маленьких чисел это очевидно. При работе же с очень большими числами их младшие разряды, а тем более дробная часть, обычно игнорируются. Ведь высокая точность результата любых вычислений достигается лишь тогда, когда все участвующие в них числа одинаково точны. Задание же всех величин с точностью до 256 бит (или примерно 76 десятичных цифр) — дело не только нереальное, но и бессмысленное, хотя бы потому, что многие из этих величин получаются в результате измерений не очень точными приборами. Не случайно в практических расчетах редко используют более трех значащих цифр, соответствующим образом округляя промежуточные результаты.

Наконец, устройство ЭВМ с 256-разрядным словом оказалось бы очень сложным, а в случае последовательно-параллельной обработки, например 16-разрядными словами, резко возросло бы время вычислений.

Выход из затруднения состоит в отказе от фиксированного расположения запятой. Действительно, 256-разрядные двоичные числа с запятой посередине можно записывать 128-разрядным словом, в котором двоичной запятой разрешено находиться в любой позиции («плавать»). Если двоичная запятая находится после крайнего справа бита, то диапазон представления положительных чисел будет от 0 до $2^{128}-1$. Если же она расположена перед старшим разрядом, то возможно изображение правильных положительных дробей от 2^{-128} до $1-2^{-128}$. Промежуточные положения запятой дают смешанные дроби, состоящие из целой и дробной части. Заметим, что двойная экономия битов в слове достигается ценой некоторой потери в точности представления именно таких дробей, ведь в их прежней записи значащими могли быть все 256 бит.

Реальный выигрыш в числе битов хотя и велик, но все же меньше, чем могло бы показаться. Ведь если разрешить запятой «плавать» во всех числах, то для выполнения операций над ними в записи каждого числа должна быть информация о месте расположения запятой. В тех случаях, когда эта информация выражена в записи чисел явным образом, говорят о *представлении чисел с плавающей запятой.*

Наиболее удобным образом информация о положении запятой задается, если числа представлены в т. н. *экспоненциальной форме записи.* Это значит, что число записывается в виде произведения дроби со знаком *(мантиссы)* и основания системы счисления, возведенного в степень с некоторым показателем *(порядком),* например $3{,}143 \cdot 10^{-7}$, или $-10{,}11_2 \cdot 2^{101_2}$. Для удобства мантиссу обычно записывают в виде правильной дроби, у которой первая же цифра после запятой — значащая. Так, $3{,}143 \cdot 10^{-7} =$

$= 0{,}3143 \cdot 10^{-6}$, а $-10{,}11_2 \cdot 2^{101_2} = -0{,}1011_2 \cdot 2^{111_2}$. Такая форма экспоненциальной записи называется *нормализованной*. В результате изображение двоичного числа с плавающей запятой состоит из следующих компонентов: знаковый бит мантиссы, мантисса, знак порядка, порядок. Положение двоичной запятой в мантиссе неявно фиксируется слева от ее первой цифры.

Здесь у читателя может возникнуть вопрос, какой же смысл говорить о плавающей запятой, если в мантиссе она фиксирована? Дело в том, что нормализованная мантисса фактически определяет только значащие цифры числа. Истинное положение запятой в числе после его перевода в обычную, неэкспоненциальную форму явно задает порядок. И хотя при этом для данной мантиссы и данного порядка положение запятой получается вполне определенным, то для той же мантиссы, но с другим порядком, запятая займет иное положение.

3.14. Как представляются числа с плавающей запятой в ЭВМ?

В реальных ЭВМ обычно хранятся два-три десятка старших битов, поэтому для хранения всего числа (вместе с порядком) вполне хватает одного-двух машинных слов. Так, формат чисел с плавающей запятой для ЭВМ типа ДВК и аналогичных имеет вид, показанный на рисунке 3.13, причем мантисса представляется в прямом коде, а порядок — в смещенном. Причины такого выбора комментируются в следующем параграфе.

Очевидно, что в нормализованном представлении двоичных чисел первая цифра мантиссы всегда равна 1, поэтому нет необходимости ее хранить. Учитывая этот «скрытый» бит, фактическая длина мантиссы составляет не 23, как следует из рисунка 3.13, а 24 бита. Как исключение, скрытый бит полагается равным нулю, если исходное число равно нулю. В этом случае равны нулю все остальные биты мантиссы и кода порядка (что дает минимальный порядок, равный — 128). Таким образом, представление нуля с плавающей запятой эквивалентно двум словам с нулями в целочисленной арифметике, что оказывается удобным в ряде случаев. Следует отметить, что в ЭВМ автоматически приравниваются к нулю все результаты, имеющие порядок — 128, независимо от значения

Рис. 3.13. В шестнадцатиразрядных ЭВМ числа с плавающей запятой представляются в виде двух машинных слов.

а $3/8 = 1/4 + 1/8 = 0,0110000\ldots_2$

б $0{,}011_2 = 0{,}11_2 \cdot 2^{-1}$

в $-1 \Rightarrow 01111111$ (СМЕЩЕНИЕ 2^7)

Рис. 3.14. Этапы перевода десятичной дроби $^3/_8$ в формат с плавающей запятой: десятичная дробь записывается в двоичном виде *(а)*; полученный результат нормализуется путем сдвига запятой на нужное число позиций, а число сдвигов дает порядок *(б)*; порядок записывается в коде со смещением *(в)*. На последнем этапе *(г)* удаляется первый бит мантиссы и выписывается полное представление дроби в виде двух машинных слов.

мантисс. Правила перевода десятичных дробей в формат с плавающей запятой иллюстрируются рисунком 3.14.

Описанный формат дает числа в диапазоне примерно от 10^{-38} до 10^{39} с точностью около семи десятичных разрядов. Когда такой точности не хватает, используется формат удвоенной точности, отличающийся тем, что для записи мантиссы отводится два дополнительных машинных слова (рис. 3.15). Это дает точность около пятнадцати десятичных разрядов в том же диапазоне порядков.

Вместо термина «плавающая запятая» часто используется *«плавающая точка»*. Это означает одно и то же. Дело в том, что в записи чисел на языках программирования высокого уровня вместо десятичной (двоичной) запятой используется символ точки. Удобство такого подхода очевидно всем, кто сталкивался с необходимостью записи нескольких подряд десятичных дробей.

Рис. 3.15. Числа с плавающей запятой удвоенной точности записываются четырьмя машинными словами.

3.15. Арифметика чисел с плавающей запятой и ее особенности

Выполнение арифметических операций над числами с плавающей запятой в разных ЭВМ производится по-разному. Процессоры простых микро-ЭВМ не имеют команд, позволяющих непосредственно оперировать числами в этом формате. С этой целью приходится составлять специальные программы, использующие команды целочисленных арифметических действий. Системы команд сложных ЭВМ включают *команды арифметики с плавающей запятой,* которые благодаря аппаратной реализации выполняются значительно быстрее. Как компромиссный вариант начальная конфигурация микро-ЭВМ, рассчитанная на целочисленные действия, может пополняться приобретаемым отдельно *процессором плавающей запятой.*

В любом случае алгоритмы арифметических действий над числами в плавающем формате одинаковы, разница состоит в их программной или аппаратной реализации. Так, для выполнения умножения нужно перемножить мантиссы чисел и сложить их порядки. Умножение 24-разрядных мантисс проще всего выполнить в виде 24-кратного цикла из операций сдвига и сложения (см. 3.6) с определением знака результата по хорошо известному правилу знаков (для умножения и деления). Именно по этой причине мантиссу удобнее представлять не в дополнительном коде, а в представлении «знак-величина» (см. 3.12).

Так как нормализованные мантиссы операндов находятся в диапазоне от 0,5 до 1, то результат их умножения лежит в границах от 0,25 до 1 и может оказаться *денормализованным.* Для его нормализации можно сдвинуть мантиссу на один разряд влево и одновременно уменьшить на единицу порядок результата.

В отличие от умножения и деления при выполнении сложения и вычитания основную сложность составляют не собственно эти действия, а процедуры *выравнивания.* Как и в «ручных» операциях над числами в экспоненциальном представлении в первую очередь необходимо произвести сравнение порядков. Если они различны, то мантисса операнда с меньшим порядком сдвигается вправо, а порядок, соответственно, увеличивается. Выравнивание производится до тех пор, пока порядки обоих операндов не сравняются. После этого производится сложение или вычитание мантисс. Наконец, как и для умножения, может понадобиться нормализация результата. В процедурах сравнения порядков представление дополнительным кодом не очень удобно, поэтому-то порядок чисел с плавающей запятой обычно записывается в коде со смещением, упрощающем алгоритмы выравнивания.

3.16. Погрешности машинной арифметики

Как и для целочисленной арифметики, в операциях с плавающей запятой возможно негативное влияние переполнения. Так, двоичная запятая результата умножения может оказаться правее младшего бита воображаемого 128-битового представления слова с плавающей запятой (см. 3.13). Эта ситуация почти аналогична рассмотренному в 3.6 переполнению для целочисленных операций. Отличие состоит в том, что переполнение в операциях с плавающей точкой обнаруживается не по изменению состояния ССП, а по недопустимо большому значению порядка. Операции над нормализованными мантиссами в принципе не могут вызвать переполнения с плавающей точкой, даже если установится бит переполнения ССП. Так, при сложении двух положительных мантисс может возникнуть перенос в знаковый разряд, который на первый взгляд искажает результат (он кажется отрицательным). На самом деле все становится на свои места в результате нормализации.

Результат вычислений с плавающей запятой может не вписаться в разрядную сетку ЭВМ и по другой причине: когда запятая результата находится левее крайней слева цифры условного 128-битового представления. Это происходит, когда порядок результата — недопустимо большое отрицательное число. Такое «переполнение наоборот» называется *исчезновением порядка:* результат становится слишком малым для его представления в стандартном, нормализованном виде. Естественная реакция ЭВМ в этих ситуациях состоит в интерпретации результата как «чистого нуля».

Очевидно, исчезновение порядка приводит к погрешностям в вычислениях. Неточности в операциях с плавающей запятой могут возникнуть и по другим причинам. В их основе — использование укороченных форм записи чисел: так из 128 бит условного слова с плавающей запятой в мантиссе остается только 24 разряда. Поэтому при вычитании двух почти равных чисел с одинаковыми порядками и мантиссами, отличающимися, скажем, в n младших битах, $24 - n$ старших бита результата окажутся равными нулю. После нормализации такого результата лишь n битов могут оказаться значащими. Вообще, каждая процедура выравнивания приводит к потере точности: так, при сдвиге вправо теряются значащие младшие биты. Еще один источник ошибок кроется в невозможности представить конечным машинным словом бесконечные периодические дроби.

Все перечисленные обстоятельства приводят к тому, что при выполнении громоздких вычислений, каждое из которых опирается на предыдущий результат, ошибки накапливаются и могут достигнуть значительной величины. В некоторой мере эти ошибки удается уменьшить за счет округления промежуточных результатов. Так, при сдвиге мантиссы вправо можно не просто отбрасывать младшие биты, а корректировать с учетом их значения оставшийся результат. Это можно сделать, например, путем прибавления еди-

ницы к результату, если старший из отбрасываемых битов равен единице.

Однако в процедуре округления существуют свои «ловушки» и иногда точность результата с применением округления может оказаться ниже. По этой причине существует множество алгоритмов округления. К сожалению, алгоритмы, приемлемые в одних ситуациях, плохо работают в других. Поэтому при решении задач на ЭВМ не следует удивляться иногда получаемым результатам типа $2 \cdot 2 = 3{,}999999$. Зная особенности используемых алгоритмов округления, можно либо устранить подобные аномалии, либо просто привыкнуть к ним.

3.17. ЭВМ и азбука Морзе (кодирование алфавитно-цифровых символов)

Попробуем выяснить, что происходит, когда на дисплее, связанном с ЭВМ, появляется изображение очередной буквы или цифры. Думается, что основные особенности этого процесса читателю уже понятны. В памяти машины должны храниться двоичные коды выводимого сообщения. Код текущего символа передается в дисплей, где специальными схемами обеспечивается его прием, распознавание и вывод изображения соответствующего символа в нужном месте экрана. В режиме ввода информации нажатие одной из клавиш (или комбинации двух клавиш) приводит к передаче и запоминанию в ЭВМ соответствующего кода. Рассмотрим правила получения двоичных кодов алфавитно-цифровых символов и расшифровки этих кодов.

История этого вопроса уходит в тридцатые годы прошлого столетия, когда американский изобретатель Сэмюэль Морзе начал разработку первого электрического телеграфного аппарата. Буквам латинского алфавита Морзе поставил в соответствие комбинации точек и тире. Это делалось очень остроумным способом: комбинация была тем короче, чем чаще встречалась соответствующая буква в различных сообщениях. Так, буква Е, самая распространенная в английском языке, обозначалась одной точкой. Самыми длинными комбинациями кодировались редко встречающиеся буквы*. Такой подход явился предвестником эффективного (с точки зрения компактности и меньшей подверженности влиянию помех при передаче) кодирования, методы которого были созданы в середине XX века**.

---

* Относительные частоты букв английского языка были определены Морзе не менее остроумно: путем подсчета литер в ячейках типографской кассы для набора текстов.

** Эти методы обосновываются в математической теории связи (другое название — теория информации), созданной в 1948 году выдающимся американским ученым Клодом Шенноном. Основы этой теории увлекательно изложены в научно-популярной книге Дж. Пирса «Символы, сигналы, шумы». Любопытно, что в начале своей работы Морзе применял еще более эффективный с точки зрения теории информации способ кодирования, при котором элементарные комбинации точек и тире соответствовали не отдельным буквам, а целым словам.

Если в *азбуке Морзе* обозначать тире единицей, а точку — нулем, то любой символ может быть выражен двоичным числом длиной максимум в пять бит (Морзе использовал $2^5 = 32$ различных комбинаций точек и тире, что хватало для обозначения латинских букв). Фактически в аппаратах Морзе использовалась троичная система: короткая посылка тока — точка, длинная посылка — тире, отсутствие тока — символ разделителя информации. Третий символ необходим для разделения как самих точек и тире (короткая пауза, с длительностью как у точки), отдельных букв (пауза той же длительности, что у тире), так и слов (пауза вдвое большей длительности).

В обычных телеграфных аппаратах, применяемых и по сей день, чаще используют двоичный *код Бодо*. В этом коде все буквы передаются комбинациями непременно из пяти сигналов, а сами сигналы представляют собой посылки тока или паузы одинаковой длительности. Специальные разделители не нужны: всегда известно, что следующая буква начнется спустя ровно пять сигналов, а для обозначения промежутков между словами можно использовать специальный символ пробела, кодируемый как одна из букв.

Основной недостаток рассмотренных кодов — малое число комбинаций, которых не хватает даже для всех букв русского алфавита, не говоря уже про цифры и знаки препинания: для их обозначения приходится идти на всевозможные ухищрения. Так, всем известно, что точка в телеграмме обозначается как «тчк».

Интересно, что увеличить число представляемых символов можно и без напрашивающегося перехода к кодам длиной более чем в 5 бит. При этом используется идея, аналогичная переключению верхнего и нижнего регистров в пишущих машинках. Вспомним, что реальное число печатаемых ими символов вдвое больше, чем имеющихся клавиш. Так, в зависимости от положения переключателя регистров нажатие одной и той же клавиши может привести к печати заглавной или прописной буквы. Чтобы реализовать эту идею при передаче информации, два из 32 пятибитных кодов следует использовать как сигналы перевода регистра. Таким образом обеспечивается передача почти удвоенного числа $(32 \cdot 2 - 2)$ символов, которые могут обозначать не только буквы, десятичные цифры и знаки пунктуации, но и исполнять функции контроля и управления аппаратурой связи. Недостаток такого подхода — необходимость надежного запоминания в принимающем устройстве последнего кода переключения. Тем не менее пятибитные коды до сих пор используются в технике связи, они же применялись для работы с алфавитно-цифровой информацией и в первых ЭВМ.

3.18. Стандартные алфавитно-цифровые коды для ЭВМ

В современной вычислительной технике наиболее употребимы семи- и восьмибитные коды, принятые как международные стандарты. Рассмотрим семибитную систему кодов в отечественном стандарте *КОИ-7* (код для обмена информацией, семибитный). Любой код этой системы удобно разделить на два поля, причем первые три бита определяют номер группы символов (зоны), а остальные — номер в зоне. Тогда, в зависимости от значения битов в этих полях, все 128 возможных кодов можно представить в компактном виде (табл. 3.2).

Таблица 3.2

$a_6a_5a_4$ (зона) / $a_3a_2a_1a_0$ (номер в зоне)	000 0	001 1	010 2	011 3	100 4	101 5	110 6	111 7
0 0 0 0	NUL	DLE	SP	0	@	P	Ю	П
0 0 0 1	SOH	DC1	!	1	A	Q	А	Я
0 0 1 0	STX	DC2	"	2	B	R	Б	Р
0 0 1 1	ETX	DC3	#	3	C	S	Ц	С
0 1 0 0	EOT	DC4	¤	4	D	T	Д	Т
0 1 0 1	ENQ	NAK	%	5	E	U	Е	У
0 1 1 0	ACK	SYN	&	6	F	V	Ф	Ж
0 1 1 1	BEL	ETB	'	7	G	W	Г	В
1 0 0 0	BS	CAN	(	8	H	X	Х	Ь
1 0 0 1	HT	EM	)	9	I	Y	И	Ы
1 0 1 0	LF	SUB	*	:	J	Z	Й	З
1 0 1 1	VT	ESC	+	;	K	[	К	Ш
1 1 0 0	FF	FS	,	<	L	\	Л	Э
1 1 0 1	CR	GS	—	=	M	]	М	Щ
1 1 1 0	SO	RS	.	>	N	↑	Н	Ч
1 1 1 1	SI	US	/	?	O	—	О	DEL

Наиболее употребимые в вычислительной технике спецсимволы означают: BEL — звонок, CR — возврат каретки (ВК), LF — перевод строки (ПС), DEL — вычеркивание, т. е. забой (ЗБ). Каждый символ в таблице находится на пересечении столбца, определяемого тремя старшими битами кода, и строки, соответствующей четырем младшим битам. Так, код $123_8 = 101\ 0011_2$

означает букву S, а символу CR отвечает код $0001101_2 = 015_8$.

Основные особенности приведенной системы кодирования:

1. *Спецсимволы* (символы управления), расположенные в зонах 0 и 1, в основном предназначены для управления работой устройств, подключаемых к ЭВМ. В устройствах вывода информации, например в печатающих, поступление спецсимвола приводит к выполнению конкретного действия типа перевода каретки и не сопровождается печатью какой-либо информации. Для получения кодов спецсимволов в устройствах ввода используют либо специальные клавиши (возврат каретки, перевод строки), либо комбинации из двух клавиш, что упрощает построение клавиатуры. Обычно одновременно нажимаются клавиша, соответствующая латинской букве строки таблицы 3.2, в которой расположен нужный спецсимвол, и клавиша, предназначенная для получения символов управления (СУ). Так, символ BEL получается одновременным нажатием клавиш СУ и G. (В литературе такую комбинацию принято обозначать СУ/G.)

2. Коды десятичных цифр можно получить, приписывая слева комбинацию 011 к четырехразрядной записи этих цифр в двоичной системе счисления. (В восьмеричной системе к такому же результату приведет сложение восьмеричной записи цифры с числом 60_8.)

3. Символы зон 2—7, за исключением спецсимвола DEL, печатаются или отображаются на устройствах вывода. Символ *DEL (забой)* используется для устранения ошибок. Если ошибка допущена при выводе на носитель информации, с которого можно стереть предыдущую информацию, например на экран дисплея, то подача DEL приводит к стиранию последнего отображаемого символа. На носителях, с которых нельзя стереть информацию, например на перфоленте, ошибочный символ можно устранить пробивками поперек ленты во всех позициях, что соответствует коду DEL (1111111_2).

4. Если в поле зоны средний бит игнорируется, то коды латинских и русских букв, лежащие в одной строке (табл. 3.2), становятся неразличимыми. Иногда это позволяет обойтись устройством вывода, рассчитанным на буквы единственного алфавита (русского или латинского). Из ЭВМ в это устройство могут поступать коды любых букв, но выводиться будут буквы только одного алфавита.

5. Последовательность кодов букв латинского алфавита образует возрастающий ряд двоичных чисел. Это часто используется в программах, например, когда нужно расположить список имен в алфавитном порядке.

6. В англоязычных странах применяется *код ASCII* (см. Список сокращений), в котором зоны 6 и 7 таблицы 3.2 используются для кодирования строчных букв латинского алфавита.

В коде *КОИ-8* для представления каждого символа отводится 8 бит (байт), что позволяет расширить набор символов строчными буквами русского и латинского алфавитов и др. знаками.