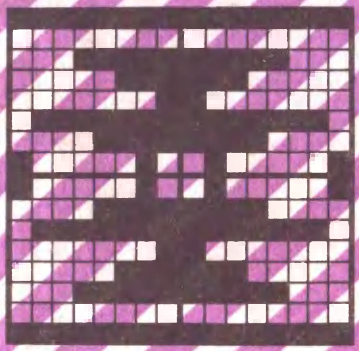
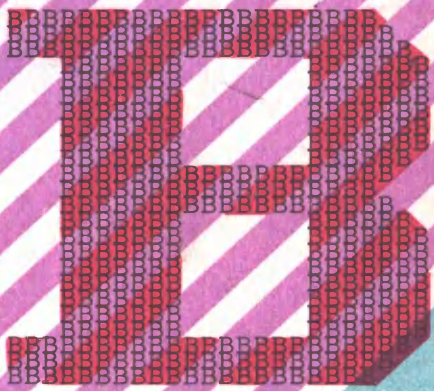
ЭЛЕКТРОННЫЕ

ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ

2. Основы информатики





ЭЛЕКТРОННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ

В **восьми** книгах

Под редакцией
д-ра техн. наук проф. А.Я.Савельева

А.Я.Савельев, Б.А.Сазонов, Э.К.Скуратович
Н.М.Когдов

Основы информатики

КНИГА **2**

**Издание второе,
переработанное
и дополненное**

Москва «Высшая школа» 1991

ББК 32.973
Э45
УДК 681.3

Рецензент: кафедра вычислительной техники Ленинградского электротехнического института им. В. И. Ульянова-Ленина

Э45 **Электронные вычислительные машины.** В 8 кн. Кн. 2. Основы информатики: Практ. пособие для вузов/А. Я. Савельев, Б. А. Сазонов, Э. К. Скуратович, Н. М. Когдов; Под ред. А. Я. Савельева.— 2-е изд., перераб. и доп.— М.: Высш. шк., 1991.— 159 с.: ил.

ISBN 5-06-001767-2

Во второй книге серии «ЭВМ» рассмотрены основные понятия информатики и вычислительной техники, изложены сведения об информационных и логических основах цифровой вычислительной техники, устройстве и взаимодействии основных узлов и блоков ЭВМ. Даны представления о средствах ввода-вывода и хранения информации в ЭВМ.

Второе издание (1-е — 1987 г.) дополнено новыми материалами.

Э $\frac{2405000000 - 433}{001(01) - 91}$ 187 — 91

ББК 32.973

6Ф7

ISBN 5-06-001767-2

Введение

На многих современных устройствах и приборах, используемых в быту и на производстве, организации-изготовители ставят специальный знак «управляется микропроцессором», тем самым давая знать потребителю, что данное устройство коренным образом отличается от своих предшественников, такого знака не имеющих. Так, даже для людей, далеких от техники, термин «микропроцессорная техника» становится синонимом понятий надежная техника; техника, обеспечивающая максимальные удобства эксплуатации; техника, наилучшим образом выполняющая возложенные на нее функции.

Не вдаваясь в особенности терминологии, отметим, что понятие «микропроцессорная техника» входит в более общее понятие «вычислительная техника», а процесс широкого внедрения вычислительной техники во все сферы человеческой деятельности получил название компьютеризации. Причем под *компьютеризацией* понимается не столько рост числа используемых обществом электронных вычислительных машин (ЭВМ), сколько перестройка профессионального и в определенной мере социального мышления современного человека, который обязан по-новому подходить к пониманию решаемых сегодня задач, методов и средств достижения поставленных целей.

Вычислительная техника является определяющим компонентом таких составляющих научно-технического прогресса, как робототехника и гибкие производственные системы, автоматизированные системы проектирования и управления. С широким внедрением вычислительной техники в народное хозяйство связывается возможность перевода его на путь интенсивного развития.

Миниатюрная вычислительная машина (микропроцессор) становится составной частью практически любого прибора, устройства, агрегата. Нет ни одной отрасли промышленности, где применение вычислительной техники не сулило бы существенного выигрыша в эф-

фективности производства, совершенствования качества выпускаемой продукции. С широким использованием вычислительной техники связываются планы по коренному совершенствованию систем телевизионной и телефонной связи, медицинского обслуживания населения, образования.

Выдающийся советский ученый В. М. Глушков писал: «Развитие сетей ЭВМ и систем терминального доступа к ним приводит к тому, что все большая часть информации, прежде всего научно-технической, экономической и социально-политической, помещается в память ЭВМ... К началу следующего столетия в технически развитых странах основная масса информации будет храниться в безбумажном виде — в памяти ЭВМ. Тем самым человек, который в начале XXI века не будет уметь пользоваться этой информацией, уподобится человеку начала XX века, не умевшему ни читать, ни писать» [5].

С учетом этого уже в ближайшие годы существенно изменятся требования к знаниям и практическим навыкам специалистов различных профессий. Главным для них станет не умение выполнить какую-то конкретную работу — выточить деталь на токарном станке, раскроить ткань, рассчитать заработную плату, выполнить расчет прочности конструкции и т. п., а умение поручить выполнение этой работы ЭВМ или автомату, управляемому ЭВМ.

Ответом на потребность дать специалистам знания в области практического использования вычислительной техники стало появление новой научной и учебной дисциплины — *информатики,* изучение основ которой в настоящее время начинается со школьной скамьи.

Круг вопросов, относящихся к информатике как к *науке о методах и средствах обработки информации и решения задач в электронных вычислительных машинах,* очень широк и «объединяет различные стороны программирования и использования ЭВМ, а также методов их конструирования и разработки программного обеспечения»*.

Основное назначение настоящей книги — помочь каждому, кто приступает к изучению информатики и вычислительной техники, усвоить главные понятия и терминологию. В книге в доступной форме излагаются сведения об информационных основах цифровой вычислительной техники, принципах представления, обработки и хранения информации в ЭВМ.

---

*А. П. Ершов. Предисловие к книге [4].

Глава 1
ИНФОРМАЦИЯ И ЕЕ ПРЕДСТАВЛЕНИЕ В ЭВМ

1.1. ОБЩИЕ СВЕДЕНИЯ ОБ ИНФОРМАЦИИ И ВЫЧИСЛИТЕЛЬНЫХ МАШИНАХ

Широко известно, что вычислительная машина — это средство для автоматизации вычислений. Однако известно и другое: вычислительные машины используются широко и для решения обширного круга задач в науке, технике, медицине, связи и т. д.

В «Энциклопедии кибернетики» термин «вычислительная машина» определяется следующим образом: «Вычислительная машина (ВМ) — физическая система (устройство или комплекс устройств), предназначенная для механизации или автоматизации процесса алгоритмической обработки информации и вычислений»*. Таким образом, понятие «вычислительная машина» тесно связано с понятиями «информация» и «алгоритмическая обработка».

Информация определяет многие процессы, происходящие в вычислительной машине. Например, в общей форме процесс решения задачи на вычислительной машине проходит следующие этапы:

— ввод информации или установка исходных данных;

— переработка или преобразование введенной информации;

— определение результатов и вывод переработанной информации.

Таким образом, вычислительная машина получает информацию, запоминает ее, обрабатывает по заданным алгоритмам и направляет потребителю (пользователю) или передает в другие системы обработки.

Термин «информация» имеет множество определе-

*Главная редакция Украинской советской энциклопедии. Т. I, 1975, с. 204.

ний. Прежде всего в широком смысле *информация — это отражение реального мира;* в узком смысле *информация — это любые сведения, являющиеся объектом хранения, передачи и преобразования.* И то и другое определения важны для понимания процессов функционирования вычислительной машины.

С практической точки зрения информация всегда представляется в виде *сообщения.* Информационное сообщение связано с *источником информации, приемником информации и каналом передачи.* Сообщение от источника к приемнику передается в материально-энергетической форме (электрический, световой, звуковой сигнал и т. д.). Человек воспринимает сообщения посредством органов чувств. Приемники информации в технике воспринимают сообщения с помощью различной измерительной и регистрирующей аппаратуры. И в том и другом случае с приемом информации связано изменение во времени значений какой-либо величины, характеризующей состояние приемника. В этом смысле информационное сообщение можно представить функцией $x(t)$, характеризующей изменение во времени материально-энергетических параметров физической среды, в которой осуществляются информационные процессы.

Чаще всего функция $x(t)$ принимает любые вещественные значения в диапазоне изменения аргумента t. Например, температура в интервале времени измерения непрерывно изменяется от некоторого начального значения до некоторого конечного. При этом функция $x(t)$ передает характер изменения температуры во времени. Непрерывно изменяется во времени, например, давление атмосферного воздуха. В этом случае имеет место *непрерывная,* или *аналоговая, информация,* источником которой обычно являются различные природные объекты, объекты технологических производственных процессов и др.

Информационные сообщения, используемые человеком, чаще имеют характер *дискретных сообщений.* Таковыми в древности являлись, например, сигналы тревоги, передаваемые посредством световых и звуковых сообщений. Дискретными являются также языковые сообщения, передаваемые в письменном виде или с помощью звуковых сигналов; сообщения, передаваемые с помощью жестов, и др.

Из-за ограниченной точности чувственного восприятия и используемых измерительных приборов человек непрерывную информацию чаще всего воспринимает

в дискретной форме. Примером этого является определение по термометру цифрового значения температуры с определенной точностью.

В ряде случаев переход от непрерывного представления сигнала к дискретному дает значительные преимущества при передаче, хранении и обработке информации. В информационной технике для этих целей широко используются специальные устройства — *— аналого-цифровые преобразователи.*

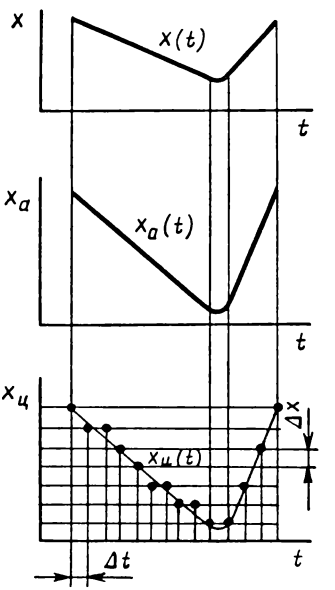


Рис. 1.1

Примером хорошо известных аналого-цифровых преобразователей могут служить используемые в магазинах электронные весы, преобразующие аналоговую величину («вес продукта») в ее цифровое представление, выраженное в граммах и килограммах.

Пример аналогового $x_а(t)$ и дискретного $x_ц(t)$ представления некоторой функции $x(t)$, характеризующей график изменения во времени, например температуры воздуха, приведен на рис. 1.1. В этом случае функция $x_а(t)$ отражает изменение во времени длины столбика ртути термометра, измеряющего температуру $x(t)$, а функция $x_ц(t)$ задает считываемые со шкалы термометра значения температуры с точностью Δx через интервалы времени Δt.

В зависимости от вида перерабатываемой информации вычислительные машины (ВМ) делят на два основных класса: аналоговые и цифровые (рис. 1.2).

Аналоговая вычислительная машина (АВМ) — машина, оперирующая информацией, представленной в виде непрерывных изменений некоторых физических величин. При этом в качестве физических переменных используются сила тока в электрической цепи, угол поворота вала, скорость или ускорение движения тела и т. п. Используя тот факт, что многие явления в природе математически описываются одними и теми же уравнениями, АВМ позволяют с помощью одного физического процесса моде-

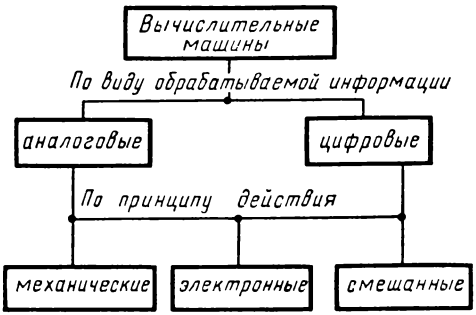


Рис. 1.2

лировать различные иные процессы. АВМ обычно предназначаются для решения определенного класса задач, т. е. являются специализированными.

Цифровая вычислительная машина (ЦВМ) — машина, оперирующая информацией, представленной в дискретном виде.

В настоящее время разработаны методы численного решения многих видов уравнений, что дало возможность решать на ЦВМ различные уравнения и задачи с помощью набора простых арифметических и логических операций. Поэтому любая ЦВМ, как правило, является универсальным вычислительным средством.

По принципу действия основных узлов АВМ и ЦВМ разделяют на механические, электронные, смешанные (оптоэлектронные, гидромеханические, электромеханические и т. п.).

Наибольшее распространение получили электронные вычислительные машины, выполненные с использованием новейших достижений электроники на основе полупроводников и интегральных схем.

1.2. КОДИРОВАНИЕ ИНФОРМАЦИИ В ЭВМ

Объектом передачи и преобразования в ЦВМ является дискретная информация. Для представления ее применяется так называемый алфавитный способ, основой которого является использование фиксированного конечного набора символов любой природы, называемого *алфавитом*.

Примерами алфавитов могут служить алфавиты естественных человеческих языков, совокупность десятичных цифр, любая другая упорядоченная совокупность знаков, предназначенная для образования и передачи сообщений.

Символы из набора алфавита называются *буквами*, а любая конечная последовательность букв — *словом* в этом алфавите. При этом не требуется, чтобы слово обязательно имело языковое смысловое значение. Например, словами являются последовательности символов, составленные из алфавита, включающего в себя русские буквы и цифры:

1986 АААА Ы1568 ФОРТРАН

Все процессы, происходящие в вычислительной системе, связаны непосредственно с различными физическими *носителями информационных сообщений*, а все узлы и блоки этой системы являются физической средой, в которой осуществляются информационные процессы. Особенности носителя информации накладывают определенные ограничения на используемый для ее представления алфавит. При подготовке задачи к решению на ЭВМ исходная формулировка, описание метода решения, задание конкретных исходных данных осуществляются на математическом языке, алфавит которого наряду с буквами естественного языка может включать буквы других языков, специальные символы знаков математических операций и другие знаки. Носителем информации на данном этапе служат листы обычной бумаги.

Для ввода в ЭВМ информация об условиях задачи и методе ее решения должна быть перенесена на специальный носитель, с которого она воспринимается ЭВМ. В качестве такого носителя используются специальные бумажные карты (перфокарты) или ленты (перфоленты), на которые буквы, цифры, другие символы наносятся с помощью специальной системы знаков, например совокупности пробитых и непробитых позиций. В качестве носителя применяются также магнитные диски и ленты; при этом для нанесения информации используется набор из двух знаков, каждый из которых представляется участком магнитной поверхности различной намагниченности. Носителем информации в электронных блоках ЭВМ, ведущих ее обработку, является электрический сигнал, у которого меняется какой-либо параметр (частота, амплитуда).

Как видно из приведенных примеров, в процессе ввода, хранения, вывода и обработки информации в ЭВМ осуществляется неоднократное ее преобразование из одной формы представления в другую. При этом с каждой из используемых форм представления информации связа-

ны различные алфавиты. Для представления информации на перфокартах и перфолентах используется, например, алфавит, включающий в себя всего две буквы (есть пробивка, нет пробивки).

Таким образом, процесс преобразования информации часто требует представлять буквы одного алфавита средствами (буквами, словами) другого алфавита. Такое представление называется *кодированием*. Процесс обратного преобразования информации относительно ранее выполненного кодирования называется *декодированием*.

Для представления информации в ЭВМ преимущественное распространение получило *двоичное кодирование*, при котором символы вводимой в ЭВМ информации представляются средствами *двоичного алфавита*, состоящего из двух букв. В дальнейшем в качестве этих букв используются символы **0** и **1**.

Двоичный алфавит по числу входящих в него символов является минимальным, поэтому при двоичном кодировании алфавита, включающего в себя большее число букв, каждой букве ставится в соответствие последовательность нескольких двоичных знаков или двоичное слово. Такие последовательности называются *кодовыми комбинациями*. Полный набор кодовых комбинаций, соответствующих двоичному представлению всех букв кодируемого алфавита, называется *кодом*.

Различают коды равномерные, неравномерные. Кодовые комбинации равномерных двоичных кодов содержат одинаковое число двоичных знаков, неравномерных — неодинаковое.

Примером неравномерного двоичного кода может служить азбука Морзе, в которой для каждой буквы и цифры определена двоичная последовательность коротких и длинных сигналов. В азбуке Морзе букве Е, например, соответствует один короткий сигнал (точка), а букве Ш — четыре длинных сигнала (четыре тире). Неравномерное кодирование позволяет повысить скорость передачи сообщений за счет того, что наиболее часто встречающимся в передаваемых текстах символам (к ним относится и буква Е) назначается для ее представления более короткая комбинация.

В вычислительной технике используются обычно равномерные коды. Такими являются, например, код обмена информацией — КОИ-8, используемый в ЭВМ для ввода и вывода информации; двоичный код обмена информацией — ДКОИ и др.

Таблица 1.1

Символы	Кодовые комбинации ДКОИ								
	8	7	6	5	4	3	2	1	
0	1	1	1	1	0	0	0	0	
1	1	1	1	1	0	0	0	1	
2	1	1	1	1	0	0	1	0	
5	1	1	1	1	0	1	0	1	
6	1	1	1	1	0	1	1	0	
7	1	1	1	1	0	1	1	1	
+	0	1	0	0	1	1	1	0	
·	0	1	0	0	1	0	1	1	
=	0	1	1	1	1	1	1	0	
O	1	1	0	1	0	1	1	0	
S	1	1	1	0	0	0	1	0	
T	1	1	1	0	0	0	1	1	
P	1	1	0	1	0	1	1	1	
␣ (пробел)	0	1	0	0	0	0	0	0	

Число символов, составляющих кодовую комбинацию, называется *длиной кода.*

В отношении двоичных кодов наряду с термином «длина кода» используют термин *разрядность кода.* Если разрядность двоичного кода обозначить через n, то легко убедиться в том, что полное число кодовых комбинаций равно 2^n. Для ДКОИ $n=8$, а полное число кодовых комбинаций составляет 256, что позволяет закодировать строчные и прописные буквы латинского алфавита, буквы русского алфавита, отличные по написанию от латинских букв, цифры, знаки препинания, знаки математических операций, некоторые специальные символы.

В табл. 1.1 приведены примеры кодовых комбинаций для представления в ДКОИ цифр, некоторых специальных символов и букв русского и латинского алфавитов. Составленный из этих символов текст, например, сообщения

1.2+2.5␣STOP,

будучи введенным в ЭВМ, хранится в ее памяти в виде последовательности двоичных знаков:

11110001	01001011	11110010	01001110
1	·	2	+
11110010	01001011	11110101	01000000
2	·	5	␣
11100010	11100011	11010110	11010111
S	T	O	P

Количество введенной в ЭВМ информации удобно измерять ее «длиной», выраженной в двоичных знаках, или *битах* (англ. bit, от binary — двоичный и digit — цифра). Так, приведенное выше сообщение имеет длину 96 бит.

Последовательность из восьми двоичных знаков называется *байтом*. Так как для представления символов в ДКОИ используются 8-разрядные кодовые комбинации, то выраженное в байтах количество информации в рассмотренном примере равно числу символов, составляющих сообщение, включая символ (пробел), т. е. 12 байт. Используются и более крупные единицы количества информации килобайт (Кбайт) и мегабайт (Мбайт). При этом $К = 2^{10} = 1024$, $М = 2^{20} = 1\ 048\ 576$.

Названные единицы измерения количества информации используются для характеристики *емкости запоминающих устройств ЭВМ*, т. е. количества информации, которое может храниться в памяти одновременно. Например, в состав оборудования ЭВМ Единой системы входят накопители со сменными магнитными дисками емкостью 29; 100; 200 Мбайт. Персональные ЭВМ комплектуются накопителями на гибких магнитных дисках емкостью 360 и 720 Кбайт, 1,2 и 1,44 Мбайт.

Для того чтобы оценить порядок приведенных значений, определим, какой объем памяти необходим для хранения текста книги в 250 страниц, на каждой из которых размещены 50 строк по 50 символов. Простой подсчет числа символов в книге дает ответ на поставленный вопрос. Для хранения указанной книги необходимо запоминающее устройство емкостью 625 000 байт. Переведем полученное значение в Кбайты:

625 000:1024 = 610,4 Кбайт.

Таким образом, книгу заданного объема можно разместить на гибком диске емкостью 720 Кбайт.

Разъясним смысл широко используемого в информатике термина *данные*. Этот термин принято применять в отношении информации, представленной в виде, позволяющем хранить, передавать или обрабатывать ее с помощью технических средств. Поэтому наряду с терминами «ввод информации», «обработка информации», «хранение информации», «поиск информации» используются термины *ввод данных, обработка данных, хранение данных* и т. п.

Необходимость выполнения арифметических действий над вводимыми в ЭВМ числами предъявляет особые требования к кодированию числовой информации. Для двоичного представления чисел в ЭВМ наиболее удобной оказалась так называемая *двоичная система счисления.*

В общем случае *система счисления* представляет собой совокупность приемов и правил для записи чисел цифровыми знаками. Способов записи чисел цифровыми знаками существует множество. Любая предназначенная для практического применения система счисления должна обеспечивать:

— возможность представления любого числа в рассматриваемом диапазоне величин;

— единственность представления (каждой комбинации символов должна соответствовать одна и только одна величина);

— простоту оперирования числами.

Все системы представления чисел делят на непозиционные и позиционные.

В непозиционной системе счисления значение символов не зависит от положения в числе. Принципы построения таких систем не сложны. Для их образования используют в основном операции сложения и вычитания. Например, система с одним символом-палочкой встречалась у многих народов. Для изображения какого-то числа в этой системе нужно записать определенное множество палочек, равное данному числу. Эта система неэффективна, так как запись числа получается длинной. Другим примером непозиционной системы счисления является римская система, использующая набор следующих символов: I, V, X, L, C, D и т. д. В этой системе имеется отклонение от правила независимости значения цифры от положения в числе. В числах LX и XL символ X принимает значения $+10$ и -10.

В позиционной системе счисления значение цифры определяется положением в числе: один и тот же знак принимает различные значения. Например, в десятичном числе 222 первая цифра справа означает две единицы, соседняя с ней — два десятка, а левая — две сотни.

Десятичная система счисления наиболее распространена в вычислительной практике. Однако своему распространению она обязана не каким-то особым преимуществом перед другими системами, а достаточно случайному обстоятельству — наличию у человека на руках десяти пальцев.

Любая позиционная система счисления характеризуется основанием. *Основание (базис) позиционной системы счисления* — число знаков или символов, используемых для изображения цифр в данной системе. Возможно бесчисленное множество позиционных систем, так как за основание можно принять любое число, образовав новую систему. В вычислительной технике, например, широко используется шестнадцатеричная система счисления, запись чисел в которой производится с помощью следующих знаков (цифр): 0, 1, ..., 9, A, B, C, D, E, F (вместо A, ..., F можно записать любые другие символы, например 1, 2, ..., 5, 6).

Для позиционной системы счисления справедливо равенство

$$A_{(q)} = a_{n-1}q^{n-1} + \ldots + a_1 q^1 + a_0 q^0 + a_{-1}q^{-1} + \ldots + a_{-m}q^{-m}, \quad (1.1)$$

где $A_{(q)}$ — произвольное число, записанное в системе счисления с основанием q; a_i — цифры системы счисления; n, m — число целых и дробных разрядов.

На практике используют сокращенную запись чисел:

$$A_{(q)} = a_{n-1} \ldots a_1 a_0 a_{-1} \ldots a_{-m}.$$

Так, сокращенной записи десятичного числа 86,54 соответствует его значение, вычисляемое согласно равенству (1.1):

$$86{,}54_{(10)} = 8 \cdot 10^1 + 6 \cdot 10^0 + 5 \cdot 10^{-1} + 4 \cdot 10^{-2}.$$

В двоичной системе счисления для представления чисел используются две цифры — **0** и **1**. В соответствии с равенством (1.1) значение двоичного числа (1001, 1101), взятого в качестве примера, можно определить следующим образом:

$$1001{,}1101_{(2)} = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4}.$$

Если по правилам десятичной арифметики выполнить действия в правой части приведенного равенства, то можно получить значение десятичного эквивалента двоичного числа $1001{,}1101_{(2)}$, равного $9{,}8125_{(10)}$.

В табл. 1.2 приведены эквиваленты десятичных цифр в некоторых системах счисления.

Таблица 1.2

Десятичная цифра	Эквиваленты в системах счисления			Десятичная цифра	Эквиваленты в системах счисления		
	$q=2$	$q=8$	$q=16$		$q=2$	$q=8$	$q=16$
0	0	0	0	8	1000	10	8
1	1	1	1	9	1001	11	9
2	10	2	2	10	1010	12	A
3	11	3	3	11	1011	13	B
4	100	4	4	12	1100	14	C
5	101	5	5	13	1101	15	D
6	110	6	6	14	1110	16	E
7	111	7	7	15	1111	17	F

Для записи одного и того же значения в различных системах счисления требуется разное число позиций или разрядов. Например, $96_{(10)}=140_{(8)}=1100000_{(2)}$. Чем меньше основание системы, тем больше длина числа (длина разрядной сетки). Если длина разрядной сетки задана, то это ограничивает максимальное по абсолютному значению число, которое можно записать.

Пусть длина разрядной сетки равна любому положительному числу, например N. Тогда

$$A_{(q)\max}=q^N-1. \tag{1.2}$$

Если же задано максимальное абсолютное значение числа, то длина разрядной сетки

$$N=\log_q (A_{(q)\max}+1). \tag{1.3}$$

Интервал числовой оси, заключенный между максимальным и минимальным числами, называют *диапазоном представления (ДП) чисел* в данной системе счисления для заданного ограничения на длину разрядной сетки:

$$-A_{(q)\max}\leqslant ДП\leqslant A_{(q)\max}. \tag{1.4}$$

В ЦВМ длина обрабатываемых чисел обычно ограничена значениями: 1 байт (8 разрядов), 2 байт (16 разрядов), 4 байт (32 разряда), 8 байт (64 разряда). Соответственно ограничены и значения записываемых с использованием указанной длины разрядной сетки целых положительных чисел. Так, максимальное целое положительное число, которое можно записать с ис-

пользованием 16 двоичных разрядов, равно $2^{16}-1=65\,535$.

Использование двоичной системы счисления для ЭВМ связано с преодолением дополнительных трудностей, вызванных необходимостью перевода вводимых в ЭВМ чисел в двоичную систему счисления и обратного преобразования числовых данных при выводе из ЭВМ. Эти преобразования в ЭВМ осуществляются автоматически с использованием специально разработанных методов.

Для перевода целых чисел и целых частей неправильных дробей из системы счисления с основанием q_1 в новую систему счисления с основанием q_2 используется метод, базирующийся на делении переводимого числа на основание новой системы счисления. В соответствии с (1.1) целое число $A_{(q_1)}$ в системе с основанием q_2 записывается в виде

$$A_{(q_2)}=b_{n-1}q_2^{n-1}+b_{n-2}q_2^{n-2}+\ldots+b_1q_2^1+b_0q_2^0.$$

Переписав это выражение по схеме Горнера, получим

$$A_{(q_2)}=(\ldots(((b_{n-1}q_2+b_{n-2})q_2+b_{n-3})q_2+\ldots b_1)q_2+b_0). \quad (1.5)$$

Представление числа $A_{(q_2)}$ в виде (1.5) позволяет уяснить суть рассматриваемого метода. Если правую часть выражения (1.5) разделить на величину основания q_2, то можно получить целую часть $(\ldots(b_{n-1}q_2+b_{n-2})q_2+\ldots+b_1)$ и остаток b_0. Разделив полученную часть на q_2, найдем второй остаток b_1. Повторяя процесс деления n раз, получим последнее частное b_{n-1}, которое в соответствии с (1.5) является старшей цифрой n-разрядного числа, представленного в системе с основанием q_2. При переводе все арифметические действия должны вычисляться по правилам системы счисления с основанием q_1.

Пример 1.1. Перевести десятичное число $A=98$ в двоичную систему счисления $(q_2=2)$.

Решение:

98	2										
98	49	2									
$b_0=0$	48	24	2								
	$b_1=1$	24	12	2							
		$b_2=0$	12	6	2						
			$b_3=0$	6	3	2					
				$b_4=0$	2	$1=b_6$					
					$b_5=1$						

Ответ: $A_{(2)}=b_6b_5b_4b_3b_2b_1b_0=1100010$.

Для перевода правильных дробей из системы счисления с основанием q_1 в систему с основанием q_2 используется метод, базирующийся на умножении переводимой правильной дроби на основание q_2 новой системы счисления. Правильную дробь $A_{(q_1)}$ в системе с основанием q_2 можно записать в виде

$$A_{(q_2)}=b_{-1}q_2^{-1}+b_{-2}q_2^{-2}+\ldots+b_{-m}q_2^{-m}.$$

Переписав это выражение по схеме Горнера, получим

$$A_{(q_2)}=q_2^{-1}(b_{-1}+q_2^{-1}(b_{-2}+\ldots+q_2^{-1}(b_{-(m-1)}+$$

$$+q_2^{-1}b_{-m})\ldots)). \qquad (1.6)$$

Если правую часть выражения (1.6) умножить на q_2, то можно найти неправильную дробь, в целой части которой будет число b_{-1}. Умножив затем оставшуюся дробную часть на величину основания q_2, получим дробь, в целой части которой будет b_{-2} и т. д. Повторяя процесс умножения m раз, найдем все m цифр дробной части числа в новой системе счисления. При этом все действия должны выполняться по правилам q_1-арифметики и, следовательно, в целой части получающихся дробей будут появляться эквиваленты цифр новой системы счисления, записанные в исходной системе счисления.

Пример 1.2. Перевести десятичную дробь $A=0{,}625$ в двоичную систему счисления $(q_2=2)$.

Решение. Выполним следующие действия:

0,	625 2
$b_{-1}=1$,	250 2
$b_{-2}=0$,	500 2
$b_{-3}=1$,	000 2
$b_{-4}=0$,	000

Ответ: $A_{(2)}=0, b_{-1}b_{-2}b_{-3}b_{-4}=0{,}1010_{(2)}$.

При переводе правильных дробей из одной системы счисления в другую можно получить дробь в виде бесконечного или расходящегося ряда. Процесс перевода можно закончить, если появится дробная часть, имеющая во всех разрядах нули, или будет достигнута заданная точность перевода (получено требуемое число разрядов результата). Последнее означает, что при переводе дроби необходимо указать число разрядов в случае ее представления в новой системе счисления. Естественно, что при этом возникает погрешность перевода чисел. В ЭВМ точность перевода обычно ограничивается длиной разрядной сетки, отведенной для представления чисел.

Для перевода неправильных дробей из одной системы счисления в другую необходим раздельный перевод целой и дробной частей по правилам, описанным выше. Полученные результаты записывают в виде новой дроби в системе с основанием q_2.

Пример 1.3. Перевести десятичную дробь $A=98{,}625$ в двоичную систему счисления $(q_2=2)$.

Решение: Результаты перевода соответственно целой и дробной частей возьмем из примеров 1.1 и 1.2.

Ответ: $A_{(q_2)}=1100010{,}1010$.

В информатике и вычислительной технике разработано множество других методов перевода чисел из одной системы счисления в другую, позволяющих получить результат с меньшими затратами времени на преобразования.

1.4. ФОРМЫ ПРЕДСТАВЛЕНИЯ В ЭВМ ЧИСЛОВЫХ ДАННЫХ

В математике широко используются две формы записи чисел: естественная и нормальная.

При естественной форме число записывается в естественном натуральном виде, например: 12 560 — целое число, 0,003572 — правильная дробь, 4,89760 — неправильная дробь.

При нормальной форме запись одного числа может быть различной в зависимости от ограничений, накладываемых на ее форму. Например, число 12 560 может быть записано так: $12\,560 = 1{,}256 \cdot 10^4 = 0{,}1256 \cdot 10^5 = 125\,600 \cdot 10^{-1}$ и т. д.

При естественном представлении чисел в ЭВМ устанавливаются длины разрядной сетки, а также целой и дробной частей. При этом распределение разрядов между целой и дробной частями не изменяется и остается постоянным независимо от величины числа. В связи с этим существует также и другое название этой формы представления числа с фиксированной запятой. В современных вычислительных машинах эта форма используется преимущественно для представления целых чисел.

Так как числа бывают положительные и отрицательные, то в разрядной сетке при машинном представлении один разряд отводится под *знак числа*, а остальные образуют *поле числа* (рис. 1.3, *а*). В знаковый разряд, который может располагаться как в начале, так и в конце числа, записывается информация о знаке числа. Примем, что знак положительного числа «+» изображается символом **0**, а знак отрицательного числа «–» — символом **1**. Если поле числа включает в себя n разрядов, то диапазон представления целых чисел в этом случае огра-

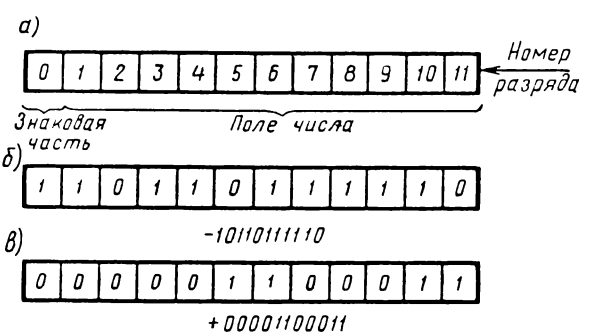


Рис. 1.3

ничивается значениями $-(2^n-1)$ и $+(2^n-1)$. На рис. 1.3, *б, в* приведены примеры записи в форме с фиксированной запятой отрицательного и положительного целых чисел.

Представление числа в ЭВМ в нормальной форме называют также представлением с плавающей запятой, так как положение запятой в записи числа, как показывают приведенные примеры, в этом случае может изменяться.

В нормальной форме запись числа A_n имеет структуру

$$A_n = m_A q^{p_A}, \tag{1.7}$$

где m_A — мантисса числа A; p_A — порядок числа A (характеристика числа).

Чтобы избежать неоднозначности представления чисел, используют так называемую нормализованную форму, для которой справедливо следующее условие:

$$q^{-1} \leqslant |m_A| < 1, \tag{1.8}$$

где q — основание системы счисления.

Так, числа $12{,}56 \cdot 10^2$ и $-0{,}00543 \cdot 10^{-5}$ в нормализованном виде в соответствии с условием (1.8) должны быть записаны следующим образом: $0{,}1256 \cdot 10^4$; $-0{,}543 \cdot 10^{-7}$.

Формат машинного изображения числа с плавающей запятой содержит знаковые части и поля для мантиссы и порядка (рис. 1.4, *а*). Кодирование знаков остается таким же, как и при представлении числа в форме с фиксированной запятой.

Рассмотрим пример записи чисел в форме с плавающей запятой. Пусть в разрядную сетку ЭВМ (рис. 1.4) необходимо записать двоичные числа $A_1 = -10110{,}1111$ и $A_2 = +0{,}000110010111$. Прежде всего эти числа необходимо нормализовать. Порядок чисел выбирают таким образом, чтобы для них выполнялось условие (1.8), т. е. $A_1 = -0{,}101101111 \cdot 2^5$ и $A_2 = +0{,}110010111 \cdot 2^{-3}$. Порядок должен быть записан в двоичной системе счисления. Так как система счисления для данной ЭВМ задана, то нет необходимости указывать на ее основание, достаточно лишь представить показатель порядка (характеристику числа).

Для изображения порядка выделено пять цифровых разрядов для знака — один разряд, поэтому машинные изображения запишутся как $[p_{A_1}] = 000101$; $[p_{A_2}] = 100011$.

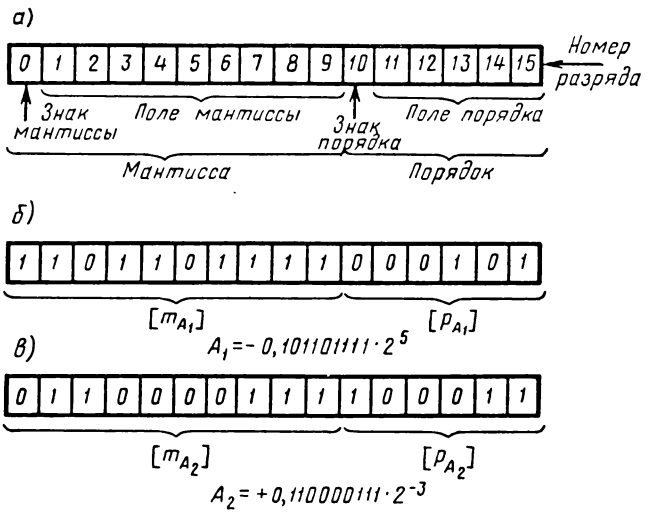


Рис. 1.4

При этом машинные изображения мантисс будут соответственно $[m_{A_1}]=1101101111$; $[m_{A_2}]=0110010111$.

Изображения чисел A_1 и A_2 в форме с плавающей запятой показаны на рис. 1.4, *б, в*.

Представление числовой информации в ЭВМ, как правило, влечет за собой появление погрешностей (ошибок), значения которых зависят как от формы представления чисел, так и от длины разрядной сетки ЭВМ.

Абсолютная погрешность представления — разность между истинным значением величины A и ее значением, полученным из машинного изображения $A_м$:

$$\Delta A = A - A_м. \tag{1.9}$$

Относительная погрешность представления — величина, равная отношению абсолютной погрешности представления входной величины к ее значению в машинном представлении:

$$\delta[A] = \Delta[A]/A_м. \tag{1.10}$$

Целые числа переводятся из десятичной системы в двоичную без погрешностей. Максимальная абсолютная погрешность перевода правильной дроби из десятичной системы счисления в двоичную при пред-

ставлении ее в форме с фиксированной запятой не будет превышать единицы младшего разряда разрядной сетки ЭВМ, т. е. $\Delta[A]_{max}=2^{-n}$, где n — длина разрядной сетки для представления дробной части числа. При этом для максимальной правильной дроби относительная погрешность представления не превысит значения:

$$\delta[A]=\Delta[A]/(A_м)_{max}=2^{-n}/(1-2^{-n})\approx 2^{-n}.$$

Для минимальной правильной дроби относительная погрешность представления ее в форме с фиксированной запятой определится из соотношения

$$\delta[A]_{max}=\Delta[A]/(A_м)_{min}=2^{-n}/2^{-n}=1.$$

Таким образом, погрешности представления малых чисел в форме с фиксированной запятой могут быть значительными, что необходимо учитывать.

При представлении чисел в форме с плавающей запятой абсолютное значение мантиссы находится в пределах

$$2^{-1}\leqslant|[m_A]|\leqslant 1-2^{-n}. \qquad (1.11)$$

Так как погрешность (1.11) относится только к мантиссе, то, для того чтобы найти погрешность представления числа в форме с плавающей запятой, ее надо умножить на величину порядка числа p_A:

$$\left.\begin{aligned}\delta[A]_{max}&=\frac{2^{-n}p_A}{2^{-1}p_A}=2^{1-n};\\ \delta[A]_{min}&=\frac{2^{-n}p_A}{(1-2^{-n})p_A}\approx 2^{-n},\end{aligned}\right\} \qquad (1.12)$$

где n — длина разрядной сетки для представления мантиссы числа.

Из (1.12) следует, что относительная точность представления чисел в форме с плавающей запятой практически не зависит от величины числа.

1.5. ОСОБЕННОСТИ ОТОБРАЖЕНИЯ И ПРЕДСТАВЛЕНИЯ В ЭВМ ГРАФИЧЕСКОЙ ИНФОРМАЦИИ

Устройства ввода и отображения графической информации современных ЭВМ обычно работают с прямоугольными форматами визуальных изображений (рисунков, чертежей, графиков). В общем случае для ввода в ЭВМ и последующего отображения прямоугольное изображение разбивается на элементы прямоугольной решеткой. Число таких элементов на единицу площади изображения характеризует *разрешающую способность системы отображения графической информации.*

В современных ЭВМ для отображения графической информации наряду с другими устройствами используются так называемые *растровые дисплеи,* в которых электронный луч «рисует» точечное изображение, как и в обычном телевизоре, линия за линией, начиная с левого верхнего и заканчивая нижним правым углом экрана. Различия между дисплеями ПК определяются такими характеристиками, как разрешающая способность, количество воспроизводимых на экране цветов и др. Режимы, в которых могут работать дисплеи современных компьютеров, разделяются на два класса: текстовые и графические.

В текстовых режимах на экран дисплея можно выводить символьные тексты и простые рисунки, составленные из специальных знаков, входящих в набор символов персонального компьютера и называемых *символами псевдографики.* На рис. 1.5, *а* приведены некоторые из

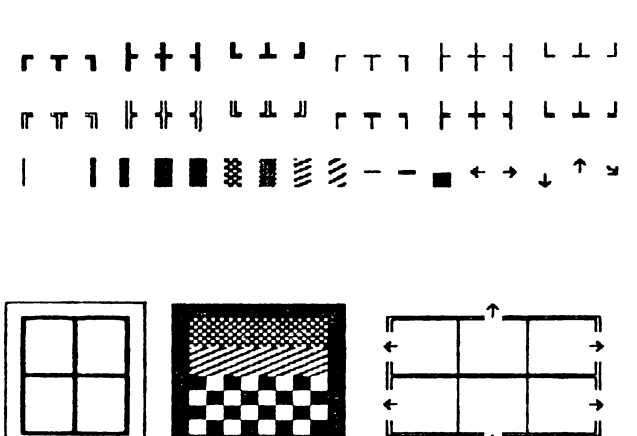


Рис. 1.5

Рис. 1.6

ТЕАТР

Teamp

ТЕАТР

Teamp

meamp

Рис. 1.7

символов псевдографики, а на рис. 1.5, *б* — примеры простых рисунков, составленных из этих символов.

Графические режимы используются для формирования сложных рисунков, а также символов, отличающихся от стандартных размерами и формой. На рис. 1.6 приведен пример изображения, полученного на экране дисплея, работающего в графическом режиме, а на рис. 1.7 — примеры нестандартных графических изображений символов русского алфавита.

Разрешающая способность дисплея характеризуется числом элементов разбиения изображения по вертикали и горизонтали и может быть низкой, средней и высокой (табл. 1.3).

При использовании растрового дисплея в текстовом режиме весь экран обычно разбивается на прямоуголь-

Таблица 1.3

Разрешение	Количество	
•	точек	символов
Низкое	160 × 200	20 × 25
Среднее	320 × 200	40 × 25
Высокое	640 × 200	80 × 25

ники размером 8 × 8 точек, в каждом из которых отображается один символ. Некоторые текстовые дисплеи реализуют более высокое точечное разрешение — 720 × 350 точек при формате символа 9 × 14 точек.

Изображения текстов и рисунков, воспроизводимые на экранах растровых дисплеев, в зависимости от цвета составляющих их точек подразделяются на штриховые, полутоновые и цветные. *Штриховые изображения* составлены из точек, каждая из которых может иметь на экране один из двух возможных уровней яркости, соответствующих двум цветам — черному и белому (коричневому и белому, зеленому и белому и т. п.). *Полутоновые изображения* составляются из точек, которые могут принимать и промежуточные между черным и белым (коричневым и белым и т. п.) градации яркости — градации «серого». Примерами полутоновых изображений являются изображения на экране черно-белого (монохромного) телевизора.

Штриховые и полутоновые изображения характерны для монохромных дисплеев.

Цветные дисплеи воспроизводят *цветные изображения*, составленные из точек, могущих иметь на экране один из возможных цветов, совокупность которых называется *палитрой*. Базовая палитра дисплеев современных ЭВМ обычно включает в себя 16 стандартных цветов и обозначается аббревиатурой IRGB, буквы которой указывают на 4 элемента, определяющих цвет в каждой точке изображения на экране: три компонента цвета — красный (англ. — Red), зеленый (Green), синий (Blue) — и один из двух возможных уровней яркости (Intensity). Все возможные сочетания компонентов дают 16 цветов палитры (табл. 1.4). Наличие каждого из четырех компонентов в цвете палитры указано цифрой **1**, а ее отсутствие — цифрой **0**.

При составлении программ для компьютера цвет в командах, обеспечивающих выдачу на экран графических и символьных изображений, задается номером цвета.

Таблица 1.4

Интенсивность	Красный	Зеленый	Синий	Номер цвета	Название цвета
0	0	0	0	0	Черный
0	0	0	1	1	Синий
0	0	1	0	2	Зеленый
0	0	1	1	3	Голубой
0	1	0	0	4	Красный
0	1	0	1	5	Фиолетовый
0	1	1	0	6	Коричневый
0	1	1	1	7	Белый
1	0	0	0	8	Серый
1	0	0	1	9	Светло-синий
1	0	1	0	10	Светло-зеленый
1	0	1	1	11	Светло-голубой
1	1	0	0	12	Светло-красный
1	1	0	1	13	Светло-фиолетовый
1	1	1	0	14	Желтый
1	1	1	1	15	Ярко-белый

В составе оборудования современных компьютеров используются высококачественные дисплеи особой конструкции, способные отображать палитры из 64, 256 цветов и более. Палитра 64 цветов также построена из трех основных цветов, для каждого из которых могут быть заданы четыре градации интенсивности (яркости). Все возможные комбинации разной интенсивности основных цветов дают 64 цвета палитры. Для реализации 64-цветной палитры электронная трубка дисплея должна обеспечивать возможность раздельного управления яркостью красного, зеленого и синего.

Изображение, выдаваемое на экран дисплея, в закодированном виде хранится в специально отведенной для этого области памяти компьютера, называемой *памятью регенерации изображения*. Память регенерации хранит последовательность кодов, определяющих уровень яркости (цвет) каждой точки. Рис. 1.8 иллюстрирует принцип кодирования точечного штрихового изображения. Каждый элемент штрихового изображения кодируется так, что точке максимальной яркости (белому) ставится в соответствие цифра **1**, а точке минимальной яркости (черному) — цифра **0**. Таким образом, последовательности светлых и темных точек экрана (рис. 1.8, *а*) ставится в соответствие последовательность единиц и нулей, которая и хранится в памяти регенерации (рис. 1.8, *б*). Данные из памяти регенерации периодически считываются, преобразуются в видеосигнал и отображаются на экране.

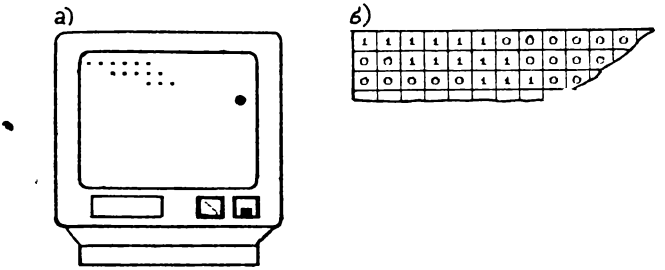


Рис. 1.8

Изображение на экране меняется с определенной частотой (обычно 60 раз в секунду), обеспечивающей ему четкость и стабильность. Электронный луч обегает экран построчно с верхней левой в нижнюю правую точку. Время возвращения луча в верхнюю левую точку (когда он гасится) используется для изменения информации в памяти регенерации.

Память регенерации может быть разделена на страницы, каждая из которых содержит информацию об изображении полного экрана. В этом случае, когда отображается одна страница памяти, в другие вносятся изменения. При необходимости происходит смена страниц и изображение на экране мгновенно изменяется. Многостраничная организация памяти регенерации позволяет удобно реализовывать эффекты движения графических изображений.

Для представления в памяти регенерации цветного изображения каждому элементу (точке) изображения ставится в соответствие код ее цвета. Таким образом, при работе в цветном графическом режиме память регенерации хранит последовательность кодов (номеров) цветов всех точек экрана.

Некоторые типы монохромных дисплеев используются в графических режимах, совместимых с графическими режимами цветных дисплеев. При этом считываемый из памяти регенерации код цвета преобразуется в код уровня яркости серого. На экран вместо цветного выдается полутоновое изображение.

В отличие от графических режимов, позволяющих управлять цветом в каждой точке экрана, в текстовых режимах задаются цвет символа (цвет переднего плана) и цвет фона (цвет заднего плана).

Существенно отличается в текстовом режиме и заполнение памяти регенерации изображения (рис. 1.9). Если

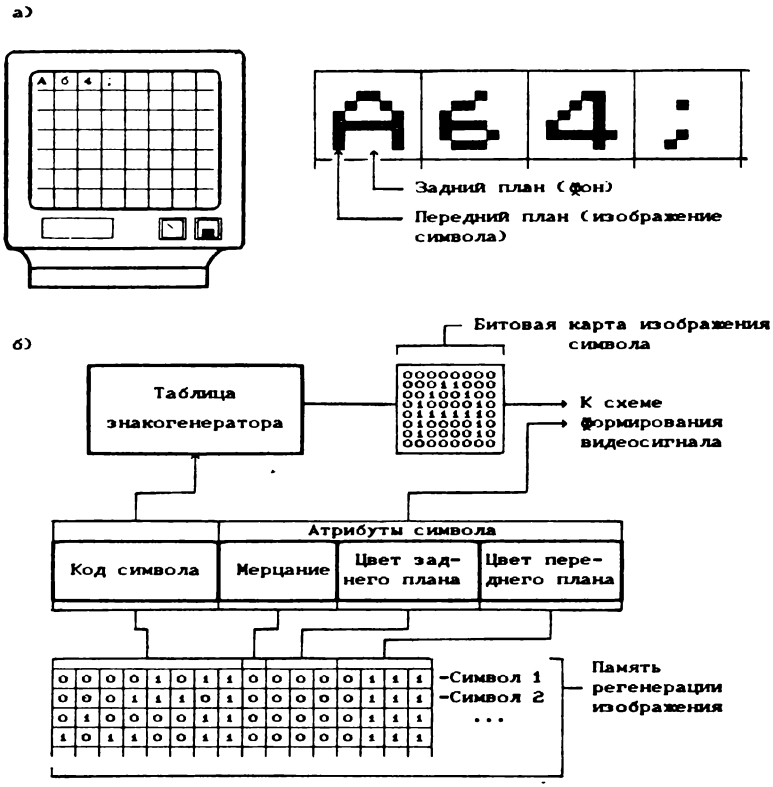


Рис. 1.9

в графическом режиме количество элементов (кодов цвета), заполняемых в карте памяти, соответствует количеству точек экрана, то в текстовом режиме количество элементов памяти регенерации соответствует количеству символов на экране (рис. 1.9, *а*). В каждом элементе памяти регенерации (рис. 1.9, *б*) для соответствующей позиции экрана запоминаются код символа (0—255) и атрибуты его изображения: код цвета переднего плана, код цвета заднего плана и параметр, в зависимости от значения которого символ может выдаваться с постоянной яркостью (значение **0**) или мерцающим на экране (значение **1**).

В монохромных текстовых режимах атрибуты символа могут задавать мерцание, подчеркивание, выделение яркостью и другие параметры.

Закодированное точечное изображение символа называется *битовой картой*. Битовые карты символов в фор-

мате (8 × 8) или (9 × 14) точек хранятся в виде таблиц в специальном блоке памяти, называемым *памятью знакогенератора*, и извлекаются из него по заданным в памяти регенерации изображения кодам символов.

Содержимое *памяти знакогенератора* в современных ЭВМ может изменяться. Это позволяет использовать в одном компьютере разные наборы символов: русские, грузинские, латинские, арабские или любые другие.

В зависимости от того, в каком режиме работает дисплей — в текстовом, графическом, цветном, монохромном и т. д.— по-разному осуществляется преобразование кодов, хранящихся в памяти регенерации изображения, в видеосигнал, поступающий на вход дисплея. Преобразование кода изображения в видеосигнал осуществляется специальной электронной схемой, называемой *адаптером дисплея* или *видеоадаптером*.

В общем случае при вводе в ЭВМ графическое изображение необходимо разбить на элементы (точки) и для каждого из них образовать двоичный код, описывающий параметры изображения: цвет, интенсивность и т. д. На практике изображение обычно сначала преобразуется в электрический сигнал (видеосигнал), для чего используются устройства, подобные телевизионным камерам. Затем оно разбивается на элементы, число которых определяется выбранным разрешением (см. табл. 1.3), и для каждого элемента посредством аналого-цифрового преобразования образуются коды цвета, интенсивности и т. д.

Для ввода в ЭВМ графических изображений, представляющих собой технические чертежи и им подобные по технике исполнения графические рисунки, используется метод, основанный на кодировании и перечислении отдельных геометрических фигур, составляющих изображение.

Чертежи, представленные в качестве примера на рис. 1.10, содержат линии, окружности, дуги. Положение каж-

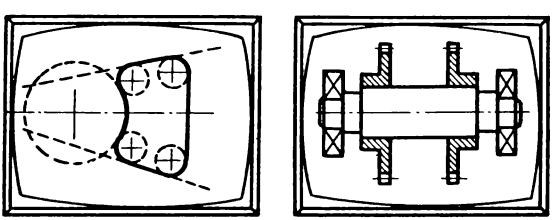


Рис. 10

дой линии на чертеже можно задать координатами двух точек, определяющих ее начало и конец. Кроме того, необходимо задать тип линии: штрихпунктирная, штриховая и т. д. Окружность задается координатами центра, длиной радиуса и типом линии. Дуга задается координатами конца и начала, радиусом и типом линии и т. д. Таким образом, для ввода в ЭВМ чертежа необходимо по определенным правилам подготовить описание составляющих чертеж геометрических элементов и ввести его в ЭВМ как обычную буквенно-цифровую информацию. Дальнейшее преобразование введенного описания в параметры элементов изображения для представления на экране дисплея или на бумаге, посредством печатающего или рисующего устройства, выполняется обычно специальными программами.

1.6. ПРЕДСТАВЛЕНИЕ ЗВУКОВОЙ ИНФОРМАЦИИ

Простейшим примером, поясняющим особенности кодирования звуковой информации, может служить нотная запись музыкальных произведений. Обозначения нот и их длительностей, длительностей пауз, знаков усиления и снижения громкости звука, музыкального ударения и другие знаки, составляющие набор символов нотной азбуки, образуют алфавит, с помощью которого любую музыкальную мелодию можно представить в закодированном виде. Последовательность символов, кодирующих музыкальную мелодию, хранится и обрабатывается в ЭВМ как обычная алфавитно-цифровая информация. Это позволяет использовать ЭВМ не только для анализа, но и для создания музыкальных произведений.

О качестве музыкальных произведений, создаваемых машинами, спорят, но тем не менее многие из современных композиторов признают и широко используют возможности компьютеров при работе над сочинением музыки. Обычно на выход таких ЭВМ подключаются электронные синтезаторы, позволяющие под управлением ЭВМ исполнять достаточно сложные музыкальные произведения.

В большинстве современных персональных ЭВМ (ПЭВМ) в качестве одного из выходных устройств используются простые генераторы музыкальных тонов. По специальным командам ЭВМ эти устройства выдают последовательность музыкальных тонов заданной длительности, что позволяет сопровождать музыкальными фрагментами выдачу на экран дисплея ЭВМ различной информации. Звуковое сопровождение существенно повышает эмоциональные характеристики общения человека

с ЭВМ, что очень важно во многих областях применения вычислительной техники.

Широкие возможности расширения сферы эффективного применения вычислительной техники обусловлены развитием систем ввода в ЭВМ и вывода из ЭВМ речевой информации. Устное сообщение можно представить как последовательность элементарных звуков, называемых *фонемами*, и *пауз* между ними. От числа фонем, выделяемых в устной речи, зависит точность ее описания. На практике для кодирования русской устной речи выделяют порядка 40—45 фонем, каждой из которых ставится в соответствие кодирующее ее обозначение. Последовательность кодов, описывающих фонемы устного сообщения, вводится и хранится в памяти ЭВМ и при необходимости выводится из нее через специальные устройства, называемые *синтезаторами речи*.

В настоящее время синтезаторы речи выпускаются промышленностью и сфера их применения непрерывно расширяется — используются различные автоматизированные информационно-справочные системы, системы автоматизированного контроля, способные голосом предупредить человека о состоянии контролируемого объекта, и другие системы.

Разработаны устройства, позволяющие преобразовать письменный текст в соответствующее ему фонемное представление, что позволяет воспроизводить этот текст на экране дисплея или через синтезатор речи.

Весьма перспективным является создание средств общения человека с ЭВМ посредством голоса. Главная трудность при этом состоит в распознавании речи. В настоящее время созданы системы, позволяющие распознавать до нескольких сотен слов из словаря, предварительно занесенного в память ЭВМ. При этом устное сообщение кодируется не на уровне фонем, а на уровне отдельных слов, каждое из которых рассматривается как элементарный символ алфавита речевого сообщения.

Вопросы для самопроверки

1.1. Поясните различие между аналоговым и дискретным представлением информации. Приведите примеры источников аналоговой и дискретной информации.

1.2. Приведите примеры информационных сообщений, составленных из следующих наборов знаков: свет трех цветов (зеленый, желтый, красный); пара состояний (звук электрического звонка, отсутствие звука).

1.3. Приведите примеры кодирования информации. Установите со-

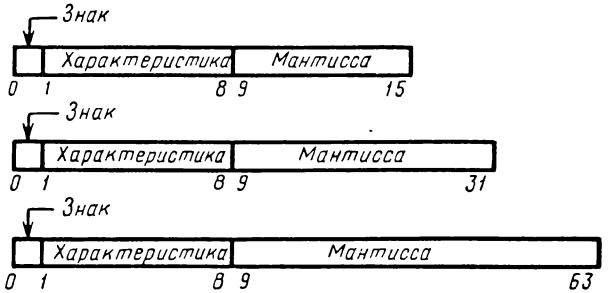


Рис. 1.11

став букв кодируемого алфавита и алфавита, в буквах которого осуществляется кодирование.

1.4. Что такое бит, байт, килобайт, мегабайт?

1.5. Для кодирования используется двоичный код длиной 4 бит. Каково полное число кодовых комбинаций данного кода? Запишите эти комбинации.

1.6. Кодируемый алфавит включает в себя 92 буквы. Какой должна быть минимальная разрядность используемого для кодирования двоичного кода?

1.7. Расшифруйте следующее закодированное в ДКОИ сообщение (используйте табл. 1.1):

11110010010011101111010101111111011110111

1.8. Допустимый объем реферата научной статьи ограничен одной страницей и содержит не более 2000 знаков. Подсчитайте, сколько рефератов может сохраниться в запоминающем устройстве ЭВМ емкостью 100 Мбайт.

1.9. Поясните различие между позиционными и непозиционными системами счисления.

1.10. Что такое основание позиционной системы счисления?

1.11. Запишите эквиваленты десятичных чисел в троичной и пятеричной системах счисления.

1.12. Подсчитайте десятичные эквиваленты восьмеричного числа $54_{(8)}$, двоичного числа $101100_{(2)}$.

1.13. Переведите методом деления на основание новой системы счисления число $121_{(10)}$ в восьмеричную и двоичную системы счисления.

Проанализируйте результаты и попытайтесь предложить метод перевода восьмеричных чисел в двоичные и обратно, основанный на использовании двоичных эквивалентов восьмеричных цифр и не требующий выполнения каких-либо вычислений.

1.14. Переведите в двоичную систему следующие десятичные числа: 0,125; 0,749 (с точностью до 6 знаков в дробной части); 15,75.

1.15. В ЭВМ для записи в память двоичных целых чисел используются представления в форме с фиксированной запятой. При этом для хранения значения знака числа отводится один разряд. Определите диапазон представления целых чисел при их 16- и 32-разрядном представлении.

1.16. На рис. 1.11 приведены форматы для представления вещественных чисел. Для каждого из этих форматов определите диапазон и относительную погрешность представления вещественных чисел.

1.17. Поясните особенности кодирования и представления в ЭВМ штриховых, полутоновых и цветных изображений.

1.18. Чем характеризуется разрешающая способность устройств отображения графической информации?

1.19. Поясните основные отличия текстовых и графических режимов отображения информации на экранах дисплеев.

1.20. Подсчитайте объемы памяти регенерации изображения дисплея, необходимые для хранения: а) цветного графического изображения высокого разрешения (см. табл. 1.3); б) изображения текста в режиме среднего разрешения (см. рис. 1.9).

1.21. Поясните особенности кодирования и представления в ЭВМ звуковой информации: устной речи, музыки.

Глава 2
ОБРАБОТКА ИНФОРМАЦИИ В ЭВМ

2.1. АЛГОРИТМ И ЕГО СВОЙСТВА

Термин «алгоритм» своим происхождением обязан имени узбекского математика Аль-Хорезми, который еще в IX в. сформулировал правила выполнения четырех арифметических действий. Появившееся несколько позже слово «алгорифм» связано с именем древнегреческого математика Евклида, назвавшего так сформулированные им правила нахождения наибольшего общего делителя двух чисел. В современной математике употребляют термин *алгоритм*, под которым понимается конечная последовательность точно определенных действий, приводящих к решению поставленной задачи. Однако алгоритмы — это не только описания последовательностей решения различных задач. В форме различных инструкций и правил алгоритмы сопровождают человека во всей его жизни. Кулинарный рецепт типа «Взять того-то и того-то, перемешать, залить, довести до кипения, дать отстояться, процедить и т. д.» является типичным примером алгоритма. Алгоритмами являются и инструкции по включению и эксплуатации различных электробытовых приборов. Добираясь от дома до работы или места учебы, каждый из нас выполняет определенный алгоритм, составленный для себя, исходя из имеющихся возможностей, условий и накопленного опыта. В зависимости от времени выхода из дома, погодных условий, других обстоятельств этот алгоритм может допускать несколько вариантов маршрутов, совершение по пути пешей прогулки, использование того или иного вида транспорта и т. п.

Устанавливаемая алгоритмом последовательность действий задается в словесной или графической форме. При этом широко используются специально разработанные алгоритмические языки.

Примером словесной формы задания алгоритма служит алгоритм Евклида для нахождения наибольшего общего делителя двух чисел:

1. Обозревая два числа a и b, переходи к следующему пункту.

2. Сравни обозреваемые числа (a равно b, a меньше, больше b) и переходи к следующему пункту.

3. Если a и b равны, то прекрати вычисление: каждое из чисел дает искомый результат. Если числа не равны, то переходи к следующему пункту.

4. Если первое число меньше второго, то переставь их местами; переходи к следующему пункту.

5. Вычитай второе число из первого, обозревай два числа: вычитаемое и остаток; переходи к п. 2.

По указаниям этого алгоритма можно найти наибольший общий делитель для любой пары натуральных чисел. В этом проявляется одно из свойств алгоритма — *массовость*, которое означает, что если алгоритм разработан для решения определенной задачи, то он должен быть применим для решения задач этого типа при всех допустимых значениях исходных данных.

Следующие свойства алгоритма — дискретность и результативность — вытекают из определения алгоритма. *Дискретность* обусловливает дискретный (пошаговый) характер процесса получения результата, состоящий в последовательном выполнении конечного числа заданных алгоритмом *действий* или *команд*. *Результативность* — свойство алгоритма приводить к получению результата после выполнения над исходными данными заданной алгоритмом последовательности действий.

Важным свойством алгоритма является ориентированность его на определенного исполнителя. Это свойство, называемое *определенностью* (иногда точностью, понятностью), требует, чтобы каждая команда алгоритма (указание о выполняемом на очередном шаге действии) была понятна исполнителю, не оставляла бы места для ее неоднозначного понимания и неопределенного исполнения.

Если выполнить алгоритм Евклида для нахождения наибольшего общего делителя двух любых натуральных чисел, то легко убедиться в том, что для получения результата исполнителю алгоритма совершенно не обязательно знать, что такое наибольший общий делитель двух чисел и вообще, в чем суть решаемой задачи. Для получения результата необходимо лишь механическое выполнение заданной последовательности команд алго-

ритма, требующей от исполнителя в рассмотренном примере лишь умения выполнять «сравнение», «вычитание» и действие по «перестановке» двух чисел. Это замечательное свойство алгоритмов и позволяет использовать в качестве исполнителя алгоритма не только человека, но и специально разработанные для этих целей автоматические устройства, способные выполнять заданный перечень команд, называемый *системой команд исполнителя алгоритма.*

К таким устройствам относятся современные универсальные ЭВМ, специально разработанные человеком для целей автоматизированной обработки информации. Системы команд ЭВМ включают сотни команд на выполнение операций по обработке данных, управлению ходом вычислительного процесса, вводу-выводу данных, их запоминанию и других операций. Алгоритм, составленный из команд и представленный в форме, воспринимаемой ЭВМ, называется *программой ЭВМ.*

Таким образом, *ЭВМ является программно управляемым автоматом, обеспечивающим автоматическое выполнение команд программы.* В соответствии с принципом программного управления после выполнения очередной команды ЭВМ автоматически переходит к следующей команде, и так до тех пор, пока не последует команда прекратить вычислительный процесс.

2.2. ОСНОВНЫЕ УСТРОЙСТВА ЭВМ

Минимальный состав устройств ЭВМ выявляется при анализе процессов, связанных с представлением и автоматической переработкой информации по заданным алгоритмам. Для автоматического решения задач ЭВМ должна включать в себя следующие устройства: ввода и вывода данных, памяти, арифметико-логическое и управления. На рис. 2.1 изображена обобщенная структурная схема ЭВМ.

Основное назначение устройств ввода информации — преобразование информации, представленной в символах входного алфавита, в представление в символах внутреннего алфавита, в котором осуществляется обработка и хранение информации в ЭВМ. В качестве устройств ввода данных в ЭВМ используются клавиатуры, считыватели информации с бумажных перфокарт и перфолент, магнитных дисков и лент.

Для подготовки информации к вводу в ЭВМ служат устройства подготовки данных, основной функцией

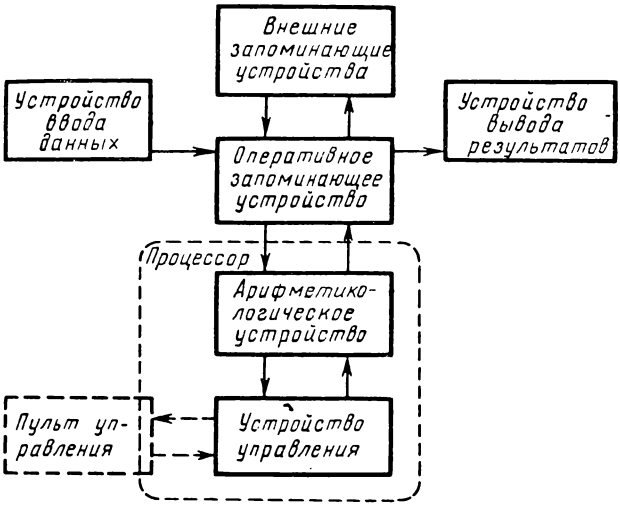


Рис. 2.1

которых является преобразование информации (текстовой, числовой, графической и пр.) в данные, записанные в символах входного алфавита ЭВМ. Наиболее распространены устройства подготовки данных на перфоносителях (перфокартах и перфолентах) и магнитных носителях (магнитной ленте, гибких магнитных дисках).

Память ЭВМ предназначена для хранения данных: входной информации, промежуточных и окончательных результатов, программ решения задач. Устройства памяти характеризуются емкостью и временем обращения.

Емкость указывает количество данных, которое может одновременно храниться в памяти. На практике емкость запоминающих устройств ЭВМ обычно изменяется в килобайтах и мегабайтах.

Время обращения — интервал времени между началом и окончанием ввода (вывода) информации в память (из памяти). В отношении устройств памяти вместо термина «ввод-вывод» используют термин «запись-чтение». Таким образом, время обращения характеризует затраты времени на поиск места в памяти и запись (чтение) информации.

Различают оперативные и внешние запоминающие устройства (ЗУ).

Оперативные запоминающие устройства (ОЗУ) характеризуются малым значением времени обращения, которое обычно составляет 10^{-7}—10^{-5} с (100 нс—10 мкс),

при емкости от сотен килобайтов до десятков мегабайтов.

Внешние запоминающие устройства (ВЗУ) характеризуются в сравнении с ОЗУ бо́льшим временем обращения (10^{-2} с и более), но емкость их может быть практически неограниченной, так как число блоков внешней памяти можно наращивать по мере необходимости.

В современных ЭВМ устройства оперативной памяти реализуются на быстродействующих электронных элементах. В качестве ВЗУ обычно используются запоминающие устройства на магнитных дисках и лентах.

С понятием «поиск места в памяти» связано понятие «адрес информации». Память ЭВМ можно представить состоящей из отдельных ячеек, в каждой из которых хранится определенное количество информации: 1 байт, 2 байт, 4 байт и т. д. Все ячейки в памяти пронумерованы. Номера ячеек называются *адресами*.

В современных ЭВМ минимальные адресуемые элементы памяти обычно имеют емкость байт. Если информация занимает в памяти несколько последовательных элементов, то она составляет *поле памяти*. Местонахождение информации в этом случае задается адресом первого байта и количеством занимаемых ею байтов памяти.

Различают запоминающие устройства произвольного и последовательного доступа. *Устройства произвольного доступа* позволяют по заданному адресу одинаково быстро извлекать информацию из любого элемента памяти. Такими устройствами являются используемые в ЭВМ оперативные ЗУ. Примером запоминающих *устройств последовательного доступа* являются ЗУ на магнитных лентах (МЛ). Данные на МЛ записываются байт за байтом по всей длине. Чтобы считать информацию, необходимо перематывать ленту до тех пор, пока участок с записью требуемой информации не будет подведен к считывающему устройству. Время обращения при этом зависит от того, как «далеко» на ленте находится требуемая информация.

Арифметико-логическое устройство (АЛУ) выполняет арифметические и логические операции, предусмотренные системой команд ЭВМ, и характеризуется:

— набором элементарных операций, которые АЛУ выполняет;

— временем выполнения элементарных операций;

— *средним быстродействием*, т. е. количеством арифметических и логических операций, выполняемых в единицу времени (секунду). Быстродействие АЛУ современ-

ных ЭВМ изменяется в довольно широких пределах — от сотен тысяч до десятков миллионов операций в секунду.

Арифметико-логическое устройство выполняет такие элементарные действия, как сложение, вычитание, умножение, деление, сравнение, логические операции.

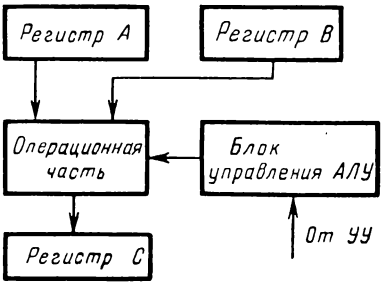


Рис. 2.2

На рис. 2.2 приведена в общем виде структурная схема АЛУ ЭВМ. В состав ее входят регистры *РгА*, *РгВ* и *РгС*. *Регистры*, как и ячейки ЗУ, являются элементами памяти. Перед выполнением операции исходные данные *А* и *В* считываются из ОЗУ ЭВМ и помещаются на временное хранение в регистры *РгА* и *РгВ*. Из устройства управления ЭВМ в блок управления *АЛУ* поступает команда на выполнение заданной операции. Операционная часть *АЛУ*, получив необходимые команды от блока управления, выполняет заданную операцию над исходными данными *А* и *В* и результат засылает на временное хранение в регистр результата *РгС*. Затем, если это требуется, результат из *РгС* помещается для хранения в ячейку памяти ОЗУ.

Быстродействие ЭВМ связано с временем обращения ОЗУ. Так, в ЭВМ с быстродействием 200 тыс. оп./с одна операция выполняется за 5 мкс, в течение которых необходимо не только выполнить операцию, но и трижды обратиться к оперативной памяти: считать два операнда и записать результат. Для обеспечения заданного быстродействия время обращения для ОЗУ данной ЭВМ должно иметь порядок 1 мкс (10^{-6} с).

Устройство управления (УУ) предназначено для автоматического управления ходом вычислительного процесса и обеспечивает необходимое взаимодействие всех устройств машины по автоматическому выполнению заданного алгоритма решения задачи. Функции УУ, как наиболее важные для понимания автоматической работы ЭВМ, описываются далее при рассмотрении реализации в ЭВМ принципа программного управления.

Устройства вывода предназначены для вывода информации из машины и преобразования результатов решения задач из символов внутреннего алфавита в выходную информацию, представленную в символах выходного алфавита.

В качестве устройств вывода ЭВМ используют печатающие устройства, графопостроители, устройства вывода информации на перфокарты и перфоленты, алфавитно-цифровые и графические дисплеи, различные звукогенерирующие устройства, позволяющие выводить из ЭВМ звуковую информацию от простейших звуковых сигналов типа «щелчок» до более сложных: музыкальных мелодий, искусственно синтезированной речи и т. д. В качестве устройства вывода ЭВМ можно рассматривать и накопители информации на магнитных лентах и гибких магнитных дисках.

Печатающие, графические, отображающие устройства, как правило, выдают информацию в виде, удобном для человека. Устройства, фиксирующие информацию на магнитных лентах и перфоносителях, используются для выдачи данных, которые в последующем предполагается обрабатывать на ЭВМ или использовать для управления какими-либо автоматами (например, задания для станков с числовым программным управлением). Специализированные ЭВМ формируют на выходе электрические сигналы, управляющие работой различных исполнительных устройств (механизмов), и выдают другую информацию, форма представления которой определяется характером управляемой системы.

Рассмотренную структурную схему ЭВМ (см. рис. 2.1) можно назвать классической. Она получила название компьютера фон Неймана по имени американского математика, предложившего такую структуру машины для автоматической обработки данных. В современных ЭВМ комплекс устройств, охватывающий АЛУ и УУ, принято называть *процессором*.

Скорость работы внешних устройств ЭВМ, включающих устройства ввода и вывода информации, значительно ниже скорости работы процессора. Поэтому для более эффективного использования возможностей процессора к нему через специальные устройства, называемые *каналами*, подключают несколько одновременно обслуживаемых внешних устройств. Такая схема связи устройств, характерная, например, для ЭВМ Единой серии (ЕС ЭВМ), представлена на рис. 2.3, где блоки *ВУ* обозначают внешние устройства, а блоки *ГУ* — устройства группового управления подключенными к ним однотипными *ВУ*. Каналы, через которые осуществляется ввод-вывод информации, по существу представляют собой специализированные процессоры для выполнения операций ввода-вывода. Структуры связи устройств

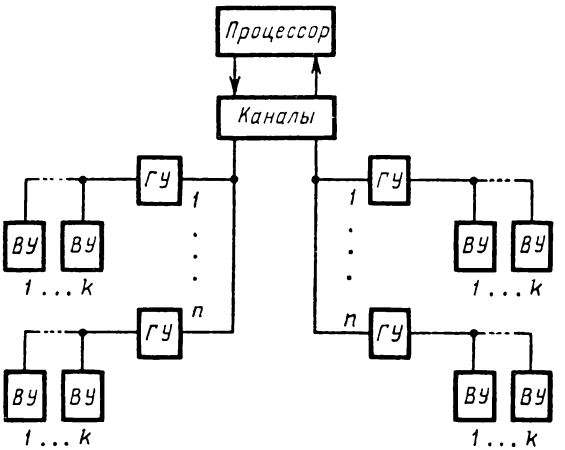


Рис. 2.3

ЭВМ, подобные изображенной на рис. 2.3, можно лишь условно называть электронными вычислительными машинами. По существу, это уже не машины, а сложные вычислительные системы, которые могут включать в свой состав несколько обрабатывающих процессоров, обеспечивать возможность обмена информацией с другими ЭВМ, выполнять передаваемые по каналам связи задания удаленных пользователей и другие сложные функции по обработке данных.

2.3. ВЫПОЛНЕНИЕ В ЭВМ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ

Как указывалось ранее, преимущественное использование в ЭВМ двоичной системы счисления объясняется простотой выполнения арифметических действий. Таблицы сложения, вычитания и умножения в двоичной системе счисления представляются следующим образом:

сложение				вычитание	умножение
0 + 0 =		0		0 − 0 = 0	0 × 0 = 0
0 + 1 =		1		1 − 0 = 1	0 × 1 = 0
1 + 0 =		1		1 − 1 = 0	1 × 0 = 0
1 + 1 =	[1]	0	[1]	0 − 1 = 1	1 × 1 = 1
	↓ перенос в старший разряд		↓ заем в старшем разряде		

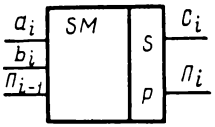


Рис. 2.4

Правила поразрядных действий при сложении чисел A и B приведены в табл. 2.1. В общем виде их можно записать так:

$$a_i + b_i + П_{i-1} = c_i + П_i,$$

где a_i и b_i — значения i-х разрядов операндов* A и B соответственно; $П_{i-1}$ — перенос из $(i-1)$-го разряда; c_i — i-й разряд суммы; $П_i$ — перенос из i-го разряда в $(i+1)$-й разряд.

Устройство, выполняющее арифметические действия по правилам, указанным в табл. 2.1, называется *двоичным сумматором*. На рис. 2.4 приведено условное обозначение двоичного сумматора, имеющего три входа, на которые подаются значения i-х разрядов операндов a_i и b_i и значение переноса из $(i-1)$-го разряда $П_{i-1}$, и два выхода: S, с которого получается значение i-го разряда суммы c_i; P, с которого получается значение переноса из i-го разряда в $(i+1)$-й разряд $П_i$.

Таблица 2.1

a_i	b_i	$П_{i-1}$	c_i	$П_i$	a_i	b_i	$П_{i-1}$	c_i	$П_i$
0	0	0	0	0	0	0	1	1	0
0	1	0	1	0	0	1	1	0	1
1	0	0	1	0	1	0	1	0	1
1	1	0	0	1	1	1	1	1	1

Двоичный сумматор является основой арифметического устройства любой ЭВМ, он позволяет реализовать алгоритмы выполнения всех арифметических операций. В частности, одним из способов выполнения операции вычитания с помощью двоичного сумматора является замена знака вычитаемого на противоположный и прибавление его к уменьшаемому:

$$A - B = A + (-B).$$

Таким образом, операцию арифметического вычитания заменяют операцией алгебраического сложения, которая и является основной операцией двоичного сум-

*Под операндом понимается число, участвующее в операции, выполняемой ЭВМ.

матора. Причем для представления отрицательных чисел используется так называемый *дополнительный код*, образуемый в соответствии с правилом:

$$A_д = \begin{cases} A, & \text{если } A \geqslant 0; \\ q^{k+1} - A, & \text{если } A < 0, \end{cases} \qquad (2.1)$$

где q — основание системы счисления; k — длина разрядной сетки целой части числа.

Использование дополнительных кодов чисел для выполнения операции алгебраического сложения базируется на следующей теореме: *сумма дополнительных кодов чисел есть дополнительный код результата.*

В справедливости приведенной теоремы убедимся на примерах, которые для наглядности выполним с десятичными целыми числами. Результатом алгебраического сложения $(k+1)$-разрядных дополнительных кодов складываемых чисел будем считать получаемое $(k+1)$-разрядное значение дополнительного кода суммы. Перенос из $(k+1)$-го разряда в $(k+2)$-й разряд не учитывается.

Пример 2.1. Найти алгебраическую сумму чисел

$A = -926$ и $B = 254$.

Решение. Произведем следующие операции:

$A_д = 10^4 - 926 = 9074$;

$B_д = 254$;

$C_д = 254 + 9074 = 9328$.

Полученное значение $C_д$ является дополнительным кодом числа -672, являющегося алгебраической суммой заданных значений A и B.

Ответ: -672.

Пример 2.2. Найти алгебраическую сумму чисел

$A = -524$ и $B = 642$.

Решение. Произведем следующие операции:

$A_д = 10^4 - 524 = 9476$;

$B_д = 642$;

$C_д = 9476 + 642 = \boxed{1}\ 0118 = 118.$

↓ перенос в $(k+2)$-й разряд теряется

Ответ: 118.

Рассмотрим особенности выполнения арифметических

операций над машинными представлениями числовых данных в двоичной системе счисления в формах с фиксированной и плавающей запятой.

Сложение чисел, представленных в форме с фиксированной запятой. Рассмотрим примеры выполнения операции сложения над целыми числами, представленными в форме с фиксированной запятой в двоичной системе счисления. Будем считать, что для представления складываемых чисел используется 8-разрядная сетка; один разряд знаковый и семь разрядов значащих.

Прежде чем приступить к рассмотрению примеров, укажем широко используемый в вычислительных машинах простой алгоритм получения дополнительного кода отрицательных двоичных чисел:

Алгоритм 2.1

1. В двоичном представлении числа заменить в значащих разрядах единицы нулями, а нули единицами. Знаковый разряд оставить без изменения.

2. Добавить к полученному изображению единицу младшего разряда.

Убедимся в правильности приведенного алгоритма. Получим дополнительный код числа $A = -101$. Образуем машинные представления числа A в форме с фиксированной запятой:

$$[A] = \boxed{1}\boxed{0\ 0\ 0\ 0\ 1\ 0\ 1}$$

Действуя в соответствии с приведенным выше алгоритмом, получим

$$[A]_д = \boxed{1}\boxed{1\ 1\ 1\ 1\ 1\ 0\ 1\ 0} + 1 = \boxed{1}\boxed{1\ 1\ 1\ 1\ 1\ 0\ 1\ 1}.$$

Действуя в соответствии с выражением (2.1), получим

$$A_д = 2^8 - 101;$$

$$\begin{array}{r} 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\ -\qquad\qquad 1\ 0\ 1 \\ \hline 1\ 1\ 1\ 1\ 1\ 0\ 1\ 1 \end{array}$$

$$[A]_д = \boxed{1}\boxed{1\ 1\ 1\ 1\ 0\ 1\ 1}$$

Сравнивая полученные результаты, убеждаемся в справедливости приведенного алгоритма.

Очевиден алгоритм и обратного преобразования отрицательного числа из представления в дополнительном

коде в обычное представление, называемое *прямым кодом числа.*

Алгоритм 2.2

1. Вычесть единицу из младшего разряда представления числа в дополнительном коде.

2. Заменить в значащей части представления числа единицы нулями, а нули единицами. Знаковый разряд оставить без изменения.

Пример 2.3. Найти алгебраическую сумму для пяти вариантов двоичных чисел:

1) $A=101$, $B=111$ $(A>0,\ B>0)$

Решение:

$$+\begin{array}{ll}[A]_д=\boxed{0}\boxed{0\,0\,0\,0\,1\,0\,1} & (5)\\ [B]_д=\boxed{0}\boxed{0\,0\,0\,0\,1\,1\,1} & (+7)\\ \hline [C]_д=\boxed{0}\boxed{0\,0\,0\,1\,1\,0\,0} & +12\end{array}$$

Ответ: 1100.

2) $A=-101$, $B=-111$ $(A<0,\ B<0)$

Решение:

$$+\begin{array}{ll}[A]_д=\boxed{1}\boxed{1\,1\,1\,1\,0\,1\,1} & (-5)\\ [B]_д=\boxed{1}\boxed{1\,1\,1\,1\,0\,0\,1} & (-7)\\ \hline [C]_д=\boxed{1}\boxed{1\,1\,1\,0\,1\,0\,0} & -12\end{array}$$

Преобразование результата в прямой код

$$-1$$

$$\boxed{1}\boxed{1\,1\,1\,0\,0\,1\,1}$$

$$[C]=\boxed{1}\boxed{0\,0\,0\,1\,1\,0\,0} \qquad (-12)$$

Ответ: -1100.

3) $A=101$, $B=-111$ $(A>0,\ B<0,\ |A|<|B|)$

Решение:

$$+\begin{array}{ll}[A]_д=\boxed{0}\boxed{0\,0\,0\,0\,1\,0\,1} & (+5)\\ [B]_д=\boxed{1}\boxed{1\,1\,1\,1\,0\,0\,1} & (-7)\\ \hline [C]_д=\boxed{1}\boxed{1\,1\,1\,1\,1\,1\,0} & (-2)\end{array}$$

$$-1$$

$$\boxed{1}\boxed{1\,1\,1\,1\,1\,0\,1}$$

$$[C]=\boxed{1}\boxed{0\,0\,0\,0\,0\,1\,0} \qquad (-2)$$

Ответ: -10.

4) $A=111$, $B=-101$ $(A>0,\ B<0,\ |A|>|B|)$

Решение:

$$
\begin{array}{rll}
+[A]_\partial = & \boxed{0}\boxed{0\,0\,0\,0\,1\,1\,1} & +(+7) \\
+[B]_\partial = & \boxed{1}\boxed{1\,1\,1\,1\,0\,1\,1} & +(-5) \\
\hline
[C]_\partial = & \boxed{0}\boxed{0\,0\,0\,0\,0\,1\,0} & (+2)
\end{array}
$$

перенос в
9-й разряд
теряется

Ответ: 10.

5) $A = 1000000$, $B = 1000011$ $(A > 0,\ B > 0,\ (A+B) > (2^7 - 1))$

Решение:

$$
\begin{array}{rll}
+[A]_\partial = & \boxed{0}\boxed{0\,1\,0\,0\,0\,0\,0} & +(+64) \\
+[B]_\partial = & \boxed{0}\boxed{0\,1\,0\,0\,0\,1\,1} & +(+67) \\
\hline
[C]_\partial = & \boxed{1}\boxed{1\,0\,0\,0\,0\,1\,1} & (-125)(!)
\end{array}
$$

$$
\begin{array}{rll}
 & -1 & \\
 & \boxed{1}\boxed{1\,0\,0\,0\,0\,1\,0} & \\
[C] = & \boxed{1}\boxed{1\,1\,1\,1\,1\,0\,1} & (-125)(!)
\end{array}
$$

Ответ: -1111101 (неправильный).

Выполнение примера в варианте 5 завершилось неправильно. Получилось значение результата, равное (-125) вместо $(+131)$. Причиной является переполнение разрядной сетки, так как сумма операндов превосходит диапазон представления 7-разрядных двоичных чисел, равный $127 = (2^7 - 1)$. Переполнение может происходить лишь при сложении операндов, имеющих одинаковые знаки. Признаком переполнения является знак результата, противоположный знакам операндов.

Рассмотренные в примере 2.3 варианты охватывают все возможные сочетания значений операндов и позволяют сформулировать обобщенный алгоритм сложения двоичных чисел в формате представления с фиксированной запятой, автоматически реализуемый в АЛУ ЭВМ. При этом операционная часть АЛУ (см. рис. 2.2) выполняет преобразование операндов в дополнительный код и обратно и суммирование их значений.

Определяемая в каждом конкретном случае последовательность действий задается блоком управления АЛУ, в который подаются сигналы от специальных логических электронных схем, выполняющих анализ знаков операндов и их сравнение по абсолютной величине. Для обнаружения переполнений в состав арифметического устройства ЭВМ включается специальный блок, который вырабатывает признак переполнения.

Сложение чисел, представленных в форме с плавающей запятой. Как было указано ранее, числа, представленные в форме с плавающей запятой, изображаются двумя частями — мантиссой и порядком. При операции алгебраического сложения действия, выполняемые над мантиссами и порядками, различны. Следовательно, в арифметическом устройстве ЭВМ должны быть два раздельных устройства: для обработки мантисс и для обработки порядков.

Рассмотрим наиболее общий случай сложения чисел, когда их порядки не равны друг другу, т. е. $p_A \neq p_B$. Для операции сложения чисел необходимым условием является соответствие разрядов операндов друг другу. Значит, прежде всего следует уравнять порядки, что естественно, повлечет за собой временное нарушение нормализации одного из слагаемых. Выравнивание порядков означает, что порядок меньшего числа надо увеличить на величину $\Delta p = |p_A - p_B|$ и сдвинуть мантиссу меньшего числа вправо на Δp разрядов.

Операция сдвига для чисел:

а) положительных;

б) отрицательных, представленных в прямом коде, выполняется по следующим правилам:

Исходная комбинация	Комбинация, сдвинутая влево на один разряд	Комбинация, сдвинутая вправо на один разряд
а) $0, a_1a_2...a_n$	$0, a_2...a_n0$	$0, 0\ a_1...a_{n-1}$
б) $1, a_1a_2...a_n$	$1, a_2...a_n0$	$1, 0\ a_1...a_{n-1}$

Чтобы выравнять порядки, АЛУ должно самостоятельно определить, какой из двух операндов меньший. На это указывает знак разности $p_A - p_B$: положительный знак будет при $p_A \geqslant p_B$, а отрицательный — при $p_A < p_B$.

Сложение мантисс осуществляется на двоичном сумматоре дополнительного кода по правилам чисел, представленных в форме с фиксированной запятой. Если после сложения мантисса результата удовлетворяет условию нормализации (1.10), то к этому результату приписывается порядок любого из операндов. В противном случае проводится нормализация числа.

Нормализация числа состоит из двух действий — проверки выполнения условия (1.8) и сдвига изображения мантиссы в ту или иную сторону. Сдвиги осуществляют-

ся на один разряд и более в левую или правую сторону в пределах разрядной сетки машины. Каждая операция сдвига сопровождается изменением значения порядка числа. При сдвигах влево значение порядка увеличивается на столько единиц, на сколько разрядов сдвигается мантисса. При сдвиге вправо значение порядка увеличивается на соответствующее число единиц.

Приведем примеры формального выполнения реализуемого АЛУ ЭВМ алгоритма сложения двоичных чисел в форме представления с плавающей запятой.

Пример 2.4. Сложить числа $A=0{,}1011\cdot 2^{-2}$ и $B=-0{,}1001\cdot 2^{-3}$. Мантисса и порядок обрабатываются на сумматорах дополнительного кода (пять двоичных разрядов для мантиссы и четыре двоичных разряда для порядка).

Решение. Прежде всего запишем машинные изображения чисел:

$[m_A]=0{,}1011$; $[p_A]=1{,}010$;

$[m_B]=1{,}1001$; $[p_B]=1{,}011$.

Определим, какой из двух порядков больше:

$[\Delta p]_\partial=[p_A]_\partial-[p_B]_\partial$;

$[\Delta p]_\partial=[p_A]_\partial+[-p_B]_\partial$;

$$\begin{array}{rr} [p_A]_\partial= & 1{,}110 \\ [-p_B]_\partial= & + \ 0{,}011 \\ \hline [\Delta p]_\partial= & 0{,}001 \end{array}$$

Так как величина Δp положительна, то $p_A>p_B$. Следовательно, надо сдвинуть мантиссу числа B вправо на Δp разрядов, т. е. на один разряд:

$[\vec{m}_B]=1{,}0100$ (стрелка над символом m_B показывает сдвиг в соответствующую сторону).

Образуем дополнительные коды мантисс и сложим их:

$$\begin{array}{rr} [m_A]_\partial= & 0{,}1011 \\ [\vec{m}_B]_\partial= & + \ 1{,}1100 \\ \hline [m_C]_\partial= & 0{,}0111 \end{array}$$

Сдвигом влево на один разряд осуществляем нормализацию мантиссы результата и соответствующим образом корректирум порядок:

$[\overleftarrow{m}_C]_\partial=0{,}1110$;

$[p_C]=[p_A]-1$;

$[p_C]_\partial=[p_A]_\partial+[-1]_\partial$;

$[p_A]_д = 1{,}110$

$+$

$[-1]_д = 1{,}111$

$[p_C]_д = 1{,}101$

-1

$1{,}100$

$[p_C] = 1{,}011$

Преобразование представления порядка из дополнительного кода в прямой по алгоритму 2.2

Ответ: $C = 0{,}1110 \cdot 2^{-3}$.

Сложение чисел с одинаковыми порядками и знаками всегда приводит к переполнению разрядной сетки сумматора мантисс. Для получения результата в этом случае необходимо сдвинуть значение мантиссы результата вправо на один разряд и увеличить порядок результата на единицу. При таком сдвиге знаковый разряд мантиссы результата должен стать старшим разрядом ее значения, а знаковый разряд результата должен получить значение, соответствующее знаку складываемых чисел.

Пример 2.5. Найти сумму чисел $A = -0{,}1100 \cdot 2^4$ и $B = -0{,}1000 \cdot 2^4$.

Рассмотрим этот пример для сумматора дополнительного кода (пять двоичных разрядов для мантиссы и четыре разряда для порядка).

Решение. Произведем следующие действия:

$[m_A]_д = 1{,}0100$ $\qquad$ $[p_A]_д = 0{,}100;$

$+$

$[m_B]_д = 1{,}1000$ $\qquad$ $[p_B]_д = 0{,}100;$

$[m_C]_д = 0{,}1100$ $\qquad$ $[p_C]_д = 0{,}100.$

Знак мантиссы результата отличен от знаков складываемых мантисс операндов, что свидетельствует о переполнении разрядной сетки.

Проведем корректировку значений мантиссы и порядка результата по правилам, изложенным выше:

$[m'_C]_д = 1{,}0110$ $\qquad$ $[p'_C] = 0{,}100 + 0{,}001 = 0{,}101;$

-1

$1{,}0101$

$[m'_C] = 1{,}1010.$

Ответ: $C = -0{,}1010 \cdot 2^5$.

Умножение двоичных чисел. Для умножения чисел

в двоичной системе счисления используется известный «школьный» алгоритм умножения в столбик:

```
    1101    — Множимое
  ×
    1101    — Множитель
  ------
    1101
   0000
+  1101     — Частные произведения
  1101
  ------
10101001    — Произведение
```

Из примера видно, что операция умножения состоит из ряда последовательных операций сложения частных произведений. Операциями сложения управляют разряды множителя: если в каком-либо разряде множителя находится единица, то к сумме частных произведений добавляется множимое с соответствующим сдвигом. Если в разряде множителя находится нуль, то множимое не прибавляется. В рассмотренном примере умножение выполнено привычным методом — начиная с младших разрядов множителя. В вычислительных машинах часто используется метод, при котором умножение начинается со старших разрядов множителя:

```
    1101    — Множимое
  ×
    1101    — Множитель
  ------
    1101
+   1101    — Частные произведения
      1101
  ------
10101001    — Произведение
```

В обоих методах получения произведения кроме операции сложения выполняется операция сдвига чисел. При этом возможен сдвиг множимого или суммы частных произведений.

Для реализации операции умножения необходимы сумматор, регистры для хранения множимого и схема анализа разрядов множителя. Сумматор и регистры должны обеспечивать сдвиг содержимого в ту или иную сторону в соответствии с принятым методом умножения.

При умножении двух величин число цифр произведения может превышать число цифр сомножителей в два раза. Поэтому в схемах умножения ЭВМ для получения

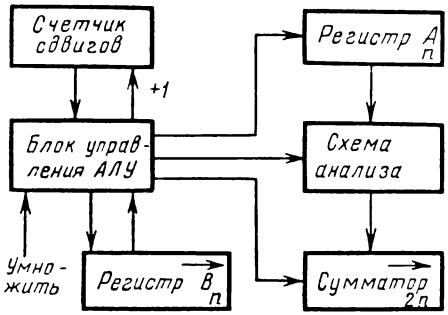


Рис. 2.5

произведения используют сумматоры, а при необходимости и регистры с удвоенным числом разрядов.

На рис. 2.5 приведена структурная схема множительного устройства, реализующего метод умножения начиная с младших разрядов множителя со сдвигом содержимого сумматора вправо. Блок управления АЛУ осуществляет автоматическое выполнение следующего алгоритма умножения двух чисел:

1. Очистить сумматор, т. е. заслать в него число 0. Установить счетчик сдвигов в 0.

2. Если младший разряд регистра *B* равен 0, то перейти к шагу 3, если — 1, то прибавить содержимое регистра *A* к сумматору.

3. Сдвинуть содержимое сумматора на один разряд вправо.

4. Сдвинуть содержимое регистра *B* на один разряд вправо.

5. Счетчик сдвигов равен *n*? Если НЕТ, то увеличить счетчик сдвигов на 1 и перейти к шагу 2. Если ДА, то закончить умножение. В сумматоре находится произведение чисел *A* и *B*.

Знак произведения в множительных устройствах ЭВМ можно получить за счет сложения знаков множителя и множимого на двоичном сумматоре.

Используя табл. 2.1, можно убедиться в том, что получаемое на сумматоре значение суммы двух слагаемых дает правильные значения кода знака произведения при суммировании кодов знаков сомножителей:

$1+1=0;\quad (-)\cdot(-)=(+);$
$1+0=1;\quad (-)\cdot(+)=(-);$
$0+1=1;\quad (+)\cdot(-)=(-);$
$0+0=0;\quad (+)\cdot(+)=(+).$

При выполнении указанной операции переносы между разрядами не используются. Указанная операция имеет

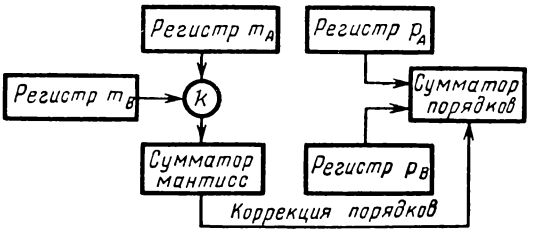


Рис. 2.6

специальное название — сложение по модулю 2 и обозначается символом $\oplus$. Таким образом, код знака произведения S_C получается в результате выполнения операции:

$$S_C = S_A \oplus S_B,$$

где S_A и S_B — коды знаков сомножителей A и B.

Для чисел, представленных в форме с плавающей запятой, при операции умножения действия, выполняемые над мантиссой и порядками, различны: мантиссы перемножаются, порядки складываются. Очевидно, что результат умножения может получиться ненормализованным. Тогда потребуется нормализация с соответствующей коррекцией порядка результата. Структурная схема множительного устройства для чисел, представленных в форме с плавающей запятой, приведена на рис. 2.6.

Деление двоичных чисел. Во многом деление двоичных чисел аналогично делению десятичных чисел. Процесс деления состоит в том, что последовательно (разряд за разрядом) находятся цифры частного путем подбора, при этом каждая цифра умножается на делитель и затем полученное произведение вычитается из делимого.

В общем случае «школьный» алгоритм деления на примере двоичных чисел представляется следующим образом:

```
Делимое   1100100 | 1010   — Делитель
        –  1010   | 1010   — Частное
          -------
Остаток   00101
        –  1010
          -------
          –0101
        +   1010   } — Восстановление остатка
          -------
          01010
        –  1010
          -------
           0000
```

Цифры частного получаются последовательно начиная со старшего разряда путем вычитания делителя из остатка. Положительному остатку соответствует цифра частного **1**; отрицательному остатку — цифра частного **0**, при этом прибавлением делителя восстанавливается предыдущий положительный остаток.

Если остаток положительный, то для получения следующей цифры частного его сдвигают влево на один разряд (либо делитель вправо на один разряд) и из него вычитают делитель и т. д.

Если остаток отрицательный, то прибавлением к отрицательному остатку делителя восстанавливается предыдущий положительный остаток, который сдвигается на один разряд влево (либо сдвигается делитель вправо на один разряд) и из него вычитается делитель. Описанный алгоритм деления называется *алгоритмом деления с восстановлением остатка.*

Для хранения делителя, делимого и накопления очередных разрядов частного выделяются регистры. Минимальный состав блоков, необходимый для выполнения деления, показан на рис. 2.7. Схема предусматривает сдвиг содержимого сумматора и передачу делителя в сумматор без сдвига. Стрелки на рисунке указывают направление сдвига. Параметр n указывает ориентировочное число разрядов в цифровой части сумматора и регистра делителя. Знак частного, как и при выполнении операции умножения, образуется отдельно от цифровой части путем сложения знаковых разрядов делимого и делителя по модулю 2. При делителе, равном 0, вырабатывается сигнал переполнения. Сигнал переполнения вырабатывается и в тех случаях, когда делимое больше делителя по абсолютной величине для чисел с фиксированной запятой.

Для получения частного от деления двух чисел, пред-

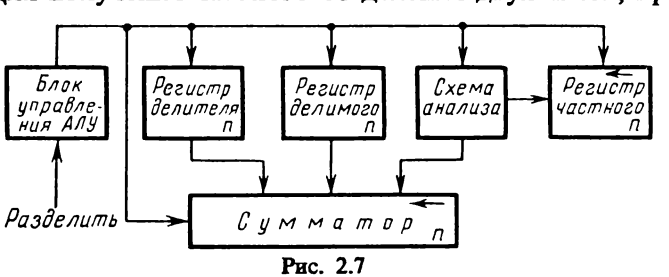


Рис. 2.7

ставленных в форме с плавающей запятой, необходимо выполнить следующие действия:

$m_C = m_A / m_B$; $p_C = p_A - p_B$.

Так как мантиссы делимого и делителя являются нормализованными числами, при делении возможны два случая, когда

1) $|m_A| \geqslant |m_B|$; 2) $|m_A| < |m_B|$.

Если мантисса делимого **больше или равна** мантиссе делителя, то алгоритм деления начинается с операции вычитания делителя из делимого и записи 1 в целую часть частного. Все остальные действия над мантиссами аналогичны действиям над числами, представленными в форме с фиксированной запятой.

Если мантисса делимого **меньше** мантиссы делителя, то после операции вычитания на шаге 1 получится отрицательный остаток, что означает наличие 0 в целой части мантиссы частного и продолжение алгоритма деления по правилам для чисел, представленных в форме с фиксированной запятой.

Так как при операции деления порядки чисел складываются, возможно переполнение разрядной сетки в сумматоре порядков. При переполнении в сторону отрицательных величин мантисса результата превращается в машинный нуль, а порядку присваивается наибольшее отрицательное значение. Если делитель равен нулю, необходима также выработка сигнала переполнения.

Вопросы для самопроверки

2.1. Что такое алгоритм?

2.2. Назовите и поясните основные свойства алгоритма.

2.3. Почему исполнителем алгоритма может быть автомат?

2.4. Что такое программа ЭВМ? В каком соотношении находятся понятия «алгоритм» и «программа»?

2.5. Перечислите основные устройства, входящие в состав ЭВМ, и охарактеризуйте их функции.

2.6. Назовите основные характеристики запоминающих устройств.

2.7. Назовите типы запоминающих устройств. В чем их отличие?

2.8. Объясните, как адресуется информация в памяти ЭВМ.

2.9. Что такое регистр? Объясните назначение регистров *РгА*, *РгВ* и *РгС* в структурной схеме АЛУ (рис. 2.2).

2.10. Перечислите основные типы устройств ввода-вывода.

2.11. Напишите изображения чисел $A = -0{,}101010$ и $B = 0{,}100010$ в прямом и дополнительном кодах. Используя сложение в дополнительном коде, найдите $A + B$, $A - B$.

2.12. Возможно ли переполнение разрядной сетки, если числа с плавающей запятой складываются, умножаются, делятся?

Глава 3
ПРОГРАММИРОВАНИЕ

3.1. ПРИНЦИП ПРОГРАММНОГО УПРАВЛЕНИЯ

Функционирование ЭВМ осуществляется на основе принципа программного управления, суть которого заключается в следующем. Программа, представляющая собой последовательность команд, реализующих алгоритм решения задачи, вводится в память ЭВМ, после чего начинается ее автоматическое выполнение с первой команды. После каждой выполненной команды машина автоматически переходит к выполнению следующей команды, и так до тех пор, пока не встретится команда, предписывающая закончить вычисления.

Структура команды ЭВМ в простейшем случае включает в себя две части (рис. 3.1): операционную и адресную. Операционная часть содержит код операции — КОп (сложить, вычесть, ...). Адресная часть содержит адреса ячеек памяти (*A1, A2* ...); в них хранятся значения операндов, с которыми надо выполнить заданную операцию. В зависимости от числа адресов, указанных в команде, различают одно-, двух-, трехадресные команды (рис. 3.1, *а, б, в* соответственно).

Команда может храниться в памяти ЭВМ, как обычное слово длиной 2, 3, 4 байт и т. д. Длина команды в байтах обычно зависит от структуры и числа разрядов, отведенных под адресную и операционную части. Восемь двоичных разрядов, отведенных под код операции, позволяют закодировать $2^8 = 256$ различных ко-

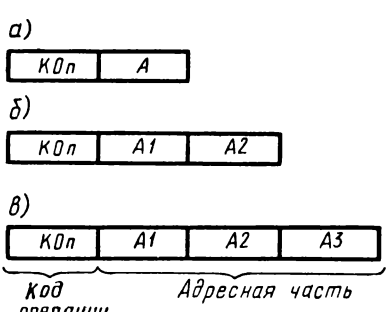


Рис. 3.1

манд, что обычно является достаточным. Длина адресной части команды зависит от числа адресов и количества разрядов, отведенных для одного адреса, которое зависит от емкости адресуемой в команде памяти ЭВМ. Так, например, чтобы иметь возможность указать в команде адрес любой ячейки оперативной памяти емкостью 64 Кбайт, необходимо под адрес отвести 16 двоичных разрядов ($2^{16}=64$ К).

Рассмотрим далее пример подготовки простейшей программы для ЭВМ, а затем — взаимодействие устройств ЭВМ по выполнению программы. Пусть требуется вычислить выражение $y=cx^2+b$. Алгоритм решения задачи:

1. Ввести в оперативную память значения исходных переменных c, x, b.
2. Вычислить x^2. Присвоить переменной y значение полученного результата.
3. Вычислить yc. Присвоить переменной y значение полученного результата.
4. Вычислить $y+b$. Присвоить переменной y значение результата.
5. Напечатать значение y.

Для составления программы решения необходимо, чтобы система команд ЭВМ включала следующие команды: ввести, сложить, умножить, запомнить, напечатать, остановить.

Будем считать, что команды используемой ЭВМ двухадресные и в общем виде записываются так:

Ka_1a_2,

где K — код операции; a_1, a_2 — значения первого и второго адреса команды (в некоторых командах a_1 и a_2 могут быть не только адресами операндов, но и операндами).

Примем, что коды операции представляются двузначными числами и имеют следующие значения:

01 — ввести с перфокарт,	04 — запомнить,
02 — сложить,	05 — напечатать,
03 — умножить,	06 — остановить.

Уточним особенности выполнения каждой команды и использования их адресных частей:

ВВЕСТИ — вводятся с перфокарт a_1 чисел и запоминаются в поле оперативной памяти, начинающемся с ячейки a_2.

СЛОЖИТЬ — складываются два операнда, один из

которых хранится в ячейке оперативной памяти с адресом a_1, второй — в ячейке с адресом a_2. Результат остается в сумматоре АЛУ.

УМНОЖИТЬ — умножаются два операнда, один из которых хранится в ячейке оперативной памяти с адресом a_1, второй — в ячейке с адресом a_2. Результат остается в сумматоре АЛУ.

ЗАПОМНИТЬ — записывается значение, хранящееся в сумматоре АЛУ, в ячейку с адресом a_1. Адрес a_2 в данной команде не используется.

НАПЕЧАТАТЬ — из поля оперативной памяти, начинающегося с адреса a_2, на печать выводятся значения хранимых в памяти a_1 чисел.

ОСТАНОВИТЬ — прекращается выполнение программы. Значения адресов a_1, a_2 не используются.

Будем считать, что в перечисленных командах адреса a_1 и a_2 изменяются в пределах 00—99, а адресуемая ячейка оперативной памяти хранит значение одной переменной или команду программы.

Составление программы. Сначала распределяется память. В рассматриваемом примере вычисляемое выражение содержит четыре переменных c, b, x, y, для хранения значений которых отводится четыре ячейки с номерами 01, 02, 03, 04 соответственно.

Запишем команды программы в соответствии с приведенным выше алгоритмом:

1) 01 03 01 — ввести с перфокарт ($K=01$) три значения ($a_1=03$) в поле оперативной памяти (ОП), начинающееся с ячейки 01 ($a_2=01$). Предполагается, что переменные в соответствии с принятым распределением памяти вводятся в порядке: c, b, x;

2) 03 03 03 — умножить ($K=03$) число, хранящееся в ячейке ОП с адресом 03 ($a_1=03$), на число, хранящееся в ячейке ОП с адресом 03 ($a_2=03$). Результат, равный значению x^2, сохраняется в сумматоре АЛУ;

3) 04 04 00 — запомнить ($K=04$) в ячейке ОП с адресом 04 ($a_1=04$) значение результата

предыдущей операции. Переменная y получает текущее значение, равное x^2;

4) 03 01 04 — умножить ($K=03$) два числа, хранящихся в ячейках ОП с адресами 01 и 04 ($a_1=01$, $a_2=04$). В результате находится произведение yc или x^2c, так как $y=x^2$;

5) 04 04 00 — запомнить ($K=04$) в ячейке оперативной памяти с адресом 04 ($a_1=04$) значение результата предыдущей операции. В ячейку 04, хранящую текущее значение переменной y, записывается значение, равное x^2c;

6) 02 02 04 — сложить ($K=02$) два числа, хранящиеся в ячейках ОП с адресами 02 и 04. В результате вычисляется значение, равное cx^2+b;

7) 04 04 00 — запомнить ($K=04$) в ячейке 04 ($a_1=04$) значение результата предыдущей операции. В итоге в ячейке ОП с адресом 04 хранится искомое значение переменной y, равное cx^2+b;

8) 05 01 04 — вывести на печать ($K=05$) одно число ($a_1=01$), хранящееся в ячейке ОП с адресом 04. На печать выводится значение переменной y;

9) 06 00 00 — прекратить выполнение программы ($K=06$).

Составленная программа включает в себя девять команд. *Для хранения программы необходимо выделить память.* Примем решение разместить программу в поле памяти, начинающемся с адреса 06 (адреса 01—04 отведены под исходные данные и результат; адрес — 05 оставлен свободным, так как он потребуется в дальнейшем).

Составленную программу и исходные данные необходимо подготовить для ввода в ЭВМ. Так как при составлении программы использовалась команда «ввести с перфокарт», то и программу для ввода в ЭВМ, и исходные данные подготовим на перфокартах. Всего будет отперфорировано двенадцать перфокарт: девять — с командами программы и три — с исходными данными.

Взаимодействие устройств ЭВМ. Согласованная работа устройств ЭВМ (см. рис. 2.1) по автоматическому

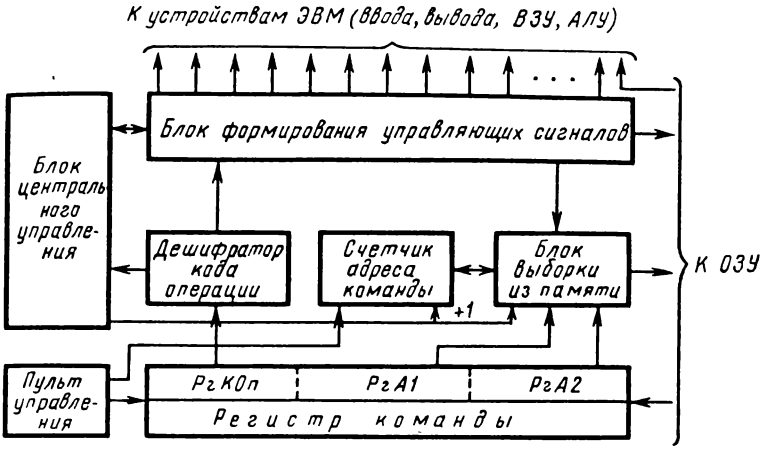


Рис. 3.2

выполнению заданной программы работы, как указывалось, обеспечивается устройством управления (УУ). На рис. 3.2 приведена простейшая структурная схема, поясняющая принцип функционирования УУ на примере выполнения составленной программы.

Устройство управления обеспечивает последовательное выполнение команд программы. Адрес очередной команды хранится в счетчике адреса команд. По сигналу от блока центрального управления «начать выполнение очередной команды» блок выборки из памяти считывает из ОЗУ очередную команду и записывает ее на временное хранение в регистр команды, в котором можно условно выделить регистр кода операции (*Рг КОп*), регистр первого адреса (*РгА1*) и регистр второго адреса (*РгА2*). Дешифратор кода команды расшифровывает код операции, а блок формирования управляющих сигналов (БФУС) выдает сигналы, необходимые устройствам ЭВМ (ввода, вывода, ВЗУ, АЛУ) для выполнения заданной операции. Если операция является арифметической, то от БФУС сначала поступает сигнал в блок выборки из памяти «считать из ОЗУ значения операндов и записать их в регистры *РгА* и *РгВ* арифметико-логического устройства» (см. рис. 2.2). Затем подается сигнал в блок управления АЛУ на выполнение этой операции. После чего в счетчик адреса команды добавляется 1 и на этом все действия по выполнению команды заканчиваются. Устройство управления готово приступить к следующей команде, хранящейся в ячейке ОЗУ с адресом, на единицу бóльшим адреса предыдущей команды.

На практике часто встречаются случаи, когда вычислительный процесс разветвляется и дальнейший ход вычислений зависит от результатов проверки соблюдения каких-либо условий. Поэтому в состав команд ЭВМ включают специальные команды, относящиеся к группе *команд управления вычислительным процессом*. Эти команды позволяют нарушить естественный порядок выполнения программы и указывать адрес команды, к которой необходимо перейти при соблюдении определенного условия. В качестве такого условия чаще всего используется сравнение результата предыдущей операции с нулем. Так, например, может выполняться двухадресная команда «иди по условию»:

Ka_1a_2.

Здесь, как и выше, K — код операции (пусть для этой операции он равен 07); a_1 — адрес команды программы, к которой следует перейти, если результат предыдущей операции отрицательный; a_2 — адрес команды, к которой следует перейти, если результат предыдущей операции положительный. В том случае, если результат предыдущей операции равен нулю, сохраняется естественный порядок выполнения программы.

Рассмотренная команда является примером команды *условного перехода*. Для организации вычислительных процессов используются также команды *безусловного перехода*. В результате действия такой команды, вне зависимости от каких-либо условий, прерывается естественный порядок выполнения программы и *управление передается любой другой команде*.

Команда безусловного перехода в принятой системе обозначений записывается как

Ka_1a_2,

где K — код операции безусловного перехода (например 08); a_1 — адрес команды, которой передается управление. Адрес a_2 в этой команде не используется.

Если очередная команда программы оказалась командой условного перехода, то блок центрального управления (см. рис. 3.2) анализирует результат предыдущей операции, который хранится в сумматоре АЛУ. Если знак результата отрицательный, то в счетчик адреса команд УУ записывается адрес из *РгА1*. Если знак результата положительный, а значение не нулевое, то в счетчик адреса команд записывается адрес из регистра *РгА2*. Если результат предыдущей операции равен нулю, то сохраняется естественный порядок выполнения програм-

мы и в счетчик адреса команд добавляется 1. На этом выполнение команды заканчивается и управление передается по адресу, который хранится в счетчике адреса команды.

Если очередная команда программы оказывается командой безусловного перехода, то выполнение ее состоит в пересылке значения из регистра *РгА1* в счетчик адреса команд.

Ввод программы в ЭВМ и ее выполнение. Будем считать, что для выполнения действий, обеспечивающих ввод программы в ЭВМ, можно воспользоваться пультом управления (ПУ), который позволяет вручную набрать любую команду, занести ее в любую ячейку оперативной памяти, вручную установить значение счетчика адреса команды.

Составленная ранее программа подготовлена к вводу на перфокартах. Для ввода ее необходимо вручную на ПУ набрать команду ввода программы с перфокарт: 01 09 06. Эта команда предписывает ввести ($K=01$) девять ($a_1=09$) перфокарт в поле оперативной памяти, начинающееся с ячейки 06 ($a_2=06$).

Занесем с ПУ набранную команду в ячейку оперативной памяти с адресом 05 и установим значение счетчика адреса команды, равное 05. Перфокарты с программой положим в приемный карман устройства ввода с перфокарт. Вслед за перфокартами программы положим перфокарты с отперфорированными значениями исходных данных в следующем порядке: *c*, *b*, *x*. После этого можно считать ЭВМ готовой к вводу программы.

Перевод машины в автоматический режим выполнения программы осуществляется нажатием на ПУ кнопки ПУСК, после чего автоматически будут выполняться следующие действия:

1. В регистр команды заносится содержимое ячейки 05 ОЗУ (в счетчик адреса команд было занесено значение 05). Это будет команда:

01 09 06

После дешифрации кода операции (01 — ввести с перфокарт) блок выработки управляющих сигналов выдает в устройство ввода с перфокарт сигнал о необходимости выполнения команды «ввести девять ($a_1=09$) перфокарт и записать считанные с перфокарт данные (команды программы) в поле оперативной памяти, начинающееся с ячейки 06». По окончании ввода блок центрального

управления увеличивает счетчик адреса команды на 1 (СчАК = 05 + 1) и разрешает считывание из памяти очередной команды.

2. Очередная команда считывается из ячейки ОЗУ с адресом 06. В этой ячейке хранится первая команда выполняемой программы. В ячейки 01, 02, 03 вводятся исходные данные *c*, *b*, *x*. Счетчик адреса команд увеличивается на 1 и начинается выполнение второй команды программы:

03 03 03

в следующем порядке: дешифрируется код операции «умножить»; блок выборки из памяти выбирает из ячейки 03 ОЗУ значение *x* и посылает его на хранение в регистр РгА АЛУ, затем в регистр РгВ АЛУ; блок формирования управляющих сигналов посылает в АЛУ сигналы, необходимые для выполнения операции умножения; после ее завершения блок центрального управления увеличивает значение СчАК на 1 и осуществляется переход к следующей команде. И так продолжается до тех пор, пока управление не будет передано команде девять:

06 00 00.

По этой команде машина прекратит автоматическую работу и перейдет в состояние ожидания следующего нажатия кнопки ПУСК.

Описанный процесс выполнения программы практически в полной мере отражает режим работы первых ЭВМ. Основной недостаток его — неэффективное использование ЭВМ. Время исполнения на ЭВМ даже сравнительно больших программ составляет секунды, а процесс перехода от одной программы к другой в описанном случае может потребовать минуту и более. Таким образом, дорогостоящее оборудование (первые ЭВМ были очень дороги) бóльшую часть времени простаивало в ожидании задач. Чтобы исключить простои ЭВМ из-за ввода программ, осуществляемых операторами, была разработана специальная управляющая программа, которая взяла на себя выполнение не только простых функций оператора ЭВМ, но и многих других достаточно сложных функций, обеспечивающих высокую загрузку ЭВМ задачами и значительно упростивших для пользователей процесс подготови и выполнения программ. Основные понятия, связанные с функциями управляющей

программы современных ЭВМ, будут рассмотрены в последующих книгах данной серии. Здесь же еще раз подчеркнем, что описанный процесс составления и выполнения программы служит лишь целям демонстрации общих идей управления автоматическими ЭВМ и является весьма условным (произвольно задана система команд, в большой степени произвольно выбран состав блоков устройства управления и т. п.).

3.2. ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Рассмотренный пример составления программы дает представление о *программировании в машинных кодах*, характерном для начального этапа развития вычислительной техники. В настоящее время подобные программы составляются редко и лишь для узкоспециализированных управляющих ЭВМ.

С начала развития вычислительной техники стала очевидной возможность автоматизации процесса получения программ. Основная идея *автоматизации программирования* состояла в разработке специальных языков, использование которых для написания программ было бы более удобным для человека, чем программирование в машинных кодах. Но в отличие от естественных языков (русского, английского и т. д.), допускающих возможность неоднозначного толкования составленных на них предложений, язык программирования должен быть строгим, позволяющим автоматически однозначно преобразовать написанную на нем программу в программу на языке машины, т. е. в машинные коды.

Первым шагом в этом направлении явилось создание так называемых *автокодов*. Автокоды являются *машинно-ориентированными языками* программирования. Это означает, что программы, написанные на автокоде, выполняются только на тех ЭВМ, для которых разработан соответствующий автокод. Состав и структура команд автокода полностью соответствуют составу и структуре команд ЭВМ. Для записи кодов операций в автокодах используются мнемонические обозначения, облегчающие программисту запоминание команд. Например, для использовавшихся в предыдущем параграфе команд можно ввести следующие мнемонические обозначения:

ВВПК — ввести с перфокарт;
СЛ — сложить;
УМ — умножить;
ЗР — запомнить результат;
ВЫПЧ — вывести на печать;
ОСТ — остановить.

При программировании на автокоде упрощается процесс распределения памяти. В адресной части команд вместо конкретных значений адресов записываются наименования переменных решаемой задачи. Ниже для сравнения приведены тексты программы вычисления выражения $y=cx^2+b$ в машинных кодах (слева) и на соответствующем автокоде (справа):

```
01 01 01    ВВПК     1, C
01 01 02    ВВПК     1, B
01 01 03    ВВПК     1, X
03 03 03    УМ       X, X
04 04 00    ЗР       Y
03 01 04    УМ       C, Y
04 04 00    ЗР       Y
02 02 04    СЛ       B, Y
04 04 00    ЗР       Y
05 01 04    ВЫПЧ     Y
06 00 00    ОСТ
```

В приведенных программах для ввода переменных *c*, *b*, *x* использованы три отдельные команды.

Рассмотренный пример позволяет убедиться в том, что программа, написанная на автокоде, более наглядна, легко читается и полностью соответствует текстовому описанию алгоритма решения задачи, приведенному в предыдущем параграфе.

Перевод программы из автокода в машинные коды осуществляется автоматически специальной *программой-транслятором*, для которой команды программы на автокоде являются исходными данными, а программа в машинных кодах — результатом работы (рис. 3.3).

Упрощенно процесс трансляции описывается следующим алгоритмом:

1. Составить список переменных, используемых в адресной части команд программы на автокоде.

В результате получим список: C, B, X, Y.

2. Поставить в соответствие каждой переменной из списка значения адресов ячеек оперативной памяти, начиная с первой свободной ячейки (например, с ячейки 01).

Рис. 3.3

Получим: С→01; В→02; Х→03; Y→04.

3. Переписать команды программы на автокоде, заменяя мнемонические наименования кодов операций их числовыми эквивалентами (для этого программа-транслятор обращается к специальной таблице, устанавливающей соответствие между мнемоническими обозначениями кодов и их числовыми эквивалентами). Вместо переменных в адресной части команд записываются поставленные им в соответствие (шаг **2** алгоритма) значения адресов ячеек оперативной памяти.

В результате первая команда запишется в виде

01 01 01, ...;

шестая — в виде

03 01 04

и т. д.

Для того чтобы понять, насколько упростился процесс программирования с появлением автокодов, надо принять во внимание, что программы даже первых ЭВМ включали в себя до нескольких тысяч команд, а современные ЭВМ работают под управлением программных комплексов, число команд которых измеряется десятками и сотнями тысяч. Поэтому вручную распределять память ЭВМ, отыскивать ошибки для программ такого размера чрезвычайно трудоемко.

Автокоды современных ЭВМ образуют группу языков программирования, известных под общим названием *ассемблеры*. Ассемблеры позволяют составлять эффективные программы, реализующие все возможности, предусмотренные системой команд и конструкцией ЭВМ. Однако программирование на ассемблерах остается весьма трудоемким, поэтому к использованию их прибегают лишь в тех случаях, когда нельзя применить другие языки программирования. К недостаткам ассемблеров можно отнести и их машинную ориентированность, высокие требования, предъявляемые к уровню подготовки программистов, от которых требуется хорошее знание конструктивных особенностей ЭВМ.

Дальнейшее развитие языковых средств шло по пути создания *машинно-независимых языков*, позволяющих выполнять написанные на них программы на любых ЭВМ. При разработке языков программирования ставилась цель в возможно большей степени приблизить их к профессиональному языку пользователей ЭВМ различных

областей науки и техники, не являющихся специалистами в области программирования. За короткое время появилось большое число так называемых *процедурно-ориентированных языков* программирования, предназначенных для решения инженерных и научно-технических задач, задач обработки экономической информации, обработки списков, моделирования и т. д. Среди первых процедурно-ориентированных языков, получивших наибольшее признание и распространение, выделяются ФОРТРАН, АЛГОЛ, КОБОЛ.

Язык программирования ФОРТРАН, разработанный в 1956 г., широко используется для решения научных и инженерно-технических задач. На языке ФОРТРАН накоплено большое число программ, относящихся к различным областям науки и техники. Язык ФОРТРАН прост в освоении. Реализации языка для ЭВМ различных типов отличаются незначительно, что в большинстве случаев позволяет переносить программы, написанные для ЭВМ одного типа, на ЭВМ других типов.

Чтобы представить себе, насколько проще писать программы вычислительного характера на языке ФОРТРАН, чем на языке Ассемблер, приведем простой пример. Пусть требуется составить программу вычисления выражения

$$y=\frac{ax^2+bx+c}{2{,}5d+1{,}14q}-h^{12}.$$

Если составлять программу на ассемблере (автокоде), подобно тому как делалось это ранее, можно подсчитать, что программа для вычисления приведенного выражения будет включать в себя около 30 команд. На языке ФОРТРАН эта же программа включает в себя всего три оператора (программа на алгоритмических языках определяется не как последовательность команд, а как последовательность операторов):

```
READ (5) A, B, X, C, D, Q, H
Y=(A*X**2+B*X+C)/(2.5*D+1.14*Q)-H**12
WRITE (6) Y
```

Первый оператор предписывает ввести (используется английское слово read — читать) с устройства ввода № 5 (это может быть, например, устройство ввода с перфокарт) значения переменных A, B, X, ... Второй оператор предписывает вычислить арифметическое выражение, стоящее справа от знака «=», и присвоить вычисленное

значение переменной Y. В арифметическом выражении знак «*» используется для обозначения операций умножения, знак «**» — возведения в степень, знак «/» — деления. Третий оператор предписывает вывести (используется английское слово write — писать) на устройство № 6 (это может быть устройство печати) значение переменной Y.

Производительность труда программистов измеряется числом строк программы, написанной и отлаженной одним программистом в единицу времени (в день, в неделю, ...). Практика разработки программ на различных языках программирования показала, что это количество практически не зависит от используемого языка программирования. Поэтому, анализируя приведенный пример, можно сделать вывод, что программирование на языке ФОРТРАН приблизительно в 10 раз эффективнее, чем на Ассемблере. Поэтому такие языки, как ФОРТРАН, АЛГОЛ, КОБОЛ и др., в отличие от ассемблеров относят к языкам высокого уровня, имея в виду, что уровень языка определяется средним отношением числа операторов в программе, написанной на этом языке, к числу команд программы в машинных кодах, полученной после трансляции. Чем выше это отношение, тем выше уровень языка, а следовательно, и обеспечиваемый им уровень автоматизации программирования.

Язык АЛГОЛ, разработанный для решения сложных вычислительных задач, был одним из первых языков, унифицированных на международном уровне. Это позволило использовать его не только как эффективный язык программирования, но и как язык для публикации алгоритмов.

Язык КОБОЛ был разработан и широко применялся для программирования задач обработки экономической информации.

Расширение сферы применения ЭВМ и усложнение решаемых задач потребовали языки, которые, обладая, например, необходимыми средствами для обработки экономической информации, позволяли бы описывать и сложные вычислительные процессы. Такие языки были разработаны и получили название *универсальных языков программирования*. Наиболее распространенный из них — ПЛ/1.

В настоящее время наряду с упомянутыми языками широко используются и многие другие языки, сочетающие большую универсальность с простотой освоения и использования: ПАСКАЛЬ, СИ, БЕЙСИК, ПРОЛОГ и др.

3.3. ОСНОВНЫЕ ЭТАПЫ ПОДГОТОВКИ ЗАДАЧ К РЕШЕНИЮ НА ЭВМ

Решение задачи на ЭВМ сводится к выполнению программы, введенной в память ЭВМ. Составлению программы предшествует разработка алгоритма решения задачи. Алгоритм реализует метод решения. Выбору метода решения предшествует анализ и формализация условий задачи. Таким образом, в процессе подготовки задач к решению на ЭВМ можно выделить ряд последовательных этапов (рис. 3.4). На каждом из них в создаваемую программу решения задачи могут быть внесены ошибки, поиск и устранение которых являются целью завершающего этапа подготовки программы — ее отладки. После того как программа отлажена, она выполняется на ЭВМ.

Математическая постановка задачи. Точное описание исходных данных, условий задачи и целей ее решения называется *математической постановкой задачи.* Этап разработки математической постановки задачи называют также *этапом формализации,* так как на этом этапе многие из условий задачи, заданные в форме различных словесных описаний, необходимо выразить на точном (формальном) языке математики. Если задача является математической, то выполнение этапа формализации может и не потребоваться.

Пусть требуется найти корни системы из 16 алгебраических уравнений с 16 неизвестными. Если известны уравнения системы и метод ее решения, то можно считать, что математическая постановка задачи ясна.

На практике ЭВМ часто используется для решения достаточно сложных задач, и не только вычислительных, поэтому этап формализации может потребовать значительных усилий и времени. Среди опытных программистов-поставщиков задач распространено мнение, что выполнить этап формализации — это значит сделать половину всей работы по созданию программы.

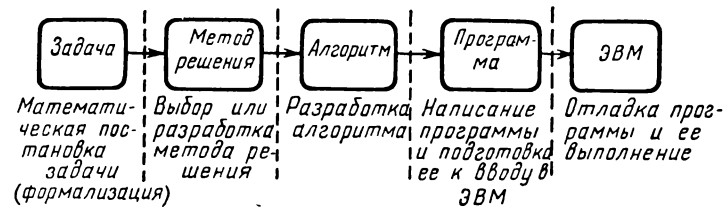


Рис. 3.4

а)

Время занятий		Учебные группы				
		П1-87	П2-87	П3-87	П4-87	П5-87
Понедельник	8^{30}-10^{05}	История проф. Афанасьев (ауд. 403)				
	10^{20}-11^{55}	Математика доц. Гогорев (ауд. 264)			Лабораторные работы по физике	Практикум по ЭВМ
	12^{10}-13^{45}	Физика (ауд. 212)	Черчение (зал черчения)	Физика (ауд. 214)		
	14^{00}-15^{35}	Физкультура				
Вторник	8^{30}-10^{05}	Лабораторные работы по физике	Практикум по ЭВМ	История (ауд. 216)	Математика доц. Круглов (ауд. 262)	
	11^{20}-11^{55}			Черчение (зал черчения)	Физика (ауд. 214)	История (ауд. 216)

б)

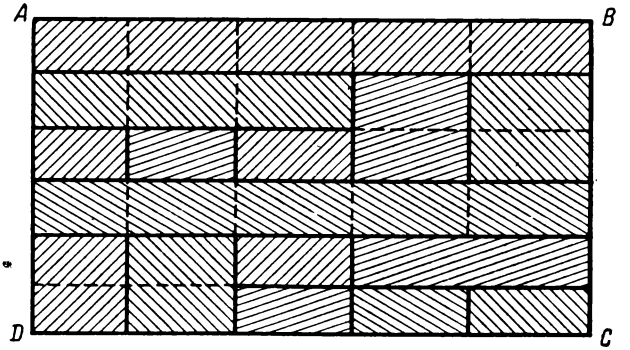


Рис. 3.5

На этапе формализации задачи определяют состав и характеристики исходных данных. Различные условия и ограничения записываются в математической форме.

На рис. 3.5, *а, б* приведен пример, поясняющий один из возможных подходов к формализации задачи составления вузовского расписания учебных занятий. При этом используется геометрическая интерпретация задачи, когда ее решение сводится к отысканию покрытия заданного прямоугольника *ABCD* (рис. 3.5, *б*) набором меньших прямоугольников, суммарная площадь которых равна площади прямоугольника *ABCD*. Каждый прямоугольник на рис. 3.5, *б* соответствует конкретному учебному занятию на рис. 3.5, *а*. При этом «размер» прямоугольника по горизонтали определяется числом учебных групп,

одновременно участвующих в занятии, а «размер» по вертикали определяется длительностью занятия, выраженной в парах часов. Возможные варианты расписаний учебных занятий определяются существующими вариантами покрытия прямоугольника $ABCD$ «прямоугольниками учебных занятий», удовлетворяющих обязательным требованиям, предъявляемым к расписанию. На этапе формализации тщательно составляется перечень этих требований, например: каждая учебная группа одновременно может участвовать только в одном учебном занятии; в одном учебном помещении в одно время может проходить только одно учебное занятие и т. д.

Каждое из требований записывается в виде математических уравнений. Так, последнее из сформулированных требований может быть задано в виде

$$\sum_{k=1}^{n} z_k^l \leqslant 1 \text{ для всех } i, j \text{ и } l,$$

где

$$z_k^l = \begin{cases} 1 \text{ — если учебное занятие } z_k \text{ назначено в учебном помещении } l; \\ 0 \text{ — в противном случае;} \end{cases}$$

n — общее число учебных занятий в расписании; $i=\overline{1,6}$ — дни учебной недели; $j=\overline{1,8}$ — номера пар часов учебных занятий в течение дня.

Кроме обязательных требований можно сформулировать дополнительные требования, по степени выполнения которых можно судить о качестве составленного расписания. Например: желательно, чтобы учебный день начинался с более сложных занятий и заканчивался более простыми; интенсивность учебной нагрузки должна возрастать к середине недели и спадать к ее концу и т. д.

Дополнительные требования также выражаются в виде различных математических условий, на основе которых составляется функция, позволяющая количественно оценить качество расписания — *критерий качества.*

Таким образом, в конечном итоге задача составления расписания учебных занятий сводится к известной в комбинаторной геометрии задаче нахождения покрытия некоторой поверхности (в общем случае она может быть и не прямоугольником) заданным набором прямоугольников, причем искомое покрытие должно удовлетворять ряду ограничений, записанных в виде математических уравнений, и оцениваться максимально возможным значением критерия качества.

Полученная на этапе формализации новая задача называется *математической моделью исходной задачи.* Она может использоваться и для решения многих других задач, относящихся как к задачам календарного планирования (составления расписаний), так и к задачам других классов. Рассмотренная задача нахождения покрытия прямоугольника прямоугольниками может быть, например, использована в качестве математической модели для задач раскроя, упаковки, компоновки.

Выбор или разработка метода решения. Этот этап нельзя рассматривать как самостоятельный. Он тесно связан с этапом формализации, так как целью его является сведение задачи к математической модели, для которой известен метод решения. Однако возможно, что для полученной модели известны несколько методов решения и тогда предстоит выбрать лучший.

Иногда оказывается, что ни один из известных методов нельзя использовать для решения задачи. Тогда требуется вернуться к этапу формализации и попытаться упростить решаемую задачу путем введения дополнительных ограничений или снятия некоторых из них. Кроме того, можно усовершенствовать существующие или разработать новые методы решения формализованной задачи. Эта работа по своему характеру является научно-исследовательской и может потребовать значительных усилий.

На практике встречаются случаи, когда приходится искать метод решения на ЭВМ задач, которые без ЭВМ решаются достаточно просто. Вот широко известный пример такой задачи: треугольник ABC задан координатами вершин (x_A, y_A, x_B, y_B, x_C, y_C). Определить, лежит ли внутри треугольника точка D, заданная координатами x_D, y_D.

Первые попытки найти метод ее «машинного» решения редко бывают удачными и обычно сводятся к предложениям определить и сравнить длины различных отрезков (AD, AB, BC, CB, BD, CD), а также высот, биссектрис и т. п.

Один из простых возможных машинных методов решения данной задачи иллюстрируется рис. 3.6, *а*, и *б*. Он базируется на следующем очевидном утверждении: если точка D лежит внутри треугольника ABC, то выполняется равенство

$$S_{\Delta ABC} = S_{\Delta ADC} + S_{\Delta ADB} + S_{\Delta CDB}, \qquad (3.1)$$

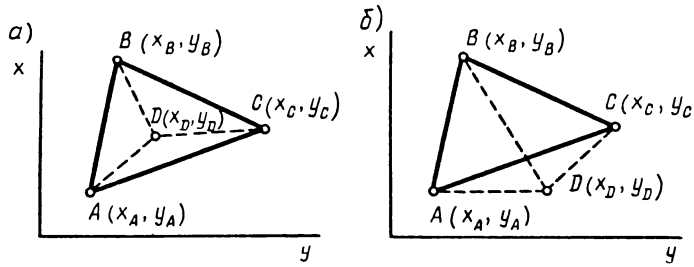


Рис. 3.6

где $S_{\Delta ABC}$, $S_{\Delta ADC}$, $S_{\Delta ADB}$ и $S_{\Delta CDB}$ — соответственно площади треугольников *ABC*, *ADC*, *ADB* и *CDB*.

Таким образом, для решения задачи необходимо вычислить площади четырех треугольников и проверить выполнение условия (3.1).

Разработка алгоритма. Алгоритм устанавливает последовательность точно определенных действий, приводящих к решению задачи (см. § 2.1). При этом последовательность действий может задаваться посредством словесного или графического описаний. Примеры словесного описания алгоритмов встречались в предыдущих параграфах данного пособия. Для того чтобы придать словесным описаниям алгоритмов определенную строгость, создаются и используются специальные алгоритмические языки; в освоении они обычно более просты, чем языки программирования, многие из которых также используются для описания алгоритмов.

Один из языков, специально предназначенных для описания алгоритмов, изучается в средней школе. Приведем описанный на таком языке алгоритм решения задачи об определении принадлежности точки *D* треугольнику *ABC*:

алг Определение принадлежности точки треугольнику (дейст x_A, y_A, x_B, y_B, x_C, y_C, x_D, y_D целое z лит a);
арг x_A, y_A, x_B, y_B, x_C, y_C, x_D, y_D;
рез z, a;
нач
дейст S_1, S_2, S_3, S_4
вычислить значение S_1, равное площади ΔABC;
вычислить значение S_2, равное площади ΔABD;
вычислить значение S_3, равное площади ΔACD;
вычислить значение S_4, равное площади ΔCDB;

```
если S1 = S2 + S3 + S4
то  z := 1,
    a := «точка внутри треугольника»,
иначе z := 0,
    a := «точка вне треугольника»;
все
напечатать значение a:
кон
```

Слова, подчеркнутые в тексте алгоритма, называются *служебными*. Алгоритм начинается служебным словом алг (алгоритм), вслед за ним записывается название алгоритма и далее в скобках перечисляются обозначения переменных, исходных для данного алгоритма (x_A, y_A, ..., y_D), и переменных, получающих значения в результате выполнения алгоритма (z и a). Здесь же описывается тип переменных. Переменные x_A, ..., y_D описаны как действительные (используется служебное слово действ). Это означает, что переменные хранят в памяти ЭВМ в форме с плавающей запятой (см. § 1.4). Как будет видно из дальнейшего, это обстоятельство необходимо учитывать. Переменная z описана как целая, а переменная a — как литерная (используется служебное слово лит). В отличие от числовых переменных (действительных и целых), значениями которых являются числа, *значениями литерных переменных являются символьные тексты*. В данном алгоритме литерная переменная a может принять одно из двух текстовых значений: «точка внутри треугольника», «точка вне треугольника». Исходные для алгоритма переменные называются *аргументами* и описываются отдельным предложением, начинающимся служебным словом арг. Следующее предложение, начинающееся служебным словом рез, указывает переменные, являющиеся результатами алгоритма. Описанные три предложения, начинающиеся служебными словами алг, арг и рез, образуют *заголовок алгоритма*. Для обозначения начала собственно алгоритма используется служебное слово нач, для обозначения его конца — служебное слово кон. Между этими служебными словами записываются в форме повелительных предложений и математических формул предписания или команды алгоритма, последовательное выполнение которых обеспечивает получение результата решения задачи.

В ходе выполнения алгоритма вычисляются значения *промежуточных переменных* S_1, S_2, S_3, S_4, описание которых помещается после служебного слова нач.

Следует обратить внимание на отличие используемых в алгоритме знаков «=» и «:=». Знак = (равно)

указывает на отношение равенства между правыми и левыми частями выражения $S_1 = S_2 + S_3 + S_4$. Подобные выражения в информатике называются *логическими условиями*. Они могут быть *истинными* или *ложными* и часто используются в алгоритмах и программах для организации ветвлений. В приведенном алгоритме используется специальная команда ветвления:

если условие,
то серия команд 1,
иначе серия команд 2;
все

Знак := называется *знаком присвоения значения*. Например, строка алгоритма $A := B$ предписывает переменной A присвоить значение переменной B. В общем случае в правой части может находиться любое арифметическое выражение. Например:

$$A := bx^2 + c \cos x.$$

В результате выполнения команды присваивания вычисляется значение арифметического выражения в правой части, а затем вычисленное значение присваивается переменной, стоящей слева от знака :=.

В приведенном алгоритме команду ветвления можно было записать и следующим образом:

если $S_1 < S_2 + S_3 + S_4$,
то $z := 0$,
$a :=$ «точка вне треугольника».
иначе $z := 1$,
$a :=$ «точка внутри треугольника»;
все

При реализации ветвлений в программах и алгоритмах необходимо учитывать влияние точности вычислений на результаты. В рассматриваемых примерах использовались два условия:

а) $S_1 = S_2 + S_3 + S_4$; б) $S_1 < S_2 + S_3 + S_4$.

Значения переменных S_1, S_2, S_3, S_4 вычисляются в машине приближенно (см. § 1.4). Пусть максимальная погрешность вычисления площади треугольника составит δ. Тогда максимальное значение $(S_1)_{max} = S_1 + \delta$, а максимальное значение суммы $(S_2 + S_3 + S_4)_{max} = S_2 + S_3 + S_4 + 3\delta$. Очевидно, что если не учесть указанного обстоятельства, то для всех случаев, когда точка D находится внутри треугольника ABC, условие $S_1 = S_2 + S_3 + S_4$ не будет выпо-

ляться. Поэтому в рассматриваемом алгоритме предпочтительнее использовать условие б), но и оно должно быть скорректировано с учетом точности вычислений:

$$S_1 < S_2 + S_3 + S_4 - \sigma,$$

где σ должна быть возможно меньшей величиной, но заведомо большей суммарной максимальной погрешности вычисления на ЭВМ.

Следует иметь в виду, что использование для разветвлений условия в виде равенства вообще нежелательно и допустимо лишь для целых чисел.

Рассмотренный алгоритм включает в себя четыре предписания «вычислить значение площади треугольника». Площадь треугольника вычисляется по формуле

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

где p — полупериметр; a, b и c — длины его сторон.

Алгоритм вычисления площади треугольника называется *вспомогательным* по отношению к алгоритму определения принадлежности точки к треугольнику и записывается следующим образом:

```
алг Площадь треугольника (дейст x_A, y_A, x_B, y_B, x_C, y_C, дейст S);
    арг x_A, y_A, x_B, y_B, x_C, y_C;
    рез S;
        нач
            дейст AB, BC, CA, p;
            вычислить длину отрезка AB;
            вычислить длину отрезка BC;
            вычислить длину отрезка CA;
            p:=0,5(AB+BC+CA);
            S:=√(p(p-AB)(p-BC)(p-CA));
кон
```

Разработав вспомогательный алгоритм «площадь треугольника», можно в главном алгоритме «определение принадлежности точки треугольнику» заменить предписания «вычислить площадь треугольника» *командой вызова вспомогательного алгоритма*. В этой команде указываются имя вызываемого алгоритма и (в скобках) имена переменных как исходных, так и тех, которые являются результатом выполнения вспомогательного алгоритма. Так, предписание

```
вычислить значение S_1, равное площади ΔABC;
```

в главном алгоритме можно записать в виде следующей команды вызова вспомогательного алгоритма:

площадь треугольника (x_A, y_A, x_B, y_B, x_C, y_C, S_1).

В результате действия этой команды выполняется вспомогательный алгоритм вычисления площади треугольника с координатами вершин x_A, ..., y_C и полученное значение присваивается переменной S_1.

Анализ алгоритма «площадь треугольника» указывает на необходимость разработки еще одного вспомогательного алгоритма для вычисления длины отрезка, заданного координатами ограничивающих его точек:

алг Длина отрезка (дейст, x_A, y_A, x_B, y_B, дейст AB);
арг x_A, y_A, x_B, y_B;
рез AB;
нач
$$AB = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2};$$
кон

Таким образом, разработка алгоритма определения принадлежности точки треугольнику сводится к разработке главного и двух вспомогательных алгоритмов. Использованный метод разработки алгоритма называется *методом последовательных уточнений*. Суть его состоит в том, что при написании главного алгоритма выделяются его части, для выполнения которых можно разработать укрупненные команды, реализуемые по вспомогательным алгоритмам.

Тот же подход реализуется при разработке вспомогательных алгоритмов. Необходимая степень детализации алгоритмов обычно выбирается такой, чтобы предписания разработанных алгоритмов могли записываться на языке программирования, выбранном для составления текста программы. Ниже приведена запись алгоритмов «определение принадлежности точки треугольнику» и «площадь треугольника» с использованием команд обращения к вспомогательным алгоритмам:

```
алг Определение принадлежности точки треугольнику
    (дейст x_A, y_A, x_B, y_B, x_C, y_C, x_D, y_D целое z лит a);
    арг x_A, y_A, x_B, y_B, x_C, y_C, x_D, y_D;
    рез z, a;
        нач
            дейст S_1, S_2, S_3, S_4, σ;
            Площадь треугольника (x_A, y_A, x_B, y_B, x_C, y_C, x_D, y_D, S_1);
            Площадь треугольника (x_A, y_A, x_B, y_B, x_C, y_C, x_D, y_D, S_2);
            Площадь треугольника (x_A, y_A, x_B, y_B, x_C, y_C, x_D, y_D, S_3);
```

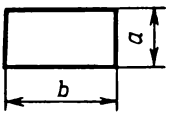
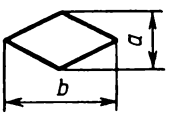
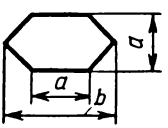
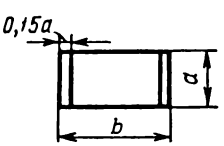
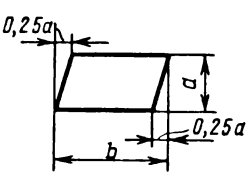
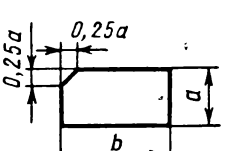
```
        Площадь треугольника (x_A, y_A, x_B, y_B, x_C, y_C, x_D, y_D, S_4);
        σ:=10^-5;
        если S_1<S_2+S_3+S_4-σ,
        то   z:=0,
             a:=«точка вне треугольника»,
        иначе z:=1,
             a:=«точка внутри треугольника»;
        все
        напечатать значение a;
кон

алг Площадь треугольника (действ x_A, y_A, x_B, y_B, x_C, y_C действ S);
    арг x_A, y_A, x_B, y_B, x_C, y_C
    рез S;
        нач
            действ AB, BC, CA, p;
            Длина отрезка (x_A, y_A, x_B, y_B, AB);
            Длина отрезка (x_B, y_B, x_C, y_C, BC);
            Длина отрезка (x_C, y_C, x_A, y_A, CA);
            p:=0,5 (AB+BC+CA);
            S:=√(p (p-AB) (p-BC) (p-CA));
кон
```

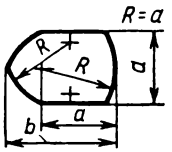
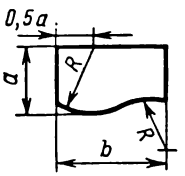
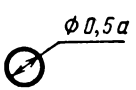
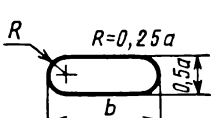
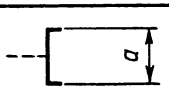
В результате выполнения главного алгоритма получают значения две переменные — z и a. Переменная a получает текстовое значение, которое последней командой алгоритма выводится на печать, чем обеспечивается сообщение человеку результата решения. С помощью переменной z значение результата «сообщается» другому алгоритму, для которого данный алгоритм используется как вспомогательный.

Наряду со словесной формой описания алгоритмов широко используется графический способ описания, который удобен во многих практических случаях. При графическом описании отдельные функции алгоритмов отображаются в виде условных графических изображений — символов. Перечень условных графических символов, их наименования, форма, размеры и отображаемые функции устанавливаются ГОСТ 19.003—80. Основные графические символы, используемые для описания алгоритмов, приведены в табл. 3.1.

Соотношения между геометрическими элементами символов устанавливаются стандартом. Размер a должен выбираться из ряда 10; 15; 20 мм. Допускается увеличивать размер a на число, кратное 5. Размер $b=1{,}5\,a$. Допускается устанавливать $b=2a$. Линии потока рекомендуется выполнять в два раза тоньше линий обводки блоков.

Таблица 3.1

Наименование	Обозначение	Функции
Процесс		Выполнение операции или группы операций, в результате которых изменяется значение, форма представления или расположение данных
Решение		Выбор направления выполнения алгоритма или программы в зависимости от некоторых переменных условий
Модификация		Выполнение операций, меняющих команды, или группы команд, изменяющих программу
Предопределенный процесс		Использование ранее созданных и отдельно описанных алгоритмов или программ
Ввод-вывод		Преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод)
Перфокарта		Ввод-вывод данных, носителем которых является перфокарта

Наименование	Обозначение	Функции
Дисплей		Ввод данных с подключенного к ЭВМ дисплея или вывод данных на дисплей
Документ		Ввод-вывод данных, носителем которых служит бумага
Линия потока		Указание на последовательность связей между символами
Соединитель		Указание на связь между прерванными линиями потока, соединяющими символы
Пуск-останов		Начало, конец, прерывание процесса обработки данных или выполнения программы
Комментарий		Связь между элементом схемы и пояснением

Правила выполнения алгоритмов с использованием рассмотренных символов устанавливает ГОСТ 19.002—80.

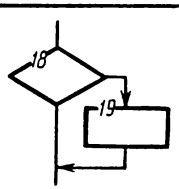
Линии потока проводят параллельно внешним краям рамки листа. Допускается пересечение их или изгиб под углом 90°. Направление линий потока сверху вниз и слева направо принимают за основное; если линии потока основного направления не имеют изломов, то их направления стрелками можно не обозначать. В остальных случаях направление линий потока обозначать стрелкой обязательно. Расстояние между параллельными линиями потока должны быть не менее 3 мм, между остальными символами схемы — не менее 5 мм.

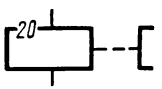
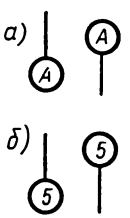
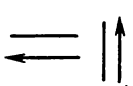
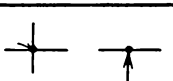
Записи внутри символа или рядом с ним должны выполняться машинописью с одним интервалом или чертежным шрифтом и должны быть краткими. Сокращенные слова и аббревиатуры, за исключением установленных стандартами, расшифровывают в нижней части поля схемы.

Для облегчения вычерчивания и нахождения на схеме символов рекомендуется поле листа разбивать на зоны. Размеры их устанавливают с учетом минимальных размеров символов, изображенных в зонах на данном листе. Координаты зон проставляют: по горизонтали — арабскими цифрами слева направо в верхней части листа; по вертикали — прописными буквами латинского алфавита сверху вниз в левой части листа. Координаты зон в виде сочетания букв и цифр присваивают символам, вписанным в поля этих зон, например *A1, A2, A3, B1, B2* и т. д. Если поле листа не разбито на зоны, то символам присваивают порядковые номера.

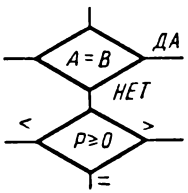
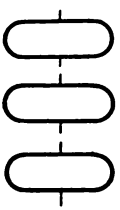
Некоторые примеры применения символов в схемах алгоритмов и программ приведены в табл. 3.2.

Таблица 3.2

Фрагмент схемы	Содержание изображения	Правила применения
	Обозначение символов в схемах: 18, 19 — порядковые номера символов на схеме	Порядковый номер символа проставляют слева в верхней части символа (при ручном исполнении схемы)

Фрагмент схемы	Содержание изображения	Правила применения
	Комментарий	Применяется, если пояснение не помещается внутри символа; комментарий записывают параллельно основной надписи, помещают в свободном месте схемы алгоритма на данном листе и соединяют с поясняемым символом
	Соединитель: *А*,5 — идентификаторы соединителя в виде: а) буквы; б) цифры	При большой насыщенности схемы символами отдельные линии потока между удаленными друг от друга символами допускается обрывать. При этом в конце (начале) обрыва должен быть помещен символ «соединитель»
	Линии потока	Символ применяют для указания линии потока. Можно без стрелки, если линия направлена слева направо и сверху вниз; со стрелкой — в остальных случаях
	Излом линии потока под углом 90°	Символ обозначает изменение направлений линий потока
	Пересечение линий потока	Символ применяют в случае пересечения двух несвязанных линий потока
	Слияние линий потока. Место слияний линий потока обозначено точкой	Символ применяют в случае слияния линий потока, каждая из которых направлена к одному и тому же символу на схеме. Место слияния линий потока обозначать точкой при ручном выполнении схем

Продолжение табл. 3.2

Фрагмент схемы	Содержание изображения	Правила применения
	Возможные варианты отображения решения: *A-B*, $p \geqslant 0$ — условия решений; *A*, *B*, *p* — параметры	При числе исходов не более трех признаки условия решения (например, ДА, НЕТ, =, >, <) проставляются над каждой выходящей линией потока или справа от линии потока
	Начало, прерывание и конец алгоритма или программы: а) пуск; б) прерывание; в) останов	Символы применяются в начале схемы алгоритма или программы, в случае прерывания ее и в конце. Внутри символа «пуск-останов» может указываться наименование действия или идентификатор программы

На рис. 3.7 приведен пример графического описания схемы рассмотренного выше алгоритма определения принадлежности точки треугольнику.

Алгоритм начинается и кончается символами «Пуск — Останов», в которых указываются соответственно записи «Начало», «Конец». Внутри функциональных символов указываются предписания алгоритма.

Для описания вспомогательных алгоритмов используется символ «Предопределенный процесс». Вспомогательные алгоритмы при необходимости могут изображаться на отдельной схеме. Тогда в символе «Пуск» обычно указывается имя вспомогательного алгоритма и его параметры (имена переменных, являющихся аргументами и результатами алгоритма). Удобно и смешанное описание алгоритмов, когда главный алгоритм изображается в виде графической схемы, а вспомогательные алгоритмы (если они не громоздки) описываются на алгоритмическом языке.

Написание программы и подготовка ее к вводу в ЭВМ. Цель данного этапа состоит в записи алгоритма на языке программирования и переносе текста программы на носитель, с которого она вводится в ЭВМ. Подготовка программы к вводу в ЭВМ может осуществляться

устройствами подготовки данных на перфолентах и перфокартах, магнитных лентах и гибких магнитных дисках. Широко используется ввод программ в ЭВМ с клавиатуры подключенных терминалов (дисплеев).

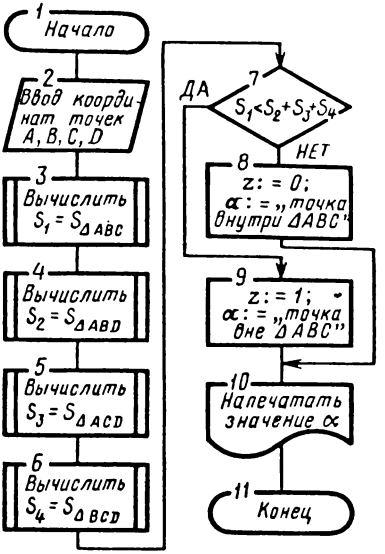


Рис. 3.7

Для написания текста программы необходимо выбрать язык программирования. Обычно решение об используемом языке программирования принимается на этапе разработки алгоритма, так как уровень детализации разрабатываемого алгоритма определяется из условия обеспечения свободной записи предписаний алгоритма на том языке программирования, который предполагается использовать.

Задачи, которые решает пользователь ЭВМ, чаще всего относятся к одному классу и могут быть запрограммированы на одном и том же языке, обычно уже освоенном пользователем. Это гарантирует более быстрое написание программы с меньшим числом ошибок.

Если программу в силу ее специфики или специфики ЭВМ нельзя написать на языке, привычном пользователю, то необходимо использовать другой язык программирования. Для написания программы могут использоваться и несколько языков. Так, опытные программисты, используя для написания программы один из языков высокого уровня (ПЛ/1, ФОРТРАН), включают в нее фрагменты, написанные на языке Ассемблер, позволяющем запрограммировать алгоритмы практически любой сложности. С той же целью в программу на языке ФОРТРАН могут включаться фрагменты на языке ПЛ/1 и т. д.

Значительного сокращения сроков программирования задач можно достичь и за счет использования готовых программ. Готовые программы, предназначенные для решения наиболее распространенных задач, составляют так называемые *библиотеки стандартных программ*

(БСП). Для современных ЭВМ разработаны БСП, включающие тысячи стандартных программ для решения различных задач вычислительной математики, статистического анализа, обработки данных и др. Программы, включенные в библиотеку, оформляются специальным способом, облегчающим их вызов, использование, передачу входных данных (аргументов) и результатов. В современных языках программирования имеются средства для написания и использования стандартных программ (подпрограмм).

Понятию *подпрограммы* соответствует понятие вспомогательного алгоритма. Как и вспомогательный алгоритм, подпрограмма имеет имя, за которым, обычно в скобках, перечисляются формальные параметры, являющиеся аргументами и результатами реализуемого подпрограммой вспомогательного алгоритма.

Написанные программистом стандартные программы помещаются в *общую* или *личную библиотеку* стандартных программ. Общие БСП предназначаются для использования всеми программистами данного вычислительного центра. Личные БСП создаются программистами для себя или небольшой группы программистов, работающих над общей задачей. Программы, включенные в БСП, хранятся во внешней памяти ЭВМ (обычно на магнитных носителях) и автоматически вызываются для выполнения специальной командой вызова подпрограммы, записываемой программистом в соответствующем месте программы.

Выявление возможностей использования стандартных программ осуществляется еще на этапе разработки алгоритма. Фрагменты алгоритма, для реализации которых можно использовать стандартные программы, оформляются как предписания вызова соответствующих вспомогательных алгоритмов. Если в библиотеках стандартных программ отсутствуют программы, реализующие вспомогательные алгоритмы, то соответствующие программы составляются программистом и оформляются по определенным правилам как стандартные подпрограммы.

Дальнейшим развитием метода автоматизации программирования, основанного на представлении программы в виде отдельных программных блоков, является так называемый *метод структурного программирования*, применение которого позволяет существенно повысить производительность труда программистов.*

*Подробнее см. кн. 3 данной серии.

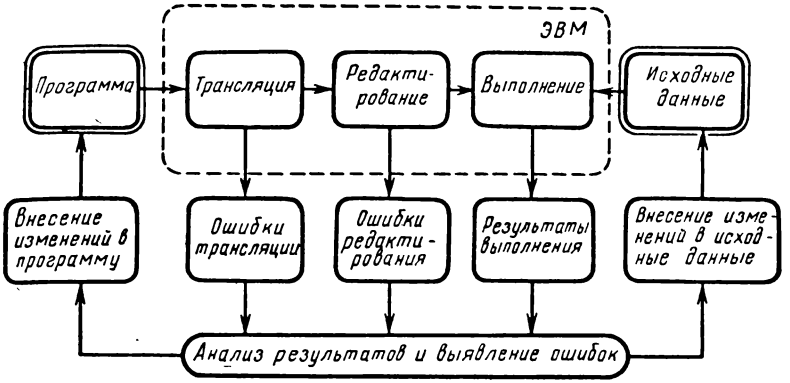


Рис. 3.8

Отладка программ. Основная цель этапа отладки — выявление и исправление ошибок в программе. По существующим оценкам, на отладку программист затрачивает от 20 до 40% времени, отводимого на разработку программы. Поэтому овладение приемами и средствами отладки программ важно для программиста.

Для проведения отладки программист готовит программу к выполнению на машине и исходные данные, обычно упрощенные, позволяющие по полученным результатам оценить правильность работы программы. Процесс отладки практически состоит из многократных попыток выполнения программы на машине и анализа получающихся результатов. Собственно процессу выполнения программы на машине предшествует трансляция программы (см. § 3.2) и ее редактирование, в ходе которого специальная программа РЕДАКТОР осуществляет «присоединение» к программе библиотечных стандартных программ и подпрограмм. Поэтому все многообразие ошибок, обнаруживаемых в процессе отладки, условно делится на ошибки, обнаруженные на этапах трансляции, редактирования и собственно выполнения программы. Общая схема отладки программы приведена на рис. 3.8.

Ошибки редактирования характерны для программных комплексов. Поэтому для облегчения отладки сначала проводят автономную отладку всех подпрограмм, а затем комплексную отладку, в процессе которой устраняют ошибки, обнаруживаемые на этапе редактирования (неправильное оформление подпрограмм), и ошибки в командах вызова подпрограмм и стандартных программ из различных БСП.

Автономная отладка программ начинается с выявления синтаксических ошибок. В процессе синтаксического контроля устраняются формальные ошибки, допущенные при записи текста программы на алгоритмическом языке, а также ошибки, внесенные на этапе кодирования и ввода программы в ЭВМ. Большинство синтаксических ошибок обнаруживается машиной автоматически на этапе трансляции. Современные трансляторы с языков программирования выдают информацию об ошибках вместе с текстом программы, указывают места ошибок и их характер. После исправления синтаксических ошибок программа транслируется и выполняется.

Небольшие программы (до нескольких десятков команд) могут отлаживаться полностью. Отладку больших программ проводят последовательно, блок за блоком в соответствии с алгоритмом. Начинают отладку с проверки правильности выполнения команд ввода исходных данных. Для этого в программу сразу же после команд ввода данных временно вставляют операторы вывода введенных данных на печать или экран дисплея. Убедившись в правильности ввода данных, переходят к отладке отдельных блоков. Для этого в программу обычно вставляют команды печати значений переменных на входе и выходе блоков. Значения переменных, полученные в результате выполнения на ЭВМ команд отлаживаемого блока, сравнивают с соответствующими значениями, вычисленными программистом другим путем, например с помощью микрокалькулятора. Особое внимание уделяется проверке правильности реализации разветвлений программы.

Необходимость постоянного изменения текста программы за счет введения в нее отладочных операторов и последующего их удаления затрудняет работу программиста. Вносимые в программу изменения часто являются источниками дополнительных ошибок. Избежать связанные с этим трудности позволяет использование специальных *средств отладки*, входящих в программное обеспечение современных ЭВМ.

При использовании средств отладки программист составляет специальное *задание на отладку* программы, в котором указывает, какие отладочные действия должны быть выполнены. Задание на отладку предусматривает вставление в текст программы отладочных операторов; выдачу на печать последовательности выполняемых машиной операторов и получаемых результатов — отладочный режим *прокрутка;* выдачу на печать информации

о последовательности выполнения операторов программы — отладочный режим *трассировка*, позволяющий проверить правильность реализации в программе предусмотренных алгоритмом разветвлений.

Полученные в результате отладочого выполнения программы данные анализируются, выявляются ошибки, в программу вносятся необходимые изменения. В соответствии с планом отладки вносятся изменения и в исходные данные.

После того как программа становится работоспособной, проводится ее *тестирование*, задачей которого является проверка правильности функционирования во всем диапазоне допустимых значений исходных данных. Из-за большого числа возможных сочетаний исходных данных провести исчерпывающее тестирование обычно не удается. Поэтому даже в длительно эксплуатируемой программе время от времени обнаруживаются те или иные ошибки.

Процесс изменения используемой программы вычислительной машины, обусловленный необходимостью устранения выявленных в ней ошибок или изменения ее функциональных возможностей, называется *сопровождением программы*. Сопровождение программ обычно выполняется их авторами или организациями-посредниками, осуществляющими распространение (продажу) программ.

Термин «сопровождение» обычно применяется в отношении программ, имеющих статус *программного изделия*. Основным отличием программного изделия от программ, разрабатываемых пользователями ЭВМ для собственных нужд или нужд своей организации, является ориентация на широкого потребителя и наличие разработанной в соответствии с требованиями ГОСТа программной документации.

Оформление программной документации. Составление и оформление программной документации ведется параллельно с разработкой и отладкой программ и является весьма ответственной частью работы по разработке программного изделия. Разработка и оформление программной документации осуществляются в соответствии с правилами, устанавливаемыми комплексом государственных стандартов, получивших общее название Единая система программной документации (ЕСПД). Состав и назначение ЕСПД устанавливается ГОСТ 19.001—77. Виды и содержание программных документов устанавливаются ГОСТ 19.101—77.

Сведения о составе и содержании основных документов, необходимых для разработки, изготовления и со-

провождения программ, приведены в табл. 3.3; сведения о составе и содержании эксплуатационных программных документов — в табл. 3.4. Состав программы и документации на нее определяется специальным документом, получившим название *спецификации*.

Таблица 3.3

Вид программного документа	Содержание программного документа
Техническое задание	Назначение и область применения программы, технические, технико-экономические и специальные требования, предъявляемые к программе, необходимые стадии и сроки разработки, виды испытаний
Пояснительная записка	Схема алгоритма, общее описание алгоритма и (или) функционирования программы, а также обоснование принятых технических и технико-экономических решений
Текст программы	Запись программы с необходимыми комментариями
Описание программы	Сведения о логической структуре и функционировании программы
Программа и методика испытаний	Требования, подлежащие проверке при испытании программы, а также порядок и методика их контроля.

Таблица 3.4

Вид эксплуатационного документа	Содержание эксплуатационного документа
Ведомость эксплуатационных документов	Перечень эксплуатационных документов на программу
Формуляр	Основные характеристики программы, комплектность и сведения об эксплуатации программы
Описание применения	Сведения о назначении программы, области применения, используемых методах, классе решаемых задач, ограничениях на применения, минимальной конфигурации технических средств
Руководство системного программиста	Сведения для проверки, обеспечения функционирования и настройки программы на условия конкретного применения

Продолжение табл. 3.4

Руководство программиста	Сведения для эксплуатации программы
Руководство оператора	Сведения для обеспечения процедуры общения оператора с вычислительной системой в процессе выполнения программы

ГОСТ допускает объединение отдельных документов с целью сокращения общего их числа. Конкретный состав документов на программное изделие в пределах допускаемых ГОСТом упрощений определяется при составлении и согласовании технического задания на программное изделие.

— для управления работой ЭВМ, распределения ее ресурсов, поддержания диалога с пользователями, оказания им помощи в разработке новых программ и выполнении работ, связанных с обслуживанием ЭВМ, и т. п.;

3.4. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Программное обеспечение ЭВМ — это совокупность разработанных для нее программ решения различных задач. Создание программного обеспечения — трудоемкая задача, требующая от исполнителей высокой квалификации и глубокого знания не только вычислительной техники и программирования, но и тех сфер деятельности, для которых предназначаются разрабатываемые программы. Этим определяется сравнительно высокая стоимость программного обеспечения, которое даже для одного рабочего места, оборудованного компьютером, часто превышает стоимость самого компьютера.

В программном обеспечении выделяют две группы программ, предназначенные:

— для решения прикладных задач, относящихся к физике, математике, конструированию и проектированию, медицинской диагностике и т. д.

Первая из указанных групп программ называется системным программным обеспечением, вторая — прикладным программным обеспечением.

Основная часть системного программного обеспечения, приобретаемая вместе с ЭВМ, называется *операционной системой* (ОС). Операционная система настолько тесно связана с оборудованием, что без нее современные ЭВМ практически не используются. Поэтому первой операцией, выполняемой после включения компьютера, является

загрузка операционной системы (рис. 3.9, *а, б*). В процессе загрузки в память ЭВМ помещаются основные программы ОС, способные управлять ее работой, воспринимать команды пользователей и обеспечивать их выполнение.

Одной из важнейших функций ОС является управление выполнением прикладнх задач, решаемых пользователями. Для того чтобы выполнить какую-либо программу, пользователь вводит соответствующую команду, которая воспринимается операционной системой. Операционная система загружает вызываемую программу в память и в дальнейшем следит за ходом ее выполнения (рис. 3.9, *в*). Если в процессе выполнения программы возникают ситуации, требующие вмешательства, ОС анализирует их и либо обеспечивает возможность дальнейшего выполнения программы, либо выдает сообщение о возникших затруднениях и указание, что необходимо сделать для продолжения выполнения программы. Например, она может потребовать включить и подготовить к работе какое-либо внешнее устройство, необходимое программе; установить или сменить дискету и т. п. Описанные функции выполняет одна из основных составляющих частей ОС, называемая управляющей программой.

Управляющая программа обеспечивает коллективную работу на ЭВМ нескольких пользователей — *коллективный доступ*, одновременное выполнение нескольких программ — *мультипрограммную обработку*, выполнение команд и программ пользователей, работающих

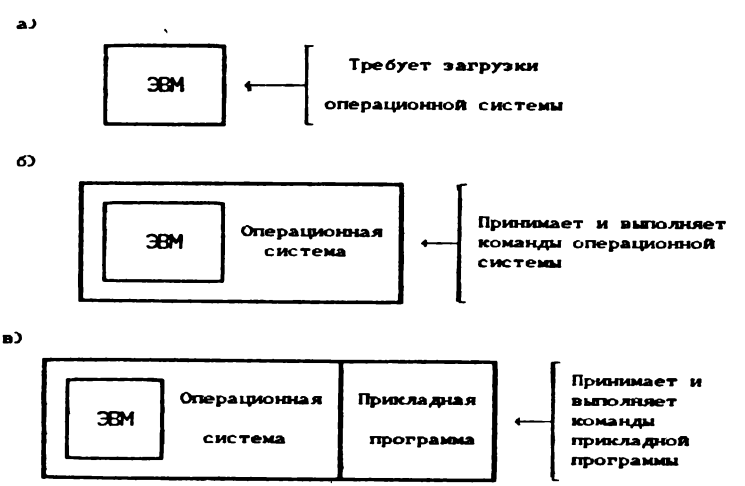


Рис. 3.9

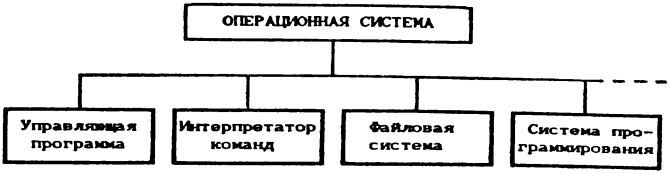


Рис. 3.10

с удаленных, соединенных с ЭВМ линиями связи терминалов — *телеобработку* и др.

Не каждый из указанных режимов может требоваться в конкретной ситуации использования той или иной ЭВМ. Наличие в составе управляющей программы модулей, обеспечивающих любую из дополнительных функций, делает программу более сложной, требует для ее размещения бóльшего объема памяти, может замедлить ее работу. Поэтому оптимальным является вариант, когда функциональные возможности управляющей программы и *ОС* в целом в максимальной степени соответствуют условиям использования ЭВМ в каждой конкретной ситуации. Процедура подготовки ОС для определенных условий работы называется *генерацией ОС*. В процессе генерации из полного набора программ ОС, называемого *дистрибутивным*, выбираются только те, которые необходимы для выполнения заказанных пользователем функций.

Кроме управляющей программы в составе ОС могут быть выделены следующие основные функциональные компоненты (рис. 3.10):

— интерпретатор команд пользователя;
— файловая система;
— система программирования;
— вспомогательные программы обслуживания.

Интерпретатор команд пользователя обеспечивает диалог пользователя с системой, воспринимает и расшифровывает (интерпретирует) его команды, обеспечивает их выполнение, вызывая необходимую в каждой конкретной ситуации программу. Вводя соответствующие команды, пользователь может получить информацию о состоянии устройств ЭВМ и архивов данных на дисках, запустить какую-либо задачу, приостановить или прекратить ее выполнение и т. д.

Файловая система представляет собой совокупность средств ОС, обеспечивающих выполнение операций поиска и ввода-вывода данных. С понятием файловая система ОС тесно связаны такие термины, как файл, файловая структура, организация данных, подробно рассматриваемые в гл. 5.

Система программирования представляет собой совокупность средств разработки программ для ЭВМ. Эта часть ОС представляется такими программными модулями, как трансляторы с алгоритмических языков программирования; средства, позволяющие объединять отдельные программные компоненты в комплексы программ, выполнить отладку и др.

Системы программирования называются *многоязыковыми*, если они позволяют разрабатывать отдельные части одной программы на разных языках, а затем объединять их в программный модуль. Многоязыковая система программирования является частью ОС, тесно с ней связана и широко использует средства ОС для реализации в программах операций ввода-вывода, управления вычислительным процессом и других функций.

Системы программирования могут быть и одноязыковыми. В этом смысле можно говорить, что программное обеспечение ЭВМ включает в себя несколько систем программирования. *Одноязыковые системы программирования* характерны для микроЭВМ.

Вспомогательные программы обслуживания, называемые *утилитами*, используются для выполнения различных функций по обслуживанию ЭВМ и ее основных устройств. Пользователи ЭВМ чаще используют утилиты, позволяющие осуществлять проверку качества лент и дисков, делать копии хранящихся на них данных, рассчитывать данные и вносить в них изменения и т. п.

Состав программного обеспечения и выполняемых функций ОС зависит от класса ЭВМ. Наиболее сложными являются операционные системы высокопроизводительных вычислительных систем. Например, для ЭВМ Единой системы наиболее широко используется операционная система, программное обеспечение которой включает сотни тысяч команд. Работают с такими ОС специально подготовленные системные программисты. Это, однако, не означает, что каждый работающий на ЭВМ должен знать все особенности и команды используемой операционной системы. Пользователи современных ЭВМ независимо от их типа работают за терминалами, мало отличающимися от персональных ЭВМ. Приемлемый для работающего за терминалом рядового пользователя уровень общения с ЭВМ обеспечивается входящими в состав ОС *диалоговыми системами коллективного пользования*. Причем доминирующей является тенденция унификации диалога пользователя с системой, которая дает

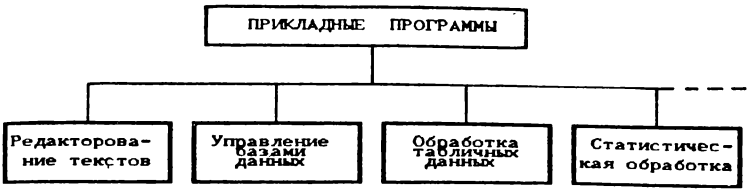


Рис. 3.11

возможность пользователям быстро адаптироваться к работе а терминалом новой для них ЭВМ.

Прикладное программное обеспечение современных ЭВМ насчитывает тысячи программ различных назначений. Прикладные программы первых ЭВМ в основном предназначались для решения математических задач. Затем стали разрабатываться и широко использоваться программы для создания баз данных, выполнения инженерно-конструкторских расчетов, проектирования, управления технологическими процессами и т. д. Сейчас, видимо, не существует таких сфер деятельности, в которых не применялись бы ЭВМ и для которых не были бы разработаны прикладные программы.

Практически каждое программное средство реализуется не как отдельная программа, а как комплекс (пакет) программ, предназначенных для решения класса задач, относящихся к какой-либо области. Поэтому прикладные программные средства называют также *пакетами прикладных программ* (ППП) или *программными пакетами.*

Современные прикладные программы разрабатываются как дружественные пользователю, взаимодействуют с ним в режиме диалога и не требуют в большинстве случаев умения программировать. Количество ППП, ориентированных на непрограммирующих пользователей, особенно возросло с появлением ПЭВМ. Среди прикладных программ для ПЭВМ по степени их использования выделились явные лидеры (рис. 3.11). К ним в первую очередь следует отнести текстовые и графические редакторы, с помощью которых готовятся различного рода документы, в том числе технические описания, служебные письма, таблицы, рисунки, схемы, графики и т. д. В составе прикладного программного обеспечения отечественных ПЭВМ используются текстовые редакторы ЛЕКСИКОН (ЕС1840, ЕС1841), ТЕКСТ (ИСКРА) и др. На отечественных ПЭВМ могут также использоваться редакторы, разработанные за рубежом (Word Star, UnvEditor) и др., позволяющие не только набирать символьные тек-

сты, но и включать в них математические формулы с их специальной символикой, рисунки, схемы.

Не менее широко используются ППП систем управления базами данных, среди которых на ПЭВМ наибольшую популярность получили различные версии dBase:

dBase — III, dBase — III — Plus, FoxBase, Clipper.

Широкий круг вычислительных задач, относящихся к финансовым и бухгалтерским расчетам, учетной деятельности и др., решается на ПЭВМ с использованием ППП табличной обработки данных SuperCalc (версии II, III, IV, V). Результаты вычислений, полученные с использованием ППП, могут представляться в графическом виде, удобном для дальнейшего анализа и использования.

Подбирая программное обеспечение, необходимо особенно заботиться о пополнении его программами, ориентированными на выполнение специфических функций того рабочего места, на котором используется ЭВМ. Это могут быть программы автоматизации проектирования, технологической подготовки производства, медицинской диагностики и т. д. Именно в таких программных средствах ощущается сейчас недостаток, но именно от того, насколько обеспечена ими ЭВМ, в большей степени зависит, получит ли ее владелец тот эффект, на который он рассчитывал, совершая столь дорогостоящую покупку.

Вопросы для самопроверки

3.1. В чем суть принципа программного управления?

3.2. Назовите основные структурные части команды ЭВМ.

3.3. Объясните разницу между одно-, двух- и трехадресными командами.

3.4. Что такое естественный порядок выполнения команд программы?

3.5. Объясните особенности выполнения команд условного и безусловного перехода.

3.6. Объясните назначение основных блоков, входящих в состав устройства управления (см. рис. 3.2).

3.7. Объясните взаимодействие основных блоков устройства управления при выполнении арифметической операции, операции условного перехода, операции безусловного перехода.

3.8. Объясните разницу между машинно-ориентированными и машинно-независимыми языками программирования.

3.9. Какие преимущества дает программирование на языках ассемблерного типа по сравнению с программированием в машинных кодах?

3.10. Назовите основной признак языков высокого уровня.

3.11. Объясните разницу между процедурно-ориентированными и универсальными языками программирования.

3.12. Объясните, что такое трансляция программы и какие функции выполняет программа-транслятор.

3.13. Назовите основные этапы подготовки программы для решения задачи на ЭВМ.

3.14. Поясните сущность этапа формализации задачи. В чем отличие математической модели задачи от ее исходного оригинала?

3.15. В чем состоит отличие двух понятий: «метод решения задачи», «алгоритм решения задачи»?

3.16. Назовите средства описания алгоритмов.

3.17. Что такое вспомогательный алгоритм?

3.18. В чем сущность метода последовательных уточнений?

3.19. Исходя из каких соображений выбирается язык программирования для записи текста разрабатываемой для ЭВМ программы?

3.20. Укажите назначение и перечислите основные возможности средств отладки программ.

3.21. В чем состоит особенность этапа тестирования программы?

3.22. Что такое сопровождение программы?

3.23. Объясните разницу между системным и прикладным программным обеспечением.

3.24. Назовите компоненты операционной системы и перечислите их основные функции.

Глава 4
ТИПЫ И СТРУКТУРЫ ДАННЫХ

4.1. ЭЛЕМЕНТАРНЫЕ ДАННЫЕ И ОПЕРАЦИИ, ВЫПОЛНЯЕМЫЕ НАД НИМИ

Каждый человек в повседневной жизни выполняет по известным правилам действия над целыми, дробными или вещественными числами, не нуждаясь в специальном указании о том, к какому типу они относятся, легко определяя его по внешнему виду числа.

При обработке в ЭВМ данные, будучи преобразованными к внутренним формам битового представления, теряют внешнюю индивидуальность и становятся неразличимыми по типу. Так, 2-байтовая последовательность битов (рис. 4.1, *а*) в зависимости от поставленной ей в соответствие формы представления может быть целым (рис. 4.1, *б*) или вещественным (рис. 4.1, *в*) числом, последовательностью двух символов (рис. 4.1, *г*) или кодом типа линии, являющейся элементом графического изображения (рис. 4.1, *д*). Таким образом, чтобы правильно обработать данные в ЭВМ, надо не только их ввести, но и сопроводить описанием.

Каждый алгоритмический язык программирования предоставляет пользователю набор различных типов элементарных данных, средства их описания и операторы обработки, обеспечивающие выполнение над данными тех или иных действий. Среди различных типов элементарных данных наиболее употребительными являются: целые числа (тип данных — ЦЕЛЫЕ); вещественные числа (тип данных — ВЕЩЕСТВЕННЫЕ); булевы величины (тип данных — БУЛЕВЫ); литеры (тип данных — ЛИТЕРЫ); цепочки литер (тип данных — СТРОКИ).

Алгоритмические языки предоставляют возможность ссылаться на элемент данных по присвоенному *имени* или *идентификатору*. Идентификатор записывается в виде последовательности алфавитно-цифровых символов, начинающейся с буквы. Длина идентификатора обычно

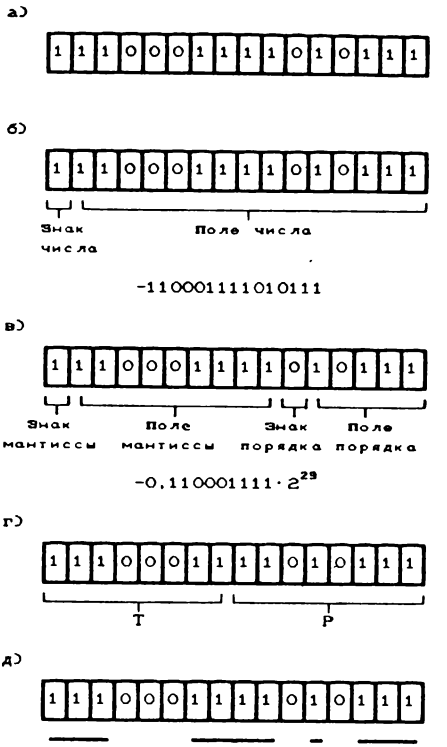


Рис. 4.1

ограничена 6—8 символами. Ниже приведены примеры возможной записи идентификаторов данных:

A, B1, NOMER, KTO _ TAM, C1. TXT

В следующей строке записаны идентификаторы, которые в большинстве алгоритмических языков воспринимаются как неправильные:

12AB3, KTO TAM, .TXT

Первый и третий идентификаторы не верны, так как начинаются не с буквы. Второй идентификатор содержит символ «пробел», который, наряду с некоторыми специальными символами, обычно запрещается для использования в идентификаторах данных.

Компилятор, транслирующий программу с алгоритмического языка, связывает имя элемента данных с определенным адресом памяти ЭВМ, по которому в процессе

выполнения программы хранится значение *именованного элемента данных*, что освобождает программиста от необходимости знать этот адрес.

Среди именованных элементов данных различают константы и переменные. *Константа*, получая свое значение, сохраняет его неизменным в течение всего времени выполнения программы. *Переменная* может изменять свое значение в результате ввода данных в ЭВМ, копирования их с одного устройства памяти на другое, выполнения операции присваивания нового значения.

Операции обработки, выполняемые над элементарными данными типов ЦЕЛЫЕ, ВЕЩЕСТВЕННЫЕ, хорошо известны из школьного курса математики. Другие, выполняемые, например, над данными типа БУЛЕВЫ, хотя и давно известны, широко начали применяться лишь в связи с развитием использования вычислительной техники. Операции над цепочками символов разработаны в информатике специально для автоматизированной обработки текстов. Ниже кратко рассматриваются особенности каждого из основных типов элементарных данных и выполняемых над ними действий.

Тип данных ЦЕЛЫЕ — представляется множеством допустимых для каждой конкретной ЭВМ значений целых чисел. Примеры записи целых чисел:

−85 1917 1985 +124867

Знак (+) при записи целых чисел, как видно из примеров, можно опустить.

К операциям, которые выполняются над целыми, обычно относятся: сложение (+); вычитание (−); умножение (*); целочисленное деление (\ или div); вычисление остатка (mod).

В скобках за названиями операций здесь и далее указаны символы-обозначения соответствующих операций, используемые в алгоритмических языках программирования. В приведенном перечне две последние операции нуждаются в кратком пояснении.

Целочисленное деление называют еще делением с отбрасыванием остатка, что вполне объясняет суть этой операции:

```
 5 div 2=2
17 \ 3=5
```

Результатом операции *вычисления остатка* является остаток, отбрасываемый при целочисленном делении:

```
 5 mod 2=1
17 mod 3=2
```

Элементарные данные типа ЦЕЛЫЕ обычно представляются в ЭВМ в форме с фиксированной запятой (см. § 1.4). При этом в памяти ЭВМ хранится двоичное значение соответствующего целого десятичного числа. Но двоичные целые во многих задачах имеют самостоятельное значение и могут средствами алгоритмического языка представляться не только их десятичными значениями, но и значениями, записанными в двоичной, восьмеричной или шестнадцатеричной системах счисления. Поэтому в типе данных ЦЕЛЫЕ выделяют элементарные подтипы: ЦЕЛЫЕ ДВОИЧНЫЕ (BINARY), ЦЕЛЫЕ ВОСЬМЕРИЧНЫЕ (OCTAL), ЦЕЛЫЕ ШЕСТНАДЦАТЕРИЧНЫЕ (HEXADECIMAL). В скобках указаны английские эквиваленты названий соответствующих подтипов данных, используемые в алгоритмических языках программирования.

Чтобы различать значения целых подтипов, при записи их дополняют соответствующими обозначениями. Так, в конце двоичного целого значения можно добавить букву В (тип BINARY):

1101B, 101110001110B

Эти же числа в восьмеричном представлении можно записать как:

15&O, 5616&O

где отделенная от значения знаком (&) буква О указывает, что данное число является восьмеричным целым (тип OCTAL).

Самую короткую запись двоичных кодов обеспечивает их шестнадцатеричное представление (см. табл. 1.2), отмечаемое дополнительной буквой Н (тип HEXADECIMAL):

D8H, B8EH

Следует иметь в виду, что приведенные здесь обозначения, используемые для записи двоичных, восьмеричных и шестнадцатеричных целых, не являются общепринятыми. Каждый из алгоритмических языков может иметь собственные обозначения для целых подтипов.

К целым подтипам применяются те же операции, что и для целых десятичных. Дополнительные возможности использования целых подтипов обеспечиваются применением логических операций побитовой обработки, которые рассматриваются далее.

Тип данных ВЕЩЕСТВЕННЫЕ для конкретной ЭВМ представляется множеством значений диапазона представления чисел в форме с плавающей запятой. В системе автоматизированной обработки данных и алгоритмических языках, так же как и в математике, используются две формы записи вещественных чисел: естественная и нормальная.

При естественной форме записи в отличие от традиционной целая часть от дробной отделяется не запятой, а точкой:

3.1415 +0.65 −1644.182

Нормальная форма чаще используется в научных задачах, так как она более удобна для записи малых и больших чисел:

+1.1581E12 1.18699714E−5 −118.699714E−8

В отличие от традиционной записи для указания порядка вещественных чисел в естественной форме используется буква E вместо основания степени 10. Таким образом, приведенным примерам чисел соответствуют следующие значения в традиционной форме записи:

$+1{,}1581 \cdot 10^{12}$ $-1{,}18699714 \cdot 10^{-5}$ $-118{,}699714 \cdot 10^{-8}$

Как видно из приведенных примеров, одно и то же значение в нормальной форме можно записать по-разному.

Над вещественными числами выполняются следующие операции: сложение (+), вычитание (−), умножение (*), деление (/), возведение в степень (**), (↑) или (∧).

Несколько непривычно выглядит запись операции возведения в степень:

A**B 5↑2 Б∧3,

что соответствует следующим примерам традиционной записи этой операции:

a^b, 5^2, b^3.

В большинстве алгоритмических языков для обработки вещественных чисел имеются так называемые *встроенные функции*, позволяющие вычислять значения наиболее распространенных математических функций:

COS(X) — косинус, $\cos x$;
LOG(X) — логарифм натуральный, $\ln x$;

SQRT(X) — корень квадратный, $\sqrt{x}$;
и т. д.

Аппарат встроенных функций существенно расширяет возможности обработки данных и повышает эффективность программирования.

Из числовых констант переменных и математических функций можно составлять арифметические выражения:

```
1.4*X**(2./3)+COS(X);
A*B/C/D*E/F.
```

Используя традиционную математическую символику, эти же выражения можно записать следующим образом:

$1{,}4\sqrt[3]{x^2}+\cos x;$

$\frac{abe}{cdf}.$

Арифметические выражения могут включать константы и переменные только одного типа (целые, вещественные) или обоих типов. Во втором случае из описания алгоритмического языка важно уяснить, что является результатом выполнения арифметической операции над операндами, относящимися к разным типам, так как небрежное отношение к типу данных при использовании алгоритмических языков может стать источником серьезных и порой трудно обнаруживаемых ошибок. Это можно проиллюстрировать, используя первое из приведенных выше арифметических выражений, записанных в соответствии с правилами алгоритмического языка ФОРТРАН. Если переписать это выражение, опустив точку после числа 2, т. е. записать его в виде

```
1.4*X**(2/3)+COS(X),
```

то ему фактически будет соответствовать совершенно другое арифметическое выражение в традиционной записи, а именно:

$1{,}4+\cos x.$

Объясняется это тем, что в языке ФОРТРАН запись (2/3) соответствует операции целочисленного деления, результатом которого является целое число (0), а $x^0=1$ при любом значении x. Запись же (2./3) выполняется как деление вещественных чисел (один из операторов вещест-

венный) и результатом его является значение 0,66..., что и обеспечивает правильность вычисления выражения для различных значений x.

Тип данных БУЛЕВЫ принимает всего два значения — «истинно» и «ложно».

Для обозначения значений булевых величин в алгоритмических языках обычно используется двухбуквенный алфавит с условием: T — «истинно», F — «ложно» (от англ. TRUE и FOLSE). В ЭВМ значения булевых величин чаще представляются в двоичном алфавите с условием: **1** — «истинно», **0** — «ложно». Простейшим примером булевого выражения является проверка отношения между двумя арифметическими выражениями:

```
(A**2+B)>(C*D-F1).
```

Если при конкретных значениях переменных A, B, C, D, F1 значение (A**2+B) действительно больше, чем (C*D−F1), то говорят, что данное отношение *истинно*, т. е. оно имеет значение T, в противном случае приведенное булево выражение *ложно* и имеет значение F.

В алгоритмических языках наиболее распространены следующие обозначения для операций отношения:

больше (>), (GT);	больше или равно (> =),(= >); (GE);
меньше (<), (LT);	меньше или равно (< =),(= <); (LE);
равно (=), (EQ);	не равно (< >),(NE).

С помощью операций отношения в программах реализуются проверки логических условий, необходимые для управления ходом вычислительного процесса. Допустим, что надо вычислить значение по формуле

$$y=\begin{cases}\dfrac{2ab}{a-b}, & \text{если } a\neq b;\\ c\ , & \text{если } a=b.\end{cases}$$

Алгоритм соответствующего вычислительного процесса (рис. 4.2) требует проверки отношения между переменными a и b.

При разработке программ на ЭВМ часто приходится проверять и более сложные условия. Допустим, необходимо реализовать алгоритм (рис. 4.3), требующий проверки совокупности простых условий. В этом случае и подобных ему можно использовать *логические операции*, определенные над булевыми переменными и позволя-

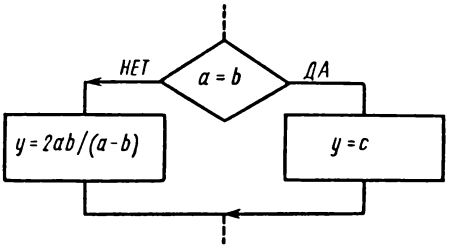


Рис. 4.2

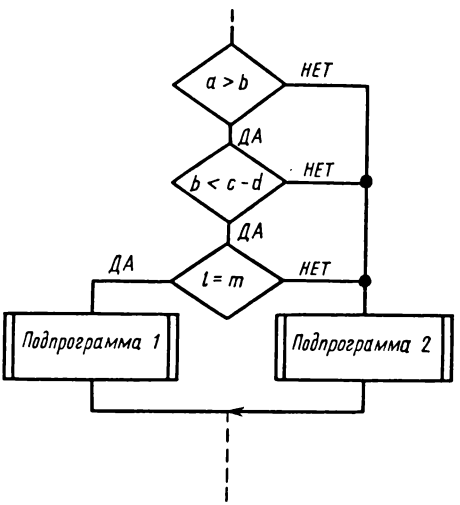


Рис. 4.3

ющие записывать логические условия любой сложности. Основными логическими операциями, определенными над величинами типа БУЛЕВЫ, являются: логическое сложение ($\vee$), (OR); логическое умножение ($\wedge$), (AND); логическое отрицание ($-$), (NOT).

Операция логического сложения для форм буквенного и числового представления булевых значений задается правилами:

а) $F \vee F = F$, $T \vee F = T$, $F \vee T = T$, $T \vee T = T$;
б) $0 \vee 0 = 0$, $1 \vee 0 = 1$, $0 \vee 1 = 1$, $1 \vee 1 = 1$.

Результат операции логического сложения *истинен*, если хотя бы один из операндов *истинен* (или один, или другой). Поэтому ее также называют операцией ИЛИ.

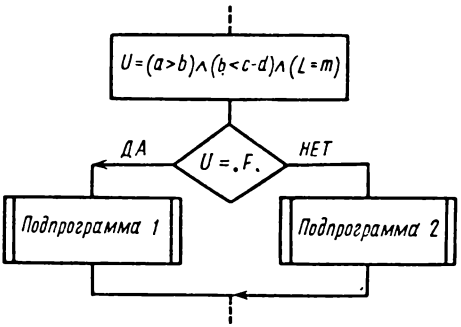


Рис. 4.4

Для операции логического умножения (операции И) используются правила:

а) $F \wedge F = F$; $T \wedge F = F$; $F \wedge T = F$; $T \wedge T = T$;
б) $0 \wedge 0 = 0$; $1 \wedge 0 = 0$; $0 \wedge 1 = 0$; $1 \wedge 1 = 1$.

Результат этой операции *истинен*, когда *истинны* оба операнда (и один, и другой).

Операция логического отрицания (операция НЕ) меняет булево значение операнда на противоположное:

а) $\neg F = T$, $\neg T = F$;
б) $\neg 0 = 1$, $\neg 1 = 0$.

Пользуясь логическими операциями, можно записать последовательность простых условий приведенного на рис. 4.3 алгоритма в виде одного логического выражения:

$$(a > b) \wedge (b < c - d) \wedge (l = m). \qquad (4.1)$$

Предварительно вычислив это выражение, можно присвоить его значение булевой переменной с идентификатором, допустим, *U*, и существенно упростить исходный алгоритм (рис. 4.4).

Опустим скобки в записи логического выражения (4.1):

$$a > b \wedge b < c - d \wedge l = m.$$

Записанное таким образом логическое выражение остается тем же и представляет собой последовательность переменных, соединенных знаками операций арифметических, логических и отношения. Однозначность вычисления подобных логических выражений обеспечивается установленным общим порядком выполнения входящих в них операций: вычисление выражений в скобках;

вычисление математических функций; возведение в степень; умножение, деление; целочисленное деление; нахождение остатка; сложение, вычитание; операция отношения; логическое отрицание; логическое умножение; логическое сложение. Руководствуясь перечнем, следует помнить, что операции равного приоритета выполняются в порядке их следования слева направо.

Очень полезными могут оказаться представляемые некоторыми алгоритмическими языками возможности выполнения логических операций над целыми и их подтипами: двоичными, восьмеричными, шестнадцатеричными. Поскольку каждый из перечисленных подтипов, включая целое десятичное, имеет одну и ту же форму внутримашинного представления — целое двоичное значение, результат применения логических операций к целым не зависит от подтипа, а определяется лишь их значениями.

Важной особенностью выполнения логических операций над целыми является их побитовая реализация. Суть ее состоит в том, что каждый двоичный разряд внутримашинного представления целого рассматривается как отдельное значение булевого типа. Итогом выполнения логической операции над целыми является двоичный код, получаемый в результате независимого ее выполнения над соответствующими разрядами операндов.

Пример 4.1. Определить результат выполнения операции:

$61 \wedge 15$.

Решение. Переведем значения 61 и 15 в двоичную систему счисления (см. 1.3):

$61_{(10)} = 1\ 1\ 1\ 1\ 0\ 1_{(2)}$;
$15_{(10)} = 0\ 0\ 1\ 1\ 1\ 1_{(2)}$.

Выполнив поразрядно над операндами операцию логического умножения, получим код $0\ 0\ 1\ 1\ 0\ 1_{(2)}$, соответствующий десятичному значению 13.

Ответ: 13.

Пример 4.2. Вычислить: $98 \wedge 61$

Решение:

$98_{(10)} = 1\ 1\ 0\ 0\ 0\ 1\ 0_{(2)}$

$$\begin{array}{r} 1\ 1\ 0\ 0\ 0\ 1\ 0 \\ 0\ 1\ 1\ 1\ 1\ 0\ 1 \\ \hline 0\ 1\ 0\ 0\ 0\ 0\ 0_{(2)} \end{array} \begin{array}{l} \wedge\ 98_{(10)} \\ \quad 61_{(10)} \\ \hline = 32_{(10)} \end{array}$$

Ответ: 32.

Пример 4.3. Вычислить: 98 ∨ 61.
Решение:

$$\begin{array}{l} 1\,1\,0\,0\,0\,1\,0 \\ \vee\ 0\,1\,1\,1\,1\,0\,1 \\ \hline 1\,1\,1\,1\,1\,1\,1_{(2)} \end{array} \quad \begin{array}{r} 98_{(10)} \\ \vee\ 61_{(10)} \\ \hline = 255_{(10)} \end{array}$$

Ответ: 255.

Пример 4.4. Вычислить: ¬98
Решение.

$$\begin{array}{l} \neg\ 1\,1\,0\,0\,0\,1\,0 \\ \hline 0\,0\,1\,1\,1\,0\,1_{(2)} \end{array} \quad \begin{array}{r} \neg 98_{(10)} \\ \hline = 29_{(10)} \end{array}$$

Ответ: 29.

Двоичные коды, представляемые в программах как целые подтипы, часто используются для задания элементов различных множеств и кодирования их свойств. Элементами таких множеств могут быть места в кинотеатре или в вагоне поезда, элементами подмножеств — удобные места и неудобные, нижние и верхние спальные места в вагоне и т. д. Многообразие возможностей применения двоичных кодов и логических операций над ними здесь проиллюстрировать трудно, поэтому ограничимся следующим шутливым примером.

Допустим, что в брачном бюро ведется картотека на 15 женихов. Степень заполненности картотеки будем характеризовать переменной GEN, значения которой (как и других переменных этого примера для удобства перевода из двоичной системы счисления будем записывать в восьмеричном представлении. Каждому жениху, сведения о котором хранятся в картотеке, в 15-битовом двоичном коде GEN поставим в соответствие один разряд: первому жениху — первый разряд, второму — второй и т. д. Когда картотека пуста, все разряды двоичного кода GEN равны 0:

$$\underbrace{0\ 0\ 0}_{0_{(8)}}\ \underbrace{0\ 0\ 0}_{0_{(8)}}\ \underbrace{0\ 0\ 0}_{0_{(8)}}\ \underbrace{0\ 0\ 0}_{0_{(8)}}\ \underbrace{0\ 0\ 0_{(2)}}_{0_{(8)}}$$

GEN = 0 0 0 0 0 2 O

Если картотека заполнена, то в каждом из пятнадцати разрядов двоичного кода GEN будет записана 1:

$$\underbrace{1\ 1\ 1}_{7}\ \underbrace{1\ 1\ 1}_{7}\ \underbrace{1\ 1\ 1}_{7}\ \underbrace{1\ 1\ 1}_{7}\ \underbrace{1\ 1\ 1_{(2)}}_{7_{(8)}}$$

GEN = 7 7 7 7 7 & O:

Проанализировав запросы потенциальных невест, обращающихся в бюро (хочу с голубыми глазами, спортсмена, высокого роста и т. п.), введем еще несколько переменных, характеризующих женихов. Значения двоичным разрядам вводимых переменных присвоим на основании данных картотеки, подтверждающих (значение 1) или отрицающих (значение 0) наличие у женихов соответствующих качеств:

GGG — женихи с голубыми глазами
101100010100100

GGG=5 4244&O

(Двоичное значение GGG указывает, что голубые глаза у женихов, числящихся в картотеке под номерами 1, 3, 4, 8, 10, 13);
GSP — спортсмены
001101000110100
GSP=1 5064&O;
GRB — любители рыбной ловли:
101001000010001
GRB=5 1021&O;
GCU — курящие:
10101010101011101,
GCU=5 2535&O.

Очевидно, что этот список можно было бы продолжать, так как количество необходимых переменных определяется количеством вопросов анкеты, на которые жених или невеста, становясь на учет, должны дать ответ ДА или НЕТ.

Допустим, требуется подготовить ответ клиентке, которая хотела бы получить сведения о женихах — спортсменах с голубыми глазами, некурящих и не рыболовах. Чтобы получить код, указывающий на наличие в картотеке жениха, удовлетворяющего предъявленным клиенткой требованиям, надо вычислить следующее логическое выражение:

$$GGG \wedge GSP \wedge \neg GRB \wedge \neg GCU$$

Если правильно выполнить все входящие в логическое выражение действия, то можно получить в результате число 4040&O, которому соответствует двоичный код:

0 0 0 100000100000

Таким образом, можно направить клиентке сведения о женихах, числящихся в картотеке под номерами 4 и 10.

Задач, подобных рассмотренной, на практике встречается очень много: обмен квартир, трудоустройство, со-

ставление расписаний и т. п. Для их решения на современных ЭВМ используются весьма эффективные средства программирования, среди которых важное место, наряду с алгоритмическими языками, занимают различные информационно-поисковые системы, системы управления базами данных и другие, имеющие развитые средства логической обработки, в том числе данных, представляемых в двоичных кодах.

Тип данных ЛИТЕРЫ представляется множеством литер символьного набора ЭВМ и обычно включает в себя строчные и прописные буквы русского и латинского алфавитов, цифры от 0 до 9, символы псевдографики и др. Элемент данных литерного типа обычно занимает в памяти ЭВМ один байт, хранящий битовую кодовую комбинацию символа в используемой системе кодирования (ASCII, КОИ-8, ДКОИ и др.).

При записи констант литерные значения заключаются в апострофы:

'A' 'Т' '+' ' ' ''''

Здесь предпоследняя константа задает литеру «пробел», а последняя — литеру «апостроф», которая записывается дважды, для того, чтобы быть отличимой от апострофов, ограничивающих литерное значение.

Применительно к литерам могут применяться логические операции отношения. Это имеет важное практическое значение при обработке текстов, так как отношение упорядочения, заданное на множестве символов-букв, соответствует их алфавитному порядку, при котором «большей» считается литера, дальше отстоящая от начала алфавита, т. е. верными являются, например, следующие отношения:

'А'<'Б', 'Ф'>'А'.

Прописная буква «меньше», чем соответствующая строчная:

'А'<'а', 'Ф'<'ф'.

Литеры-цифры «меньше» литер-букв и упорядочиваются между собой по числовому значению:

'A'>'9' '1'<'2'.

Реализация операций отношения применительно к литерным значениям обычно основана на том, что заданной последовательности упорядочения множества литер

ЭВМ соответствует последовательность возрастающих числовых значений кодов, представляющих эти литеры в памяти ЭВМ. Так, например, в КОИ-7 латинской литере A соответствует код 65, литере B — код 66, литере C — код 67 и т. д. Это дает возможность при реализации операций отношения над литерами сравнивать числовые значения соответствующих им кодов.

Для упорядоченного набора литер во многих алгоритмических языках используются функции, позволяющие по известному значению кода находить значение соответствующей литеры — функция CHR (в некоторых случаях CHR$) и, наоборот, по известному значению литеры получать ее код — функция ASC (используются и другие обозначения). Приведем несколько примеров, поясняющих функции CHR и ASC:

Аргумент	Функция	Значение функции
65	CHR (65)	A
66	CHR (66)	B
A	ASC (A)	65
B	ASC (B)	66

Тип данных СТРОКИ представляется множеством цепочек литер. Цепочка литер как элемент данных типа СТРОКИ обычно занимает в памяти ЭВМ поле памяти, длина которого в байтах на единицу превышает число литер цепочки. Дополнительный байт отводится для хранения атрибута длины, указывающего число литер в цепочке (не более 255, так как $2^8 - 1 = 255$).

При записи литерные значения заключают в апострофы:

'ИНФ', 'БЕЛЯЕВ', '121', ''.

Последнее из приведенных значений литерной цепочки является «пустым», т. е. не содержит ни одной литеры. Пустому значению литерной строки соответствует атрибут длины, равный нулю. Чтобы получить доступ к атрибуту длины литерной цепочки, программист использует специально предназначенную для этого функцию (LENGTH — в языке ПАСКАЛЬ, LEN — в некоторых версиях языка БЕЙСИК), аргументом которой является элемент данных типа СТРОКИ, а значением — соответствующий атрибут длины:

LEN ('БОРИСОВ') = 7

Элементы данных типа СТРОКИ можно сравнивать между собой, применяя к ним операции отношения. Две строки сравниваются последовательно литера за лите-

рой, начиная слева. Сравнение заканчивается сразу же, как только обнаруживаются две неравные литеры. Строка, содержащая «бóльшую» очередную литеру, считается «большей». Добавление к одной из двух равных строк любого дополнительного символа делает ее «больше» второй строки.

Ниже приведены примеры истинных логических выражений, иллюстрирующих правила сравнения цепочек литер:

```
'АНДРЕЕВ'<'БЕЛЯЕВ'
'Андреев'  >'АНДРЕЕВ'
'Fortran'  >'Fort'
'СТРОКА'  ='СТРОКА'
'121'      <'125'
'12'       >'115'      (!)
```

Два последних примера иллюстрируют отличие сравнения строковых значений от сравнения чисел.

К символьным строкам можно применить *операцию сцепления*, обозначаемую символом (+) или символом (||):

```
'ИНФОР'+'МАТИКА'='ИНФОРМАТИКА'
'18'||'84'='1884'
```

Операцию сцепления литерных цепочек называют также *операцией конкатенации*. В языке программирования ПАСКАЛЬ она реализуется специальной функцией (CONCAT), аргументами которой являются сцепляемые литерные цепочки, а результатом — сцепленная цепочка:

```
CONCAT ('A18', '84', '.DOC')='A1884.DOC'
```

Среди других функций, полезных для обработки строковых данных, отметим следующие:

Функция POS позволяет определить вхождение одной цепочки литер (первый аргумент) в другую цепочку (второй аргумент). Результатом является номер литеры второй строки, начиная с которой найдено ее совпадение с литерами первой строки:

```
POS ( 'С', 'ПАСКАЛЬ')=3,
POS ( 'А', 'ПАСКАЛЬ')=2,
POS ( 'П', 'ПАСКАЛЬ')=0,
POS ('КА', 'ПАСКАЛЬ')=4.
```

Функция COPY формирует подстроку (результат) из заданных: строки (первый аргумент); номера символа (второй аргумент), с которого начинается подстрока; ко-

личества символов (третий аргумент), переносимых (копируемых) в подстроку:

COPY (ПАСКАЛЬ, 1, 3)='ПАС',
COPY (ПАСКАЛЬ, 4, 2)= 'КА'.

К литерным строкам можно применить и операторы, обеспечивающие удаление заданного количества символов в строке (DELETE), вставление заданной подстроки в строку (INSERT) и др.

4.2. СРЕДСТВА ОПИСАНИЯ ЭЛЕМЕНТАРНЫХ ДАННЫХ

Тип элементарных данных в программах можно указать явно или неявно.

Неявное задание типа данных основано на использовании специальных соглашений об отличиях в формах записи значений данных, а также в правилах записи идентификаторов переменных и констант различных типов. Поэтому неявное описание называют также *описанием типов данных по соглашению*.

В соответствии с подобными соглашениями наличие десятичной точки в записи числовых значений позволяет отнести их к типу данных ВЕЩЕСТВЕННЫЕ:

−14.15 52. .52 +2.1416

а отсутствие десятичной точки — к типу данных ЦЕЛЫЕ:

−1415 52 311416

В алгоритмических языках ФОРТРАН и ПЛ/1 переменные, имена которых начинаются с букв I, J, K, L, M, N, относятся к типу данных ЦЕЛЫЕ:

NOMER, KOL1, M, JOTA,

а переменные, имена которых начинаются с любой из оставшихся букв алфавита,—к типу данных ВЕЩЕСТВЕННЫЕ:

A1, X, YMAX, GAMMA, ALFA.

В некоторых версиях языка БЕЙСИК тип переменной указывается посредством специального символа в конце имени (табл. 4.1).

Таблица 4.1

Символ в конце имени	Тип переменной	Примеры
%	Целый	A%, B%
!	Вещественный обычной точности	A!, C!
≠	Вещественный удвоенной точности	A≠, D≠
$	Строка символов	A$, F$

С помощью тех же символов при необходимости уточняются типы констант:

5! — вещественное обычной точности;
57846.1278≠ — вещественное удвоенной точности.

Явное задание типа данных основано на использовании специальных операторов, которые обычно содержат список описываемых переменных и ключевое слово, определяющее их тип. В табл. 4.2 приведены наиболее употребительные ключевые слова, используемые в алгоритмических языках для описания типа данных.

Таблица 4.2

Тип данных	Ключевое слово
ЦЕЛЫЕ	INTERGER
ВЕЩЕСТВЕННЫЕ	REAL
БУЛЕВЫ	BOOLEAN
ЛИТЕРЫ	CHAR
СТРОКИ	STRING

Для каждого типа данных в алгоритмическом языке определяется стандартная длина полей памяти, отводимых для хранения элементов данных. Если же данные имеют нестандартную длину, то ее необходимо указать в описании. Например, в языке ФОРТРАН нестандартную длину можно указать после имени описываемой переменной в виде *S, где S — количество байтов памяти, которое требуется отвести для хранения значения переменной:

A*8, N*2.

В приведенных примерах длина 8 байт, указанная для переменной, описывает ее как вещественную удвоенной точности, а длина 2 байт для целой переменной N является укороченной, так как стандартная длина представле-

ния в памяти ЭВМ, как целых, так и вещественных данных в ФОРТРАН принята равной 4 байт. Указатель нестандартной длины может записываться и в конце ключевого слова, описывающего тип данных, и тогда он относится ко всем переменным, описываемым с помощью этого ключевого слова.

Иногда для указания нестандартной длины используются дополнительные ключевые слова: DOUBLE — удвоенная точность, SHORT — укороченный формат и др.

Рассмотрим несколько примеров и кратко прокомментируем их.

Пример 4.5. Описание элементарных данных на языке ФОРТРАН:

```
INTERGER*2 K, K1, R1, L2
INTERGER SUMMA
REAL NOMER, A15*8
```

В приведенных операторах переменные K, K1, R1, R2 описываются как целые нестандартной длины (2 байт), переменная SUMMA — как целая стандартной длины (4 байт), переменная NOMER — как вещественная обычной точности, переменная A15 — как вещественная удвоенной точности.

Пример 4.6. Описание элементарных данных на языке ПАСКАЛЬ:

```
const SC = -1; MAX = 2000;
  var SUM, YEAR : INTERGER;
      VIS : BOOLEAN;
      NOMBRE : STRING;
```

Здесь описываются две константы — SC и MAX (им предшествует ключевое слово const), целые переменные SUM и YEAR, переменная VIS типа БУЛЕВЫ и переменная NOMBRE типа СТРОКИ. Описание переменных начинается с ключевого слова var (от англ. VARIABLE — переменная).

Пример 4.7. Описание элементарных данных на языке ПЛ/1:

```
DECLARE PI   FIXED (5, 4) INIT (3.1416),
        PROIZ  FLOAT (7),
        NAME   CHARACTER (15);
```

Описание данных начинается с ключевого слова DECLARE (англ. ОБЪЯВЛЯТЬ). Переменная PI описывается как вещественная, задаваемая в форме с фиксированной запятой (на это указывает ключевое слово FIXED), причем значение PI в соответствии с указателем (5,4) выражается пятью цифрами, из которых четыре представляют дробную часть. Переменной PI присваивается начальное значение (3.1416), что указывается ключевым словом INIT.

Переменная PROIZ описывается как вещественная, задаваемая в форме с плавающей запятой (ключевое слово FLOAT), точность

представления мантиссы — 7 значащих десятичных цифр (указатель 7).

Переменная NAME описывается как цепочка из 15 символов. Заметим, что, хотя переменная PI и получила начальное значение, она не является константой и в процессе выполнения программы в принципе может изменять свое значение, тогда как переменные SC и MAX, описанные в примере 4.4 как константы, в программе на языке ПАСКАЛЬ на могут изменять заданных описанием значений.

Стандартная длина представления вещественных данных в ЭВМ, если она равна, например, 4 байт, обеспечивает хранение вещественных чисел с точностью до семи значащих цифр. Для входных и выходных данных многих практических задач эта точность является излишней. Представьте себе, что статистическая обработка данных о росте мальчиков в возрасте 11 лет дает машинный результат — 1,391617 м. Очевидно, что можно удовлетвориться значением 1,39 м. При решении других задач это же значение, возможно, удобнее получить из машины в одной из следующих форм:

```
1.3916        0.139E1        139.16E-2
```

Приведенные варианты записи значения одной и той же переменной дают представление о возможных *форматах* внешнего представления данных (на входе и выходе программы), которые задаются посредством специальных описателей, называемых *спецификациями формата.*

Иногда, как это было показано в примере 4.5, алгоритмические языки позволяют совместить описание типа данных и формата их внешнего представления. Так, используя средства описания алгоритмического языка ПЛ/1, можно описать переменную для хранения значения роста (1,39) следующим образом:

```
DECLARE  ROST  FIXSED (3,2);
```

В этом примере переменная ROST описывается как вещественная и имеет значение, представляемое тремя цифрами, из которых две последние — значение дробной части. Здесь спецификация формата (3,2) является составной частью описания переменной.

Указатели или спецификации формата используются и отдельно от описания типа данных. Необходимость в этом обычно возникает при описании входного потока данных. Допустим, что программа вводит в память ЭВМ значения трех переменных, две из которых описаны как вещественные и одна как целая:

```
A=-15.68, B=13.84E+12, N=16.
```

Если значения переменных вводятся сплошным потоком, т. е. символ за символом без всяких разделителей, то на вход ЭВМ поступит последовательность символов:

−15.6813.84E+1216

Очевидно, что существует множество вариантов разбить эту последовательность на два вещественных и одно целое число, например такие:

−15.6	813.94E+1	216
−15.681	3.84E+12	16

Однозначная интерпретация данных входного потока обеспечивается указанием для каждого элемента данных формата его представления. Например, средствами алгоритмического языка ФОРТРАН формат данных приведенного выше входного потока можно описать следующим списком указателей формата:

F6.2, E9.2, I2.

Спецификация F6.2 указывает на вещественное число, записанное в естественной форме (на это указывает буква F) и представленное шестью символами (включая символы десятичной точки и знака), из которых два последних — значение дробной части.

Спецификация E9.2 указывает на вещественное число, записанное в нормальной форме (с мантиссой и порядком). Всего в записи числа 9 символов, из которых два отведены под дробную часть мантиссы.

Спецификация I2 указывает на целое число, значение которого записывается двумя символами.

Подобные спецификации формата с некоторыми отличиями в правилах записи используются практически во всех языках программирования.

Еще один прием указания формата, используемый, в частности, в различных версиях алгоритмического языка БЕЙСИК, можно проиллюстрировать следующими примерами:

192.44	«≠≠≠.≠≠»
1984	«≠≠≠≠»
−2.35E+02	«≠≠.≠≠∧∧∧∧»

Здесь в левой колонке записаны числовые значения различных типов, а в правой — соответствующие указатели формата, называемые *шаблонами*. Символы (≠) указывают позиции цифр в целой и дробной частях числа

(мантиссы). Символы (∧ ∧ ∧ ∧) указывают позиции, отводимые для порядка вещественного числа, записанного в нормальной форме.

В заключение уточним роль каждого из рассмотренных средств описания элементарных данных:

1. Явное или неявное описание типов переменных определяет формы их внутреннего представления в памяти ЭВМ и обеспечивает выполнение над значениями действий по правилам, установленным для элементарных данных соответствующего типа;

2. Указание формата для входных и выходных данных программы обеспечивает правильное преобразование значений соответствующих переменных из форм внешнего в формы внутреннего представления и обратно.

Следует также иметь в виду, что большинство алгоритмических языков имеют специальные встроенные функции, позволяющие преобразовывать данные из одной формы внутреннего представления в другую. Так, значения переменных целого типа можно преобразовать в действительные значения, последовательность символов-цифр, определенную в программе как тип СТРОКИ,— в значение типа ЦЕЛЫЕ и ВЕЩЕСТВЕННЫЕ и т. д.

4.3. РАСШИРЕНИЕ ПРЕДСТАВЛЕНИЙ О ТИПЕ ЭЛЕМЕНТАРНЫХ ДАННЫХ

Традиционное представление о типах данных существенно обогатилось с появлением в арсенале средств автоматизации программирования алгоритмического языка ПАСКАЛЬ, который позволил программисту не только относить данные к одному из типов данных — ЦЕЛЫЕ, ВЕЩЕСТВЕННЫЕ, БУЛЕВЫ, ЛИТЕРЫ и др., но и определять на их базе собственные типы данных. Одним из дополнительных определяемых программистом в программах на языке ПАСКАЛЬ типов данных является так называемый *перечисляемый тип.*

Перечисляемый тип задается перечислением в *определенном порядке* всех возможных значений, которые могут принимать элементарные данные, отнесенные к этому типу. Упорядоченность элементов перечисляемого типа является его важным свойством, позволяющим применять к элементам типа операции отношения. При этом считается, что «бóльшим» из двух значений перечисляемого типа является то, которое в списке возмож-

ных значений его элементов стоит левее. Слово «большим» взято в кавычки, так как применение операций отношения к элементам перечисляемого типа основано не на привычном сравнении их числовых значений, а на использовании понятия «больше по определению». Например, если элементами перечисляемого типа являются цвета радуги, заданные перечислением (красный, оранжевый, желтый, зеленый, голубой, синий, фиолетовый), то справедливо логическое выражение:

красный < зеленый,

причем оно справедливо только потому, что при определении типа в перечне значений элементов указан сначала «красный», а затем «зеленый». Подобное применение операций отношения рассматривалось при обсуждении элементарного типа данных ЛИТЕРЫ.

Типы данных, которые можно представить как перечисляемый тип, называются *порядковыми*. Следует обратить внимание на то, что перечисляемый тип данных не является каким-то особым типом. К перечисляемым порядковым типам относятся и рассмотренные типы данных ЦЕЛЫЕ, ЛИТЕРЫ, БУЛЕВЫ, так как возможные значения элементарных данных этих типов можно перечислить в порядке возрастания.

Элементы типа данных ЦЕЛЫЕ могут принимать ограниченное число значений от наибольшего по модулю отрицательного числа до максимально возможного для каждой конкретной ЭВМ положительного. Перечисляемым является тип данных БУЛЕВЫ, элементы которого могут принимать лишь одно из двух возможных значений (FOLSE или TRUE), а также тип данных ЛИТЕРЫ, значениями элементов которого являются символы, принадлежащие определенному для ЭВМ упорядоченному литерному набору.

Чтобы продемонстрировать средства, предоставляемые языком ПАСКАЛЬ для описания дополнительных перечисляемых типов данных, покажем описание типа данных БУЛЕВЫ, в том случае, если он не являлся бы типом элементарных данных этого алгоритмического языка:

type BOOLEAN = (.F.,.T.);

Описание типа начинается словом type (от англ. ТИП), затем следует название типа данных (BOOLEAN), знак равенства и последующее перечисление в скобках

возможных значений элементов типа (две точки в записи значения позволяют отличить их от букв F, T, используемых как имена переменных).

Описание типа данных БУЛЕВЫ, подобное приведенному, встроено в компилятор алгоритмического языка наряду с описаниями типов данных ЦЕЛЫЕ, ЛИТЕРЫ и др. Поэтому рассмотренные элементарные типы данных в отличие от дополнительных, определяемых программистом, называются *встроенными*.

Аналогично тому, как это было показано на примере описания типа данных БУЛЕВЫ, в программе на языке ПАСКАЛЬ можно описать любой дополнительный тип, введение которого по тем или иным соображениям представляется программисту оправданным.

Допустим, что разрабатывается программа обработки на ЭВМ кадровых анкет. В такой программе можно определить ряд дополнительных типов, подобных следующим:

```
type POL=(МУЖ, ЖЕН);*
   PART=(БЕСП, ВЛКСМ, ПАРТ);
    OBR=(ДР, СРЕД, СРЕД-ТЕХН, СРЕД-СПЕЦ, ВЫСШ);
```

Слова POL, PART, OBR при наличии в программе приведенного определения типов являются такими же ключевыми для определения типов переменных, как слова BOOLEAN, REAL и др., и их можно использовать для описания типов переменных в программе:

```
var BPOL, APOL : POL;
    BPART, APART : PART;
    BOBR, AOBR : OBR;
```

Описанным таким образом переменным APOL, APART, ... BOBR, ... можно присвоить в программе, например, следующие значения:

```
APOL  =ЖЕН;
APART=ВЛКСМ;
AOBR  =ВЫСШ;
```

При решении задачи выбора специалиста для выполнения какой-либо работы можно записать в программе условие:

```
if AOBR>BOBR then... else...,
```

*Следует иметь в виду, что разные версии языка ПАСКАЛЬ накладывают те или иные ограничения на правила записи указываемых в скобках значений элементов перечисляемого типа, которые здесь не рассматриваются.

реализующее примерно такой алгоритм: «если специалист А имеет образование более высокое, чем специалист В, то оказать предпочтение специалисту В при направлении на сельхозработы...»

Какие же преимущества дает использование дополнительных перечисляемых типов? Запись программы становится более компактной, имеет простую и ясную логическую структуру, соответствующую характеру решаемой задачи. Например, если не ввести для обработки кадровых анкет дополнительных типов, то придется, вероятнее всего, использовать специальные *кодификаторы*, ставящие в соответствие значениям пунктов анкеты цифровые коды: ЖЕН — 1, МУЖ — 2, БЕСП — 1, ВЛКСМ — 2, ПАРТ — 3 и т. д.

В этом случае программа может стать менее ясной и более громоздкой хотя бы за счет необходимых преобразований кодов в значения и обратно.

Можно избежать кодирования, отнеся в программе соответствующие переменные к типу данных СТРОКИ. Но представьте себе, что при вводе значения переменной AOBR оператор ошибся и вместо ВЫСШ ввел значение ВЫШС. Если в программе не предусмотрены специальные средства контроля исходных данных, эта ошибка не обнаружится и приведет в дальнейшем к далеко не всегда очевидным искажениям результатов обработки. Если же переменная AOBR отнесена к перечисляемому типу OBR, то введенное оператором значение «ВЫШС» сразу же идентифицируется как ошибочное, так как оно не входит в перечень возможных значений переменных типа OBR.

Эффективным способом выявления ошибок в данных является определение в программах еще одного дополнительного типа данных — *интервального*.

Базовыми для определения интервальных типов служат порядковые встроенные типы и дополнительно определенные в программе перечисляемые типы. Интервальный тип описывается так же, как и перечисляемый тип, но допустимые значения элементов типа задаются не перечислением, а указанием интервала значений, которые принимают элементы определяемого типа. Интервал задается указанием минимального и максимального значений, разделенных двумя точками. Например, можно определить интервальные типы, элементы которых принимают целые значения в заданном интервале:

```
type GOD = 1991..1995;
     TEMP = -50..+50;
```

В отличие от элементов типа данных ЦЕЛЫЕ элементы типа GOD не принимают значений, меньших 1991

и больших 1995, а элементы типа TEMP — соответственно (−50) и (+50). Если вводится значение, выходящее за пределы определенного интервального типа, то оно идентифицируется как ошибочное. На этом положительном свойстве интервальных типов и основывается их практическое использование.

Дополнительные типы имеет смысл вводить, если они используются для описания многих переменных. Если же в программе для каждой переменной необходимо определить свой интервальный тип, что вполне возможно, то в этом случае в языке ПАСКАЛЬ можно совместить описания переменной и типа и таким образом сделать программу короче. Так, переменную GOD−PUSKA можно описать двумя способами:

```
type GOD = 1991..1995;
var  GOD−PUSKA : GOD;
```

или так:

```
var GOD−PUSKA : 1991..1995;
```

Интервальные типы можно задать и с помощью констант:

```
const GODMAX = 1995;
      GODMIN = 1991;
type GOD = GODMIN..GODMAX;
```

Это позволяет при необходимости легко переопределить интервальные типы. Для этого достаточно изменить описания констант в начале программы.

Как уже указывалось, встроенные типы INTERGER, LITERAL, BOOLEAN, а также дополнительные перечисляемые и интервальные типы являются порядковыми. Все возможные значения элементов порядковых типов можно пронумеровать в соответствии с порядком их следования от минимального до максимального значения. Так, для интервального типа

```
type DLINA = 0..250;
```

порядковый номер элемента совпадает с его значением, т. е. он изменяется от 0 до 250. Для других порядковых типов связь порядкового номера элемента с его значением не столь очевидна.

На практике для того, чтобы по значению элемента определить порядковый номер и обратно — по номеру определить значение, используются специальные функ-

ции. Примером являются рассмотренные выше функции CHR$ (N) и ASC (L), используемые при операциях с данными типа ЛИТЕРЫ. Эти функции позволяют по известному порядковому номеру литеры N, являющемуся ее кодом, получать литеру и, наоборот, по известной литере L получать ее код (порядковый номер в литерном наборе).

В общем случае для определения номера элемента порядкового типа данных используется функция ORD (V), которая по известному значению V элемента выдает его порядковый номер. Так, для определенных выше порядковых типов POL, PART, OBR использование функции ORD можно продемонстрировать с помощью следующих примеров:

```
ORD (ЖЕН)=1
ORD (ПАРТ)=2
ORD (ВЫСШ)=4
```

В практических приложениях для заданного элемента порядкового типа может возникать потребность в определении *предшественника* и *преемника*. Для элементов базового типа данных ЦЕЛЫЕ это легко достигается соответственно вычитанием или прибавлением единицы к исходному значению элемента. В общем случае для нахождения значений предшественника и преемника элемента порядкового типа в языке ПАСКАЛЬ используются специальные функции, соответственно PRED (V) и SUCC (V), где V — значение элемента порядкового типа, для которого определяются предшественник и преемник. Используя эти функции для элементов порядкового перечисляемого типа OBR, можно записать:

```
SUCC (ДР)          =СРЕД
PRED (ВЫСШ)        =СРЕД — СПЕЦ
ORD (SUCC (СРЕД))=2
```

4.4. СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ

Обрабатываемые при решении задач на ЭВМ данные обычно взаимосвязаны и образуют структуры различной сложности, которые в зависимости от типа входящих элементов, характера их взаимосвязи и других особенностей можно отнести к тому или иному структурированному типу. В данном параграфе описываются наиболее используемые структуры данных — массив и запись.

Массив данных представляет собой совокупность значений одинакового типа. Примерами массивов могут служить:

— перечень фамилий и инициалов учащихся:

Андреев В. Г. Большаков С. А. Григорьев В. П. Талызова А. Е.

— соответствующий перечню фамилий список годов рождений учащихся:

1975 1976 1975 1973 1977

— перечень значений аргументов некоторой функции

−4.6 −2.1 −0.4 0.6 0.9 1.6 4.3

Каждое из значений, составляющих массив, называется его *компонентой*. Компонентами первого массива из приведенных примеров являются значения типа данных СТРОКИ, второго — ЦЕЛЫЕ, третьего — ВЕЩЕСТВЕННЫЕ. Ограничений на тип компонент массива обычно не накладывается. Это может быть любой из встроенных или дополнительно определенных программистом типов элементарных данных или один из структурированных типов, описываемых в данной главе.

Массив данных в программе рассматривается как переменная структурированного типа. Массиву присваивается имя, посредством которого можно ссылаться как на массив данных в целом, так и на любую из его компонент. Имена или идентификаторы массивов обычно записываются по правилам, устанавливаемым алгоритмическим языком для записи имен обычных переменных (см. § 4.1). Приведенным выше в качестве примеров массивам данных можно было бы присвоить имена FAMI, GR, X. Чтобы сослаться в программе на компоненту массива, надо кроме имени массива указать индекс компоненты, который в простейшем случае равен порядковому номеру этой компоненты в массиве. Значение индекса обычно записывается в круглых или квадратных скобках и следует за именем. Таким образом, компонентам массивов FAMI, GR, X соответствуют следующие обозначения:

```
FAMI (1), FAMI (2), FAMI (3), FAMI (4), FAMI (5);
  GR (1),   GR (2),   GR (3),   GR (4),   GR (5);
   X (1),    X (2),    X (3),      ...    X (7).
```

Значение «Григорьев В. П.» представляется компонентой FAMI (3) массива FAMI, год рождения Дмитри-

евой Т. М.— компонентой GR (4) массива GR, а значение 0.9 массива X — компонентой массива X (5).

В ряде алгоритмических языков используется правило нумерации компонент массива, при котором первой его компоненте ставится в соответствие индекс (0) — «ноль», а не (1) — «один».

Переменные, представляющие в программах компоненты массивов, называются *переменными с индексами* в отличие от простых переменных, представляющих в программах элементарные данные. Индекс в обозначении компонент массивов может быть константой, переменной или выражением, обычно целого типа. В некоторых алгоритмических языках допускается вещественный тип индексов массивов, значения которых для определения номера соответствующей компоненты массива округляются до целых.

Стройное расширение представлений о возможном типе индексов компонент массивов дает алгоритмический язык ПАСКАЛЬ, в котором индекс определяется как интервальный тип на базе любого из порядковых типов. При таком определении типа индекс может приниматься например, отрицательные целые или литерные значения. В основе этого, на первый взгляд, произвола лежит строгий порядок: номер компоненты массива равен порядковому номеру значения его индекса в последовательности возможных значений элементов соответствующего интервального типа.

Для размещения массива в памяти ЭВМ отводится поле памяти, размер которого определяется типом, длиной и количеством компонент массива. Тип и длина компонент массива указываются средствами явного или неявного задания типа, подобно тому, как это делается при описании простых переменных. Количество компонент массива задается в описании массива посредством указания интервала возможных значений индекса его компонент. При этом интервальный тип индекса задается явно, как это делается в программах на языке ПАСКАЛЬ:

```
[1..30], [-10..+10]
```

или неявно, когда в скобках указывается лишь максимальное значение индекса:

```
(30), (20), (100),
```

а минимальное значение предполагается равным единице или нулю.

Приведем несколько характерных примеров описания массивов средствами различных алгоритмических языков.

Пример 4.8. Описание массивов средствами алгоритмического языка ПАСКАЛЬ:

```
var SPISOK : array [1..20] of INTERGER;
    X,Y : array [1..50] of REAL;
```

В этом примере описаны массивы SPISOK, X и Y. Массив SPISOK содержит 20 компонент типа ЦЕЛЫЕ, а массивы X и Y — по 50 компонент типа ВЕЩЕСТВЕННЫЕ. То, что переменные SPISOK X и Y являются массивами, указывается ключевым словом array (англ. массив). Тип компонент указан ключевыми словами INTERGER (ЦЕЛЫЕ) и REAL (ВЕЩЕСТВЕННЫЕ).

Пример 4.9. Описание массивов средствами алгоритмического языка БЕЙСИК:

```
DIM A!(29), B%(39), NOM$(24)
```

Ключевое слово DIM (от англ. DIMENSION — размерность) указывает на то, что данная строка является описанием массивов. В строке описываются:

— массив A!, содержащий 30 элементов (30, так как минимальное значение индекса равно нулю) типа ВЕЩЕСТВЕННЫЕ, на что указывает последний символ имени (!) (см. табл. 4.1);

— массив B%, включающий в себя 40 элементов типа ЦЕЛЫЕ (тип элементов массива указывает последний символ имени — %);

— массив NOM$ из 25 элементов типа СТРОКИ.

Пример 4.10. Описание массивов средствами алгоритмического языка ФОРТРАН:

```
DIMENSION A(30), N(40)
    REAL*8 B(40), D
```

Первая строка описывает размерность (DIMENSION) массивов A, содержащего 30 компонент, и N, содержащего 40 компонент. Тип компонент массивов A и N задается неявно в соответствии с соглашением, по которому переменные с именами, начинающимися с букв I, J, K, L, M, N, относятся к типу ЦЕЛЫЕ , остальные — к типу ВЕЩЕСТВЕННЫЕ.

Вторая строка описывает ВЕЩЕСТВЕННЫЕ удвоенной точности: массив B(40) и переменную D. Удвоенная точность задается указателем (*8) нестандартной длины, равной 8 байт. Размерность (40) массива B указана в операторе описания REAL, и специально описывать ее в операторе DIMENSION не требуется.

Рассмотренные выше примеры являются примерами одномерных массивов, для указания компонент которых используется один индекс. Структуре одномерного массива в документах, используемых человеком в повседневной деятельности, соответствуют различные списки, перечни, подобные приведенным спискам фамилий и годов рождения.

Наряду со списками при решении практических задач часто приходится иметь дело с различными таблицами данных, математическим эквивалентом которых служат *матрицы*. Как известно из математики, *матрицей* называ-

ется система чисел, образующих прямоугольную таблицу из m строк и n столбцов:

$$\|A\| = \begin{vmatrix} a_{11} & a_{12} \dots a_{1n} \\ a_{21} & a_{22} \dots a_{2n} \\ \vdots & \vdots \quad \vdots \\ a_{m-1} & a_{m-2} \; a_{mn} \end{vmatrix}.$$

Матрица является примером двухмерного массива данных, для ссылки на каждую компоненту которого необходимо указать два индекса: номер строки i и номер столбца j.

В программах при описании размерности двухмерного массива в квадратных или круглых скобках указываются пределы возможного изменения каждого из двух индексов. Приведем примеры описания двухмерного массива A, соответствующего прямоугольной матрице, имеющей по 20 значений в каждой из 10 строк:

а) на языке ПАСКАЛЬ:

```
Aarray [1..10, 1..20] of REAL;
```

б) на языке ФОРТРАН:

```
DIMENSION A(10, 20)
```

в) на языке БЕЙСИК:

```
DIM A! (9, 19)
```

Компоненты двухмерного массива обозначаются именем массива, дополненным двумя индексами, указанными в скобках и разделенными запятыми. Например:

```
A (9, 16), A (I, J), A (I*2, J−3).
```

При выполнении инженерных и математических расчетов часто используются переменные более чем с двумя индексами. При решении задач на ЭВМ такие переменные представляются, как компоненты соответственно трех-, четырехмерных массивов и т. д.

Важно понимать, что описание массива в виде многомерной структуры делается лишь из соображений удобства программирования как результат стремления наиболее точно воспроизвести в программе объективно существующие связи между элементами данных решаемой задачи. Что же касается образа массива в памяти ЭВМ, то как одномерные, так и многомерные массивы хранятся в виде линейной последовательности своих компонент, и принципиальной разницы между одномерными и многомерными массивами в памяти ЭВМ нет. Однако порядок,

в котором запоминаются элементы многомерных массивов, важно себе представлять. Необходимость этого особенно очевидна для правильной реализации операций ввода и вывода. Так, если транслятор с алгоритмического языка рассматривает двухмерный массив как хранящийся в памяти «по столбцам», а данные для ввода в ЭВМ значений компонент массива подготовлены «по строкам», то, несмотря на возможное внешне благополучное решение задачи, полученные результаты скорее всего окажутся неправильными.

В большинстве алгоритмических языков реализуется общее правило, устанавливающее порядок хранения в памяти элементов массивов: элементы многомерных массивов хранятся в памяти в последовательности, соответствующей более частому изменению младших индексов. Младшими индексами считаются те, которые расположены левее, т. е. ближе к началу списка индексов переменной, указанному в скобках за ее именем. Интерпретируя это правило применительно к двухмерному массиву B (3, 4), получим следующий порядок хранения его компонент в памяти ЭВМ:

B (1,1), B (2,1), B (3, 1), B (1, 2), B (2, 2), B (3, 2),
B (1,3), B (2, 3), B (3, 3), B (1, 4), B (2, 4), B (3, 4).

Если записать соответствующую массиву B (3,4) матрицу ‖B‖, то можно убедиться в том, что элементы матрицы хранятся в памяти ЭВМ «по столбцам», т. е. сначала все элементы первого столбца, затем все элементы второго столбца и т. д.

Чтобы полнее реализовать преимущество представления данных в виде многомерных массивов в некоторых алгоритмических языках, используется такое понятие, как сечение массива, позволяющее реализовать обращение по имени не только к массиву в целом или любой из его компонент, но и к совокупностям компонент. Применительно к двухмерным массивам сечением массива могут являться все элементы какого-либо столбца или строки соответствующей прямоугольной матрицы. Для обозначения сечений массивов вводятся специальные обозначения. Например, в алгоритмическом языке ПЛ/1 обозначение B (*,1) является сечением массива B (N, M) и представляет все элементы столбца 1 соответствующей матрицы, а обозначение B (3,*) — сечением, представляющим все элементы строки 3.

К компонентам массивов применяются любые операции обработки, разрешенные в используемом алгорит-

мическом языке для элементарных данных соответствующего типа.

Существенно позволяет повысить эффективность программирования использование разрешенных в ряде алгоритмических языков **массивных выражений**, в которых операции арифметические, логические, отношения применяются не только к элементарным типам данных, но и к таким структурированным типам, как массивы. Если, например, A, B и C являются структурно-подобными массивами, т. е. массивами, имеющими одинаковые размерности, тип индексов и компонент, то можно записать выражения

A+B;
B*B−C.

Результатом вычисления таких выражений является структурно-подобный массив, компоненты которого равны сумме, произведению, разности соответствующих компонент массивов-операндов.

Очевидно, что применение массивных выражений существенно повышает эффективность программирования, особенно при подготовке к решению на ЭВМ таких задач, в которых для представления исходных данных и результатов часто используются матрицы: решение систем уравнений, анализ электрических цепей и др.

Над массивами выполняют такие широко используемые операции, как:

— поиск максимального и минимального значений;

— упорядочение (сортировка) элементов массивов в порядке убывания или возрастания;

— подсчет количества элементов в массиве, удовлетворяющих заданному условию.

Для реализации подобных операций обычно используются предназначенные для работы с массивами специальные стандартные программы и встроенные функции.

Запись. Основное назначение структурированного типа ЗАПИСЬ — отображение в памяти ЭВМ документов, традиционно используемых в повседневной практике при решении широкого класса экономических, информационно-поисковых и других «нематематических» задач. Наиболее распространенными типами таких документов являются картотеки и таблицы, примеры которых приведены на рис. 4.5, *а*; 4.6, *а*; 4.7, *а*.

Каждая строка таблицы или отдельная карточка картотеки обычно содержит совокупность разнотипных сведений в каком-либо объекте. В памяти ЭВМ эти сведения хранятся в виде последовательности полей элементарных

а)

Фамилия	Иванов
Имя	Сергей
Отчество	Иванович
Дата рождения:	
число	11
месяц	02
год	1978
Пол	м
Успеваемость:	
физика	5
математика	4
русский язык	4
физвоспитание	5
музыка	4
рисование	4

б)

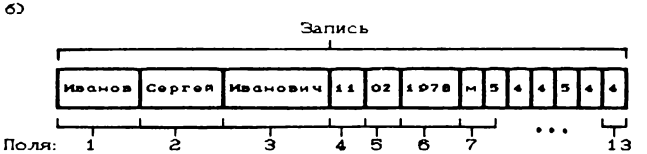


Рис. 4.5

данных различных типов и описываются в программе как особая переменная структурированного типа ЗАПИСЬ. В качестве примера на рис. 4.5, *а* приведена условная карточка учащегося из картотеки, которая в той или иной форме ведется в большинстве учебных заведений. Условное изображение структурированной переменной типа ЗАПИСЬ, соответствующее этой карточке, приведено на рис. 4.5, *б*.

Тип элементарных данных отдельных полей записи определяется характером хранимой в них информации, а также тем, какие действия предполагается выполнять над этой информацией. Например, сведения об оценках учащегося могут храниться в памяти как элементарные данные типа ЛИТЕРА, но если предполагается их обработка с целью вычисления таких показателей, как «средний балл», «% успеваемости» и др., то оценки предпочтительнее описать как элементарные данные типа ЦЕЛЫЕ или ВЕЩЕСТВЕННЫЕ.

При описании записи в программах указываются тип и длина каждого из ее полей. Логическая последовательность полей в записи задается порядком описания. Каждое поле записи и запись в целом получают имена.

Приведем пример описания средствами языка ПАСКАЛЬ записи, содержащей внесенные в карточку на рис. 4.5, *а* сведения об учащемся:

```
var
   KARTUCH : record
      FAMI, IMIA, OTCH : STRING;
      CHISLO : 1..31;
      MES : 1..12;
      GOD : INTERGER;
      POL : (M, Ж);
      FIZ, MAT, R, FV,
      MUZ, RIS : 2..5
   end;
```

В приведенном примере запись KARTUCH описана как переменная (var) структурированного типа ЗАПИСЬ, на что указывает ключевое слово record (англ. ЗАПИСЬ). В описании записи перечисляются поля, для каждого из которых указывается идентификатор (FAMI, OTCH, MES, GOD, ...) и тип, который может быть как встроенным (INTERGER, STRING, ...), так дополнительно определяемым, например интервальным: 1...31, 1...12 или перечисляемым: (М, Ж). Перечень описываемых полей записи заканчивается служебным словом end, указывающим на конец описания записи.

В общем случае для ссылки на отдельные поля записи в программе используются идентификаторы, в которые входят разделенные точкой имя записи и имя соответствующего поля. Например, к полю GOD можно обратиться по идентификатору KARTUCH.GOD, а к полю POL — по идентификатору KARTUCH.POL. Необходимость использования такой идентификации обусловлена тем, что в одной программе могут определяться и другие записи, содержащие поля, имеющие те же имена: GOD, MES и т. п.

Обычно объектом обработки в ЭВМ является не единственная карточка или анкета, а некоторый их набор, которому соответствует не единственная запись, а совокупность или массив записей. Массив записей описывается так же, как обычный массив, но тип его компоненты описывается как структурированный тип ЗАПИСЬ.

В программе, записанной на языке ПАСКАЛЬ, описание массива записей KLAS — 7А, содержащего, например, сведения о 25 учениках 7 «А» класса, имеет такой вид:

```
type
   KARTUCH =record
         FAMI, IMIA, OTCH :  STRING;
         CHISLO :  1..31;
```

```
            MES :  1..12;
            GOD :  INTERGER;
            POL :  (М, Ж);
            FIZ, MAT, R, FV, MUZ,
            RIS :  2..5
        end;
KLAS __ 7A : array [1..25] of KARTUCH;
```

Последняя строка является описанием массива, содержащего 25 компонент, имеющих тип KARTUCH, предварительно описанный как структурированный тип ЗАПИСЬ. Компонента массива KLAS __ 7A [1] содержит все сведения о первом по списку ученике класса, компонента KLAS __ 7A [2] — о втором и т. д. Чтобы обратиться к полю любой компоненты массива записей, следует так же, как и в случае отдельной записи, указать его идентификатор, состоящий из разделенных точкой имени записи и имени поля. Именем записи в этом случае является имя соответствующей компоненты массива, например KLAS __ 7A [2], KLAS __ 7A [14] и т. д. Имена полей определены описанием структурированного типа record и одинаковы для всех записей массива. Так, идентификатор KLAS __ 7A [14].FIZ определяет поле, содержащее оценку по физике ученика, четырнадцатого по списку; фамилия этого ученика хранится в поле KLAS __ 7A [14].FAMI.

Массив записей — наиболее естественная форма представления в памяти ЭВМ различных таблиц, которые являются широко используемым на практике типом традиционных документов. Различные картотеки, подобные рассмотренной, по сути, являются таблицами, которые, исходя из соображений удобства работы с данными, организованы специальным образом, когда каждая строка таблицы записывается на отдельной карте. Таблицы оказались столь универсальной формой представления данных, что в настоящее время разработан ряд программных средств, специально предназначенный для обработки на ЭВМ таблиц.

Учитывая это, рассмотрим несколько подробнее сами таблицы и обозначим более явно связи между структурными элементами таблиц и элементами, отображающих их в памяти ЭВМ структурированных типов данных. На рис. 4.6, *а* приведен пример простейшей таблицы. Вертикальные линии делят таблицу на *графы*, а горизонтальные — на *строки*. В верхней части таблицы расположен элемент, называемый *шапкой*, в которой содержатся названия всех граф. Под шапкой находится информационная часть таблицы, представляющая собой последовате-

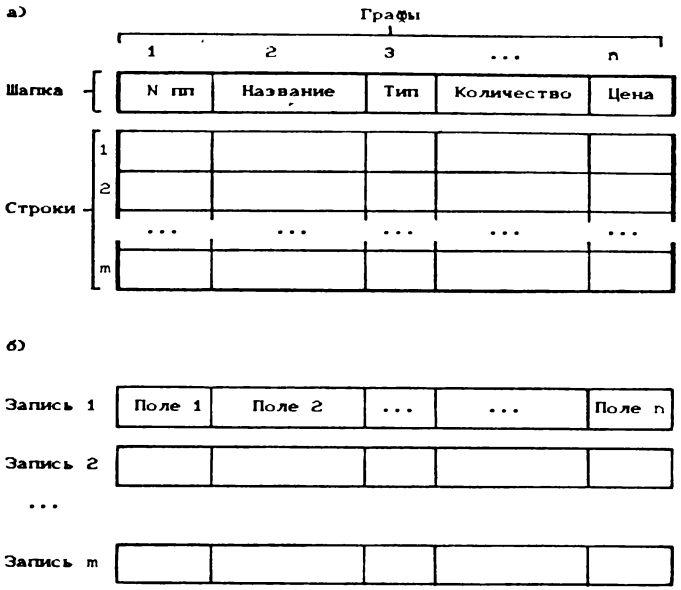


Рис. 4.6

льность строк. При представлении данных таблицы в памяти ЭВМ в виде массива записей количество строк таблицы соответствует количеству записей в массиве (рис. 4.6, *б*). Количество граф таблицы определяет количество полей каждой записи. Шапка таблицы является ее своеобразным описанием, позволяющим правильно интерпретировать данные таблицы, и используется в качестве исходного документа для составления описания соответствующего структурированного типа ЗАПИСЬ.

В общем случае шапка таблицы имеет так называемую иерархическую структуру. Пример структурно более сложной таблицы приведен на рис. 4.7, *а*. Особенностью этой таблицы является то, что такие ее графы, как «Начислено», «Вычеты», «В том числе», разбиваются на подграфы. Запись, хранящую в памяти ЭВМ данные строки, можно представить состоящей из пяти полей (рис. 4.7, *б*) с именами: NP, FAMI, NACHISLENO, VICHET, SUMA. Среди перечисленных поля NR, FAMI и SUMA являются *простыми* (на рисунке они заштрихованы), так как хранят элементарные данные. В отличие от простых поля NACHISLENO и VICHET называются *составными*, так как включают в себя по два поля. Так как поля записи образуют иерархическую совокупность дан-

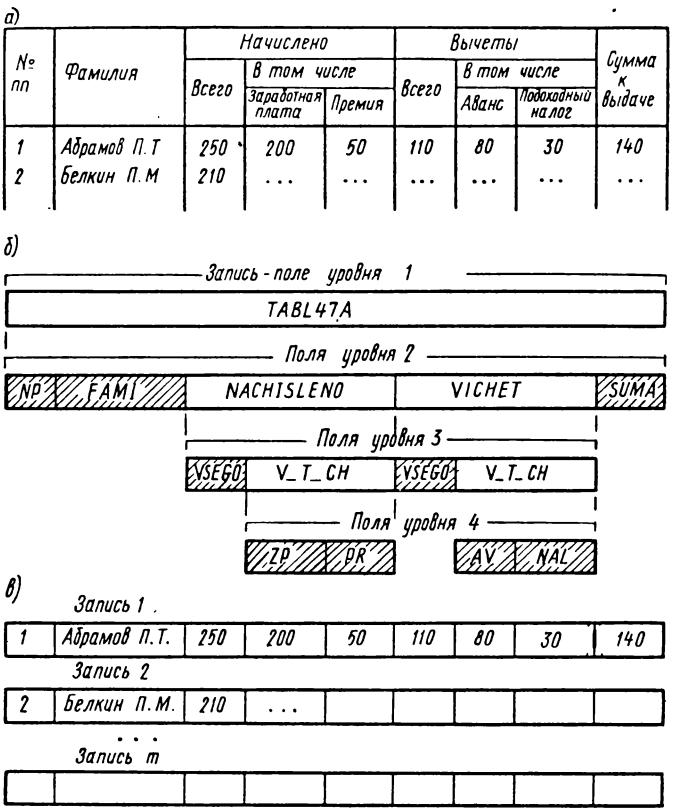


Рис. 4.7

ных, при описании записи удобно использовать понятие *номер уровня поля*. При этом запись рассматривается как составное поле уровня 1. Входящие в него простые и составные поля являются полями уровня 2. Составные поля уровня 2 включают в себя поля уровня 3 и т. д.

Составное поле, входящее в запись, может рассматриваться как «запись в записи». Именно такое представление о составном поле и используется при описании записей с иерархической структурой полей. В качестве примера приведем описание средствами языка ПАСКАЛЬ массива записей, отображающего в памяти ЭВМ данные таблицы на рис. 4.7, *а*, содержащей 50 строк:

```
var TABL47A :  array [1 : 50]  of
    record
      NP :  1..50;
```

```
FAMI :  STRING;
NACHISLENO : record
                VSEGO :  0..999;
                V–T–CH : record
                            ZP, PR :  0..999
                         end
             end;
VICHET :     record
                VSEGO :  0..999;
                V–T–CH : record
                            AV, NAL :  0..999
                         end
             end;
SUMA :  0..999
end;
```

Для обращения к любой компоненте массива достаточно указать имя массива и индекс компоненты. Например, компонента TABL47A [2] содержит данные строки 2 таблицы. Для ссылки на любое поле компоненты массива надо указать последовательно имена всех составных полей, в которые входит это поле. В качестве разделителя имен используется точка. Например, имя

TABL 47A [2].VICHET.VSEGO

идентифицирует поле, хранящее значение вычетов из зарплаты тов. Белкина П. Н., второго по списку в ведомости на зарплату. Сумма начисленной ему премии хранится в поле, на которое можно сослаться по имени

TABL 47A [2].NACHISLENO.V–T–CH.PR

Важно понимать, что вне зависимости от того, описывается запись как включающая только простые поля или простые и составные, форма представления записей в памяти ЭВМ остается идентичной (см. рис. 4.5, *б* и 4.7, *в*).

Из приведенных примеров видно, что описание записей в виде иерархических структур довольно громоздко, так же как громоздки и имена составляющих их полей. Поэтому использование таких структур должно оправдываться теми преимуществами, которые дает подобное представление данных.

Вопросы для самопроверки

4.1. Перечислите основные встроенные типы элементарных данных.

4.2. Сформулируйте характерное для алгоритмических языков общее правило записи имен переменных.

4.3. В чем состоят особенности выполнения операций целочисленного деления, вычисления остатка? Приведите примеры.

4.4. Определите значения булевых переменных B1, B2, B3 в следующих выражениях:

B1 : =17>3;

B2 : =6>8;
B3 : =X**1> =X.

4.5. Определите значение функции CHR (67), если известно, что ASC (C)=67. Что это за функции, для чего они используются?

4.6. Определите значения следующих выражений и функций:

LEN ('ДЛИНА СТРОКИ')+LEN ('');
'19 июля'+'1989';
'ниже'+'нуля';
'выше'+'нуля';
POS ('И', 'ДЛИНА');
COPY ('ДЛИНА', 4,2).

4.7. Объясните назначение операции конкатенации. Определите значение функции:

CONCAT ('НО', 'ВОЕ', 'ВРЕМЯ')

4.8. Объясните разницу между средствами явного и неявного описания типа данных. Приведите примеры.

4.9. Прокомментируйте примеры 4.5, 4.6, 4.7 описания данных (см. § 4.2).

4.10. Что такое спецификации формата, для чего они используются? Приведите примеры записи значений данных для следующих спецификаций формата F8.3, E12.5, I4.

4.11. Приведите примеры записи значений данных по следующим шаблонам: '≠≠≠≠.≠≠≠', '≠≠≠≠','≠≠≠.≠≠≠^^^^'.

4.12. Объясните разницу между встроенными типами данных и типами данных, определяемыми программистом.

4.13. Используя приведенные в § 4.2 примеры, составьте описание на языке ПАСКАЛЬ перечисляемых типов: ДНИ НЕДЕЛИ, МЕСЯЦЫ ГОДА. Определите значения логических выражений:

ЯНВАРЬ>ФЕВРАЛЬ
МАРТ<=АПРЕЛЬ

4.14. Что такое интервальные типы данных? Какие преимущества дает их применение в программах?

4.15. Что такое массив данных? Приведите примеры массивов данных.

4.16. Может ли являться совокупность данных массивом: —14.5 86 1914 'ПЕТРОВ'. Если нет, то почему? Если да, то при каком условии?

4.17. Что такое размерность массива? Чем отличаются одно- и двухмерный массивы?

4.18. Поясните приведенные в 4.4 примеры описания массивов (примеры 4.8, 4.9, 4.10).

4.19. В чем состоит главное отличие структурированных типов ЗАПИСЬ и МАССИВ?

4.20. Приведите пример таблицы с иерархической шапкой. Используя приведенные в 4.4 примеры, составьте для этой таблицы описание ее как массива записей.

Глава 5
ОРГАНИЗАЦИЯ ДАННЫХ

5.1. ФАЙЛЫ ДАННЫХ

Выполнение на ЭВМ большинства программ сопровождается вводом-выводом данных. Используемые для этого устройства отличаются назначением, конструкцией, типом носителя информации, другими особенностями, обусловливающими специфику размещения данных и доступа к ним. Совокупность правил, определяющих особенности размещения данных на внешних устройствах, методы доступа к ним, средства защиты данных от несанкционированного доступа и т. д., составляет *систему организации данных*.

Средства описания данных определяются выбранным для написания программы алгоритмическим языком, а организация данных на внешних устройствах ЭВМ — используемой на данной ЭВМ операционной системой. Совокупность средств операционной системы, обеспечивающих доступ к данным, называется *системой управления данными* или *файловой системой*.

Файлом называется совокупность данных, размещенная на внешнем устройстве вычислительной машины. Файл может хранить текст программы, массив, последовательность записей данных и т. п. С точки зрения программы пользователя и работающих с файлами программ операционной системы данные файла имеют определенную, иногда достаточно сложную, внутреннюю логическую структуру. Для внешнего же устройства ЭВМ файл является последовательностью символов, имеющей определенную длину и имя. Имя необходимо для обращения к файлу, хранящемуся вместе с другими файлами на магнитных дисках и лентах.

Так же как и имя переменной, *имя файла* составляется по определенным правилам и обычно включает в себя 8 символов и более, каждый из которых является буквой,

цифрой или одним из разрешенных специальных знаков. Имя файла может иметь расширение, которое отделяется точкой и содержит не более трех символов.

Ниже приведены примеры правильной записи имен файлов:

BASIC.EXE
SUMMA.BAS
GLAVA1.TXT
RIS1-1.UNV
GRAPHICS.COM

Имя файла обычно составляется так, чтобы пользователь мог легко вспомнить, что в нем хранится. Так, судя по приведенным в качестве примеров именам, можно предположить, что в файле GLAVA1.TXT хранится текст гл. 1 какой-либо книги, а в файле RIS1-1.UNY — один из рисунков этой главы.

Расширение имени обычно указывает на тип данных в файле. Например, в операционной системе MS DOS расширения EXE и COM присваиваются файлам, хранящим программы, готовые к выполнению. Файлы с расширением имени TXT хранят текстовые данные, а с расширением имени BAS — программы, написанные на алгоритмическом языке БЕЙСИК.

Для обращения к группам файлов применяются *групповые имена файлов*, образуемые с использованием символов * и ?. Символ *, встречающийся в имени файла, трактуется как «любая последовательность символов», символ ? — как «любой один символ». Ниже приведены примеры групповых имен файлов и пояснения к ним:

*.EXE — все файлы с расширением имени EXE (файлы типа EXE);
A*.COM — все файлы типа COM, имена которых начинаются с A;
B ??.BAS — все файлы типа BAS, имена которых содержат три символа, из которых первый — буква B;
. — все файлы;
PRG1.* — файлы любых типов, имеющие имя PRG1.

Основным типом устройств внешней памяти ЭВМ являются дисковые запоминающие устройства. Организация данных на дисках тем сложнее, чем больше суммарная емкость внешней памяти, используемой в составе оборудования конкретной ЭВМ. Наиболее просто организованы данные на дисковых накопителях персональных ЭВМ, для которых в качестве интернациональных утвер-

дились стандарты организации данных, разработанные фирмой IBM.*

Для долговременного хранения программ и данных ПЭВМ используются *накопители на гибких магнитных дисках* (НГМД) и так называемые *жесткие*, или *винчестерские*, диски. Чтобы пользователь мог сослаться на конкретное устройство дисковой памяти, каждый из дисковых накопителей имеет обозначение (идентификатор) — букву с последующим двоеточием (A:, B:, C: и т. д.). Идентификатор C: используется для указания жесткого диска, а идентификаторы A:, B:— для ссылок к накопителям на гибких дисках.

Для того чтобы обратиться к файлу, хранящемуся на одном из дисковых запоминающих устройств, к имени файла добавляется буквенный идентификатор накопителя, например:

```
A: FILE1. TXT
C: NOMBRES. DAT
```

Имя файла с приставкой — указателем устройства — называется *спецификацией*. Спецификация файла кроме идентификатора диска может включать также дополнительные указатели о местонахождении файла на диске с древовидной структурой каталогов. Особенности и правила составления полной спецификации файлов рассматриваются далее.

5.2. ОРГАНИЗАЦИЯ ДАННЫХ НА МАГНИТНЫХ ДИСКАХ

Накопители на магнитных дисках (НМД), используемые в современных ЭВМ, несмотря на существующие различия в конструкции, емкости, стоимости, быстродействии, имеют общее свойство: данные запоминаются на магнитной поверхности диска на *дорожках*, представляющих собой концентрические окружности (рис. 5.1). Количество данных, которое может храниться на поверхности диска, зависит от геометрических размеров диска и *плотности записи*. При заданных размерах диска плотность записи определяет *количество дорожек на поверхности* диска и *емкость каждой дорожки*.

Следующим параметром, связанным с емкостью дис-

*Поскольку эти стандарты широко используются и в отечественных персональных компьютерах, далее общие вопросы организации данных на дисках иллюстрируются примерами для IBM-совместимых ПЭВМ.

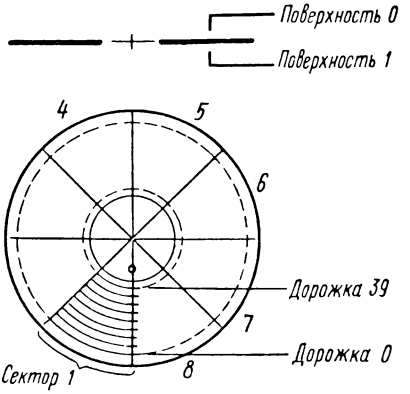


Рис. 5.1

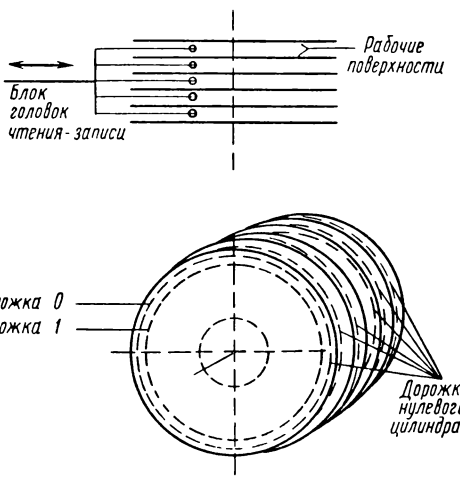


Рис. 5.2

ка, является *количество его рабочих поверхностей.* В гибких дисках для хранения данных используются одна или обе поверхности. Для увеличения количества рабочих поверхностей жесткие диски объединяются в *пакеты* (рис. 5.2).

Количество поверхностей диска и *максимальное число дорожек* на каждой из них являются параметрами физической структуры диска и фиксированы для каждого конкретного типа накопителя.

Различают запоминающие устройства со *сменными* и с *несменными* дисками. Механическая часть устройства несменных дисков является более простой, поэтому при

одинаковых геометрических размерах и уровне технологии изготовления они имеют более высокую емкость каждой поверхности.

Жесткие диски ПК обычно являются несменными, а гибкие магнитные диски — сменными.

Для подготовки диска к работе средствами используемой операционной системы выполняется операция, *форматирования диска*, в процессе которой каждая дорожка делится на сегменты, называемые *секторами* (см. рис. 5.1). Емкость сектора является параметром форматирования и может составлять 128, 256, 512, 1024 байт. Наиболее употребительным размером является емкость сектора в 512 байт.

Емкость диска может быть вычислена по формуле

$$v = pdk_c v_c,$$

где p — количество поверхностей диска; d — количество дорожек на одной поверхности; k_c — количество секторов на одной дорожке; v_c — емкость сектора.

В табл. 5.1 приведены семь основных форматов, используемых при форматизации гибких дисков IBM-совместимых микроЭВМ, работающих под управлением операционной системы MS DOS.

Таблица 5.1

Обозначение форматов	Количество поверхностей	Количество секторов на дорожке	Количество дорожек	Емкость диска, Кбайт
S-8	1	8	40	160
D-8	2	8	40	320
S-9	1	9	40	180
D-9	2	9	40	360
QD-9	2	9	80	720
QD-15	2	15	80	1200
QD-18	2	18	80	1440

В принятых в табл. 1.1 обозначениях буква S указывает форматы, использующие одну поверхность диска, буква D — форматы, использующие две поверхности, буква Q — форматы для дисков с повышенной плотностью записи, обеспечивающей размещение на поверхности 80 дорожек. Цифры 8, 9, 15, 18 в обозначениях форматов указывают количество секторов, на которое делится при форматировании каждая дорожка.

Гибкие диски часто называют дискетами. Наиболее распространенными типоразмерами дискет, используемых в ПК, являются дискеты диаметром 5,25 и 3,5 дюйм, т. е. $5^{1/4"}$ и $3^{1/2"}$. Для дисков диаметром $3^{1/2"}$ преимущественно используются форматы QD-9 и QD-18.

Диски диаметром $5^{1/4"}$ выпускаются в двух исполнениях: с удвоенной и повышенной (учетверенной) плотностью записи. Внешне они идентичны и отличить их обычно можно лишь по маркировке. Импортные диски удвоенной плотности обычно имеют маркировку 2S-2D, что означает «две стороны» (Double Side) и «удвоенная плотность» (Double Density). Встречаются и другие варианты маркировки дисков этого типа: MD-2-D, MD2-2D, DS-DD, MD-2/48. В последнем обозначении на удвоенную плотность записи указывает число 48. На этикетке дискеты это значение указывается среди прочих параметров в виде: 48 Tpi (48 дорожек на дюйм). Диски повышенной плотности (High Density) имеют маркировку, в которой плотность указывается буквой H или параметром плотности — 96 (96 Tpi): 2S-HD, MD2HD, MD-2/96. На этикетках дисков обоих типов можно встретить также надпись «Soft Sectored», указывающую на то, что количество секторов на дорожке диска не фиксировано и задается при выполнении форматирования.

Диски диаметром $5^{1/4"}$ удвоенной плотности преимущественно используются в формате D-9 (360 Кбайт), повышенной плотности — в форматах QD-9, QD-15 (720 Кбайт, 1,2 Мбайт).

В отличие от физического формата гибких дисков, параметры которого определяются окончательно лишь в результате выполнения форматирования, физический формат жестких дисков задается в процессе производства. В табл. 5.2 приведены параметры физического формата, используемого в IBM-совместимых ПЭВМ твердых дисков емкостью 10 и 20 Мбайт.

Таблица 5.2

Емкость диска, Мбайт	Количество поверхностей	Количество дорожек на поверхности	Количество секторов на дорожке
10	4	306	17
20	4	615	17

Для твердых дисков, являющихся обычно пакетированными, вместо параметра «количество дорожек на поверхности» чаще используется характеристика «количество цилиндров». *Цилиндр* образуют дорожки всех поверхностей диска, находящиеся одна под другой, т. е. на одинаковом расстоянии от центра диска (см. рис. 5.2).

Адрес любого сектора на диске определяется тремя координатами — номерами дорожки (цилиндра), поверхности и сектора на дорожке. Можно пользоваться адресом сектора для записи или чтения файлов (а их может быть на диске десятки и сотни), однако это трудоемко. Программисту в этом случае пришлось бы составлять каталог всех файлов с указанием для каждого из них перечня занимаемых секторов. Операционная система освобождает пользователя от необходимости вести подобные каталоги и ведет их сама. Для этого в процессе *инициализации* диска, которая выполняется после форматирования, на нем создаются дополнительные служебные файлы, образующие файловую структуру диска.

Файловая структура диска при возможных отличиях в разных операционных системах предполагает наличие на диске *каталога файлов* и специальных файлов, облегчающих поиск данных.

Каталог является тоже файлом и состоит из записей, содержащих сведения о хранящихся на диске файлах и параметрах доступа к ним. Общая схема доступа к файлу такова: программист обращается к файлу по имени; операционная система, прежде чем выполнить затребованные программистом действия, обращается к каталогу, находит в нем запись с соответствующим именем файла, определяет его местонахождение, а затем выполняет необходимые действия.

Простейшая структура каталога, изображенная на рис. 5.3, *а*, обычно используется для гибких дисков небольшой емкости, все файлы которых принадлежат одному пользователю. Для доступа к данным жестких дисков, хранящих сотни и тысячи файлов, применяется древовидная структура каталогов (рис. 5.3, *б*). В вершине древовидной структуры находится корневой каталог, создаваемый в процессе инициализации диска. Ветви дерева образуют подкаталоги, которые создаются и удаляются по инициативе пользователей по специальным командам.

Положение на диске корневого каталога фиксировано и известно операционной системе. Подкаталоги хранятся в области данных диска как обычные файлы. Имена подкаталогам присваиваются при их создании, как и любым другим файлам, по тем же правилам. Данные о ме-

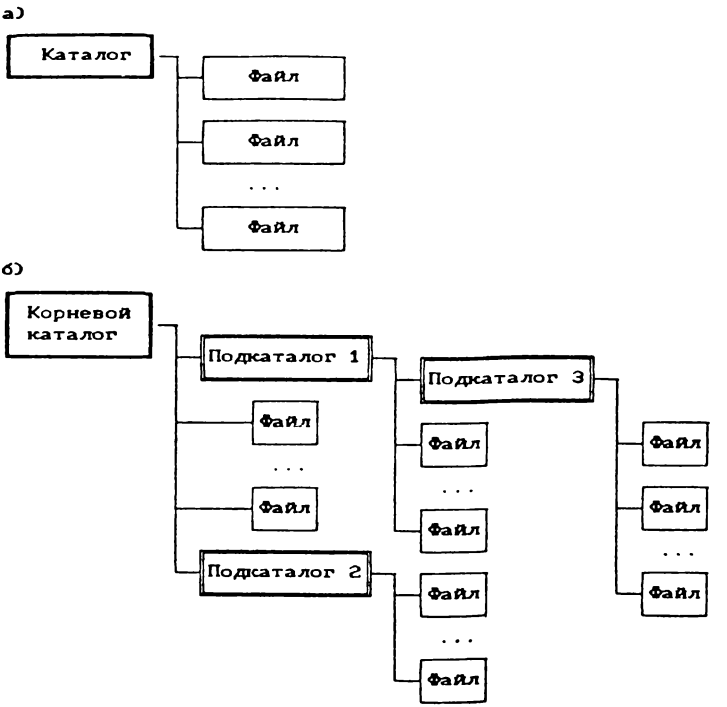


Рис. 5.3

стонахождении подкаталогов корневого каталога хранятся в корневом каталоге. Подкаталоги корневого каталога хранят сведения о подчиненных им подкаталогах и т. д.

Для того чтобы найти файл на диске с древовидной структурой каталогов, системе требуется просмотреть всю цепочку каталогов по пути от корневого каталога до подкаталога, хранящего сведения о файле. Таким образом, чтобы сослаться на файл, надо указать операционной системе не только имя файла, но и перечислить имена всех подкаталогов на пути к файлу. Указатель перечня имен каталогов на пути к файлу называется *маршрутом*. Перечисляемые в маршруте имена подкаталогов разделяются символом \. Поскольку корневой каталог имени не имеет, то запись маршрута от корневого каталога начинается с символа \, как бы отделяющего не существующее имя корневого каталога, от имени первого подкаталога, подчиненного корневому.

При ссылке на файл маршрут включается в спецификацию файла, при этом следует за идентификатором

дискового накопителя (A:, B:, C:, ...) и отделяется от имени файла знаком \.

Например, чтобы обратиться к файлу FILE.DAT, хранящемуся в подкаталоге PETROV жесткого диска C:, надо в спецификацию файла включить: индентификатор диска C:, маршрут \PETROV и имя файла, отделив его от указателя маршрута символом \:

C:\PETROV\FILE.DAT

Полную спецификацию файла вводить в компьютер каждый раз неудобно. Поэтому операционная система позволяет объявить любой подкаталог на диске *рабочим*, и тогда к файлам его можно обращаться, не указывая маршрута. В исходном состоянии рабочим считается корневой каталог диска. Чтобы поменять рабочий каталог, используется специальная команда операционной системы. В спецификацию файла можно не включать и идентификатор дискового накопителя, если этот накопитель объявлен *текущим*. Присвоение устройству дисковой памяти статуса «текущее» выполняется по команде пользователя.

Операционными системами миниЭВМ и больших ЭВМ обычно предусматривается возможность объединения деревьев каталогов используемых дисков в одно общее дерево, называемое *главным каталогом системы*. Процедура включения корневого каталога диска со всеми его подкаталогами в главный каталог системы называется операцией *монтирования диска*. Она выполняется всякий раз после установки диска на любой из накопителей. По завершении работы с диском его можно снять с устройства, выполнив предварительно операцию *демонтирования диска*.

В процессе работы пользователей на компьютере содержимое диска меняется: добавляются новые файлы, удаляются ненужные, некоторые файлы расширяются и т. д. Реализация этих изменений требует наличия специального механизма распределения запоминающего пространства диска между файлами и обеспечения доступа к ним. В операционной системе персональных ЭВМ MS DOS этот механизм реализуется путем использования *таблицы размещения файлов* FAT (от англ. File Ablocation Table).

При выполнении операций чтения-записи данных обмен информацией между дисковым накопителем и памятью ЭВМ осуществляется блоками. Минимальный объем блока равен сектору. Чтобы уменьшить количество об-

ращений к дисковому накопителю, за одно обращение можно записать или считать информацию из нескольких последовательно расположенных секторов, образующих своеобразный суперблок, называемый *кластером*. Размер кластера зависит от используемого физического формата диска и может быть равен двум, четырем секторам и более. Файлу, записываемому на диск, выделяется определенное количество кластеров, причем выделяемые кластеры могут находиться в различных местах диска. В отличие от *непрерывных файлов*, хранящихся в одной области памяти, файлы, занимающие на диске несколько областей, называются *фрагментированными*. Назначение FAT — хранить данные о местонахождении на диске фрагментов (кластеров) файлов.

Механизм доступа к файлам с использованием FAT реализуется следующим образом. Область данных диска рассматривается как последовательность пронумерованных кластеров. Каждому кластеру ставится в соответствие элемент FAT с тем же номером. Например, элемент 2 соответствует кластеру номер 2 области данных диска, элемент 3 — кластеру номер 3 и т. д.

В каталоге, содержащем сведения о файлах на диске, для каждого файла указывается номер первого из кластеров, занимаемых файлом. Этот номер называется *точкой входа* в FAT. Система, прочитав в каталоге номер первого из занимаемых файлом кластеров, обращается к этому кластеру, например записывает в него данные, затем она обращается к соответствующему элементу FAT (элементу, номер которого равен номеру первого кластера файла). Этот элемент FAT содержит номер следующего кластера, отведенного для файла на диске, и т. д.

Пример, приведенный на рис. 5.4, иллюстрирует использование FAT при обращении к файлу, имеющему три фрагмента на диске. Как видно из рис. 5.4, *а*, первый фрагмент файла занимает кластеры с номерами 9, 10, второй фрагмент — кластер с номером 12. Кластер 11 не используется как дефектный (в процессе форматирования на этом участке диска были обнаружены дефекты поверхности, препятствующие нормальной записи данных). Третий фрагмент файла занимает кластеры с номерами 45, 46.

На рис. 5.4, *б* приведено условное изображение соответствующего фрагмента FAT и пояснены значения элементов для каждого из кластеров, изображенных на рис. 5.4, *а*. Кластеры с номерами 13, 44, 47 свободны, и поэтому значения их равны 0. Кластер номер 11 отмечен как

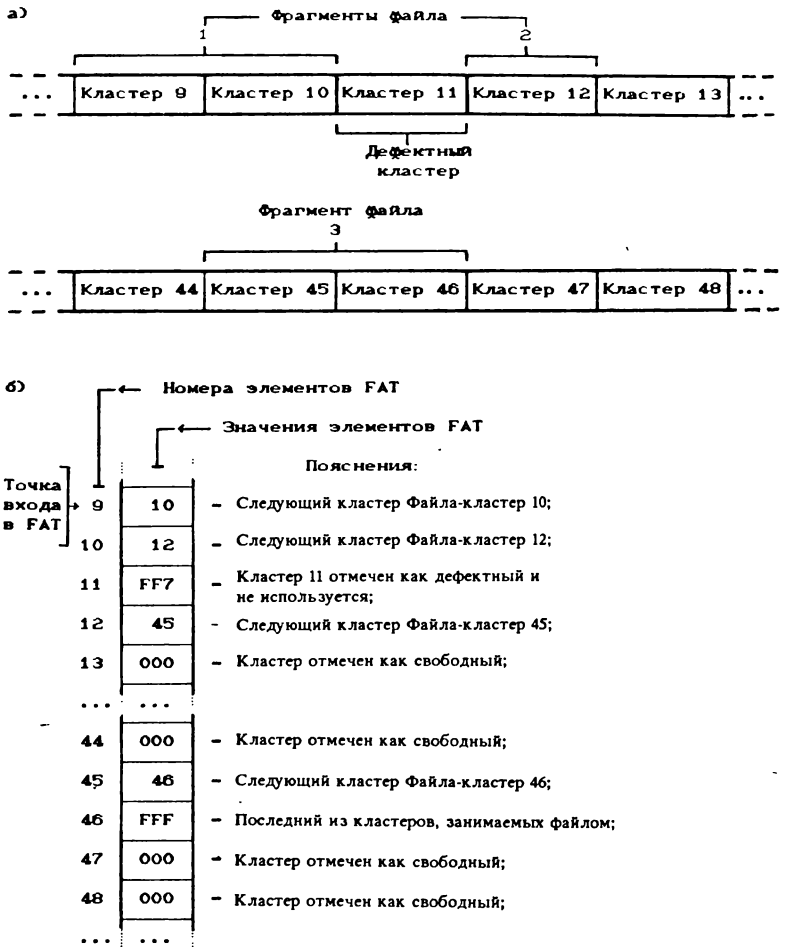


Рис. 5.4

дефектный специальным кодом FF7, а кластер номер 46 отмечен кодом FFF как последний кластер файла.

Если файл удаляется, то занимаемые им кластеры освобождаются и в соответствующие элементы FAT записывается код 000 — признак свободного кластера. При этом данные удаленного файла остаются на диске до тех пор, пока занимаемые ими кластеры не будут выделены системой другим файлам, вновь создаваемым или расширяемым. Пока этого не случится, удаленный файл может быть восстановлен.

Отметим, что использование ЕАТ — один из простейших способов реализации механизма распределения памяти диска и обеспечения доступа к файлам, широко применяемый в системах организации данных микроЭВМ. В миниЭВМ и больших ЭВМ для сокращения времени доступа к данным используются другие идеи, которым соответствует иная структура соответствующих вспомогательных файлов на дисках.

5.3. ФАЙЛЫ НА МАГНИТНЫХ ЛЕНТАХ

На магнитных лентах файлы располагаются последовательно один за другим. Чтобы прочитать данные, например, из 5-го по порядку файла, надо подвести ленту к началу, затем пропустить четыре первых файла и только после этого начать чтение данных из 5-го файла. Такая схема доступа к данным называется последовательной, и поэтому накопители на магнитных лентах (НМЛ) называются *устройствами последовательного доступа.*

Различают кассетные и катушечные НМЛ. Кассетные НМЛ обычно применяются в составе оборудования учебных микроЭВМ и микроЭВМ, ориентированных на использование в быту. Это могут быть обычные бытовые однодорожечные магнитофоны, при использовании которых в качестве запоминающего устройства на дорожку магнитной ленты записывается не музыка или речь, а импульсный электрический сигнал, представляющий собой последовательность высоких и низких уровней напряжения, соответствующую записываемой последовательности значений битов (0, 1) закодированной информации. Кассетные НМЛ являются вспомогательными и обычно используются только в микроЭВМ, в состав оборудования которых не включаются дисковые запоминающие устройства.

Более широкое применение в качестве устройств внешней памяти находят катушечные НМЛ, широко используемые в составе оборудования миниЭВМ и больших ЭВМ для ввода-вывода данных и программ, а также для переноса данных и программ с одной ЭВМ на другую. Параметры физической структуры катушечных магнитных лент (МЛ) и организация данных на них в значительной степени унифицированы, что позволяет использовать магнитные ленты для переноса данных не только между однотипными ЭВМ, но и между ЭВМ различных систем, например ЕС ЭВМ и СМ ЭВМ.

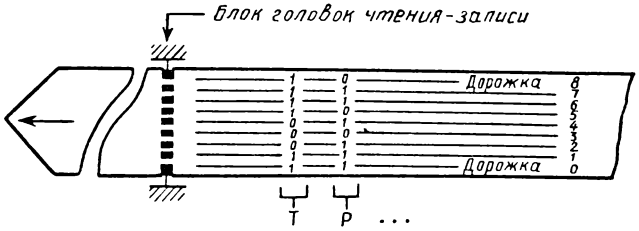


Рис. 5.5

В ЕС и СМ ЭВМ используются 9-дорожечные НМЛ, что позволяет записывать в каждой строке 8-битовой байт и контрольный бит. Значение контрольного бита формируется при записи байта на МЛ. При этом подсчитывается количество битов в байте, имеющих значение 1. Если число единиц нечетное, то контрольный бит принимается равным 1, как, например, для символа Т, запись которого условно изображена на рис. 5.5, если число единиц четное, то контрольный бит принимается равным 0, как, например, для символа Р. Таким образом, при чтении данных с МЛ сумма единиц в девяти битах каждой считанной строки должна быть четной. Если эта сумма оказывается нечетной, то устройство управления НМЛ фиксирует ошибку чтения данных.

Описанный принцип контроля считываемых с МЛ данных, называемый *контролем на четность*, как один из простейших и достаточно эффективных, используется, в частности, для контроля правильности передачи данных между устройствами ЭВМ.

Данные на МЛ записываются посимвольно (в ЕС ЭВМ в коде ДКОИ) с плотностью 64 или 32 байт/мм. Таким образом, при длине ленты в 500 м ее расчетная емкость при повышенной плотности записи составит около 30 Мбайт. Фактическая емкость будет значительно меньше, так как данные на МЛ записываются блоками, которые отделяются друг от друга межблочными промежутками длиной примерно в 15 мм.

Блоки на МЛ располагаются между физическими маркерами начала и конца ленты, в качестве которых используются полоски фольги, наклеиваемые на поверхность ленты в 4—5 м от ее начала и конца.

Чтобы создать определенную файловую структуру на МЛ, лента так же, как и диск, должна быть предварительно проинициализирована. В результате инициализации вслед за маркером «начало ленты» записывается, как

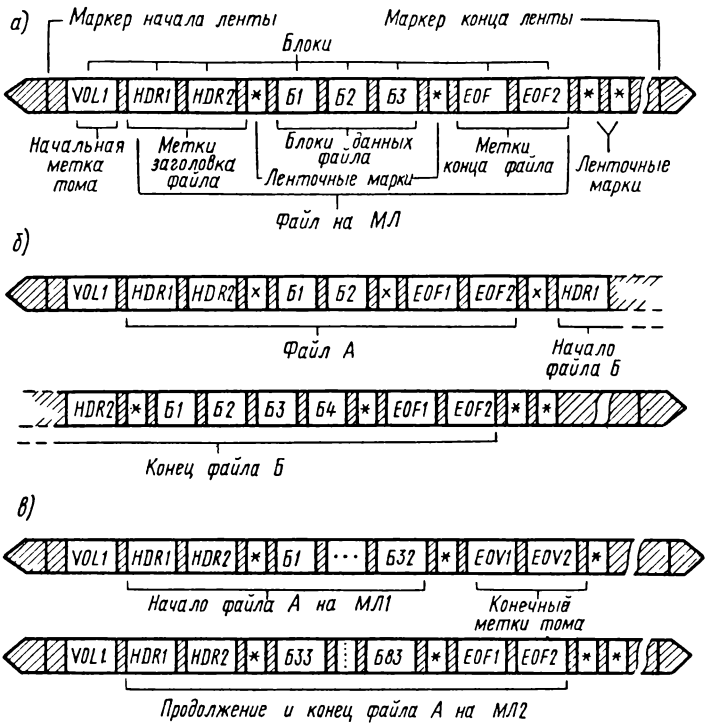


Рис. 5.6

минимум, один 80-байтный блок «метка тома», имеющий обозначение VOL1. В зависимости от типа операционной системы и параметров инициализации могут быть выполнены и другие действия: запись пустого файла, проверка всей ленты на наличие дефектных участков и т. д.

В метку тома VOL1 записываются обозначение метки (VOL1), имя или идентификатор тома, идентификация владельца, которому принадлежит этот том, и другие параметры.

После метки VOL1 на ленту записывается первый файл (рис. 5.6, *а*). Файл выводится на ленту блоками определенной длины и имеет, как минимум, две метки заголовка файла (HDR1 и HDR2) и две конечные метки (EOF1, EOF2). Каждая из них занимает блок длиной 80 байт.

Метка HDR1 содержит обозначение метки HDR1, имя файла, порядковый номер файла на томе, дату создания и другие параметры.

Метка HDR2 наряду с другими параметрами содержит сведения о длине блока информационной части файла, а также о типе и длине записей файла.

Метки EOF1 и EOF2 по содержанию аналогичны меткам HDR1 и HDR2. Они отличаются обозначениями меток и наличием в блоке EOF2 дополнительного параметра, указывающего размер файла в блоках. Блоки файла (Б1, Б2, Б3) отделяются от боков начальных и конечных меток так называемой *ленточной маркой* (*), которая рассматривается как отдельный блок длиной 2 байт. Ленточная марка опознается и используется устройством управления НМЛ. Содержащиеся в ней коды не считываются в основную память ЭВМ. Конечные метки файла отделяются от начальных меток следующего файла также ленточной маркой (*). В конце последнего файла на МЛ после конечных меток записываются две ленточные марки (*) (*).

На рис. 5.6, *а* представлен простейший состав блоков файловой структуры для случая, когда на одной ленте записан один файл. На рис. 5.6, *б* показана последовательность стандартных меток, ленточных марок и блоков данных для случая, когда на МЛ записано два файла.

На ЭВМ могут обрабатываться файлы, для размещения которых потребуется несколько магнитных лент. Такие файлы называются *многотомными*. Пример размещения на МЛ двухтомного файла приведен на рис. 5.6, *в*. В конце занимаемой файлом первой МЛ вместо конечных меток файла записываются конечные метки томов EOV1 и EOV2, указывающие на то, что файл имеет продолжение на следующем томе.

Наличие на МЛ стандартной файловой структуры позволяет обращаться к файлам по их именам. Новый файл размещается на МЛ обычно вслед за последним, ранее записанным файлом. Операция записи файла на ленту включает в себя три этапа: открытие файла, вывод данных в файл, закрытие файла.

В процессе *открытия файла* отыскивается конец последнего существующего на МЛ файла и вслед за ним записываются начальные метки (HDR1, HDR2открываемого файла. Вывод данных в файл осуществляется поблочно. После вывода последнего блока выполняется *закрытие файла*, состоящее в записи на ленту конечных меток файла (EOF1, EOF2) и двух ленточных марок (*) (*), указывающих на начало свободного пространства МЛ.

Для считывания данных из файла необходимо предварительно выполнить команду *открыть файл для чте-*

ния, по которой файловая система осуществит поиск файла на МЛ и подведет к головкам чтения данных первый информационный блок файла.

Файловые системы ОС предусматривают также возможность работы с МЛ, не имеющими стандартных меток. При этом информация, записанная на МЛ, рассматривается как один файл, состоящий из последовательности блоков данных, не обязательно одинаковой длины, размещенных за маркером начала ленты. Для выполнения операций чтения-записи данных в этом случае программисту предоставляется набор следующих основных команд: перемотать ленту в ее начало; установить значение длины блока данных; записать блок данных; прочитать блок данных; пропустить указанное количество блоков; вернуть ленту назад на указанное количество блоков и др.

Посредством таких команд на ЭВМ обрабатываются «чужие» ленты, имеющие файловую структуру, отличную от той, которая создается используемой на данной ЭВМ операционной системой.

5.4. ВВОД-ВЫВОД ДАННЫХ И ОРГАНИЗАЦИЯ ФАЙЛОВ

При выполнении программы на ЭВМ любая последовательность вводимых или выводимых данных рассматривается как файл независимо от того, являются эти данные файлом, сохраняемым на магнитных дисках и лентах, или представляют собой поток символов, которым программа обменивается с одним из устройств — печати, клавиатурой, дисплеем и т. п.

Файлы на дисках и лентах имеют уникальные имена, хранятся длительное время и повторно используются по мере необходимости. Файлы данных, которыми программа обменивается с другими устройствами, существуют лишь в процессе ввода-вывода и в уникальных именах не нуждаются. В качестве имен таких файлов используются *имена внешних устройств,* полный список которых для типового комплекта оборудования ЭВМ устанавливается операционной системой. Например, если в типовую конфигурацию внешних устройств ЭВМ входят два печатающих устройства, то они могут обозначаться именами LP1:, LP2:, дисплеи, подключенные к ЭВМ, если их несколько,— именами T1:, T2: и т. д. Соответственно файл данных, выводимый на печать через устройство LP1:, именуется как файл LP1:, файл данных, выводимый на терминал T1:,— как файл T1: и т. д.

Разрабатываемая программа может обращаться к файлам на внешних устройствах по именам, но это не всегда удобно, так как жестко привязывает программу к устройствам конкретной ЭВМ. Большинство современных систем программирования обеспечивают *независимость программ* от особенностей конфигурации внешних устройств, что позволяет при выполнении программы в случае неисправности любого из внешних устройств ЭВМ заменить его однотипным. Встречаются ситуации, когда требуется *переназначить* устройство ввода-вывода при выполнении программ. Так, например, может потребоваться, чтобы программа, обычно осуществляющая вывод данных на экран дисплея, в очередной раз вывела их на печатающее устройство или записала на диск.

Аппаратная независимость ввода-вывода дастигается за счет того, что разрабатываемые программы обычно не адресуются непосредственно к файлам внешних устройств. Файлам программ присваиваются идентификаторы, в качестве которых используются имена, подобные именам переменных, либо так называемые *логические номера файлов*. Например, если файл с логическим номером $\neq 1$ закреплен за устройством печати, то команда, предписывающая переслать содержимое файла на диске с именем FILE.DAT в файл с логическим номером $\neq 1$, при исполнении программы на ЭВМ интерпретируется как команда вывода файла FILE.DAT на печать.

Установление соответствия между логическими номерами файлов в программе и внешними устройствами ЭВМ осуществляется либо посредством специальных команд, включаемых в программу при подготовке ее к выполнению, либо посредством команд операционной системы, позволяющих указать конкретные условия выполнения программы на ЭВМ. При необходимости переназначение устройств ввода-вывода можно произвести непосредственно в процессе выполнения программы.

Один из возможных форматов команды *назначения* устройства логическому номеру файла можно проиллюстрировать следующим примером:

```
ASSIGN    LP1:=≠1
```

Приведенная команда назначает (ASSIGN — англ. НАЗНАЧИТЬ) файлу с логическим номером $\neq 1$ печатающее устройство LP1:.
Назначение

```
ASSIGN    T1:=≠1
```

заставит программу вывести тот же файл на экран дисплея (терминал T1:), а назначение

```
ASSIGN    B:FILE.DAT = ≠1
```

— записать файл ≠1 на диск B: под именем FILE.DAT. Наряду с логическими номерами для обозначения файлов в программах используются логические имена устройств. Так, файл, выводимый на печатающее устройство в программе, может иметь обозначение LP, которое трактуется как «любое печатающее устройство». Конкретное назначение в этом случае выполняется операционной системой автоматически (назначается любое свободное устройство), либо оператором, например, по команде:

```
ASSIGN LP2:=LP:,
```

назначающей логическому имени LP: конкретное печатающее устройство, известное операционной системе под именем LP2:.

Установление соответствия между логическим номером файла в программе и файлом на диске или магнитной ленте обычно производится при выполнении команды открытия файла для чтения или записи. Следующая команда является примером открытия файла в программе, записанной на языке БЕЙСИК:

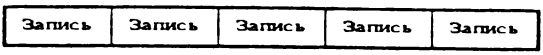
```
1030    OPEN    "B:FILE.DAT"    AS≠1
```

Команда открывает файл FILE.DAT на диске B: и назначает ему логический номер ≠1. Все последующие команды программы для ввода и вывода данных из этого файла обращаются к нему по логическому номеру ≠1.

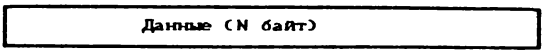
Любой запрос программы на выполнение ввода-вывода адресуется файловой системе ОС, которая, получив такой запрос, выполняет затребованные действия на командном уровне процессора. Важной функцией файловой системы является установление соответствия между определенной программистом внутренней логической структурой файла программы и совокупностью хранящих его физических блоков внешнего устройства.

На логическом уровне программы файл представляет собой набор отдельных записей (рис. 5.7, *а*). Размер записи и структура ее внутренних полей определяются программой пользователя. Записи могут быть *фиксированной, переменной* и *неопределенной* длины. Размер записи фиксированной длины (рис. 5.7, *б*) является ат-

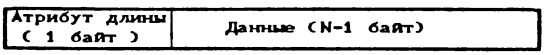
а) Файл данных



б) Запись фиксированной длины



в) Запись переменной длины



г) Запись неопределенной длины

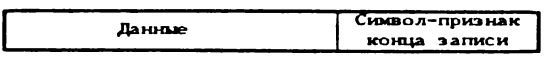
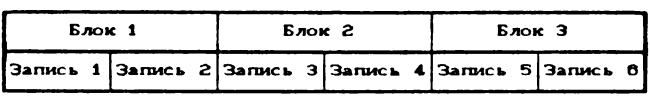


Рис. 5.7

а)



б)

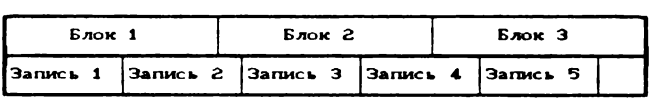


Рис. 5.8

рибутом файла. Размер записи переменной длины (рис. 5.7, *в*) является атрибутом записи и хранится вместе с данными записи как отдельное поле. Записи неопределенной длины (рис. 5.7, *г*) отделяются друг от друга в файле специальными *символами-разделителями.*

На физическом уровне (уровне устройства) файл является набором выделенных ему блоков памяти, каждый из которых хранит одну из нескольких записей файла. Если длина блока кратна длине записи, то граница последней записи блока совпадает с границей блока (рис. 5.8, *а*), если длина блока не кратна длине записи, то последняя запись блока пересекает его границу и заканчивается в следующем блоке (рис. 5.8, *б*). Возможность работать с записями, пересекающими границы блоков, обеспечивается не во всех операционных системах и не для любой организации файлов.

Наиболее широко используются файлы, имеющие последовательную и произвольную организацию.

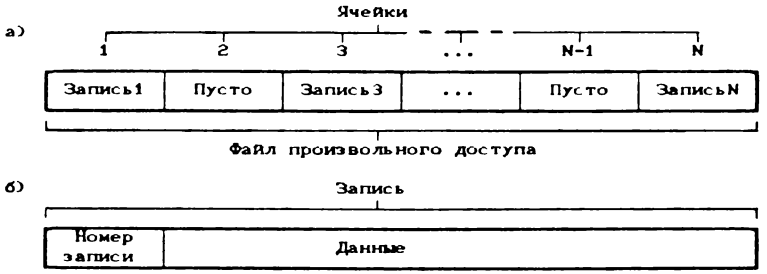


Рис. 5.9

При последовательной организации логические записи располагаются в файле в порядке их поступления. Считывание данных возможно лишь в порядке их физического расположения. Последовательно организованные файлы хранят записи любых типов — фиксированной, переменной, неопределенной длины. Последовательная организация характерна для файлов на магнитных лентах. Как последовательно организованные можно рассматривать файлы на устройствах печати, файлы данных на перфолентах и перфокартах.

Файлы произвольного доступа создаются только на дисковых запоминающих устройствах. Файл произвольного доступа организован как последовательность ячеек фиксированной длины (рис. 5.9, *а*). Размер каждой ячейки одинаков и равен максимальной длине записи. Ячейки имеют номера от 1 до N, где N — максимальное число записей файла. Номер ячейки указывает ее место относительно начала файла, поэтому такую организацию файла называют также относительной организацией. Каждая ячейка содержит только одну запись. В файле наряду с заполненными ячейками могут оставаться и пустые. Номера ячейки соответствует номеру хранящейся в ней записи.

Наиболее характерны для файлов с произвольной организацией записи фиксированной длины (рис. 5.9, *б*). Номер записи является ее атрибутом. При обращении к записи программист указывает номер. Файловая система по номеру записи устанавливает место нахождения на диске ячейки с соответствующим номером и, непосредственно обращаясь к ней, считывает из нее или записывает данные. К записям файлам с произвольной организацией возможен и последовательный доступ. Организация файла определяется в процессе его создания, метод доступа — в процессе его открытия.

Последовательная организация является простой, эко-

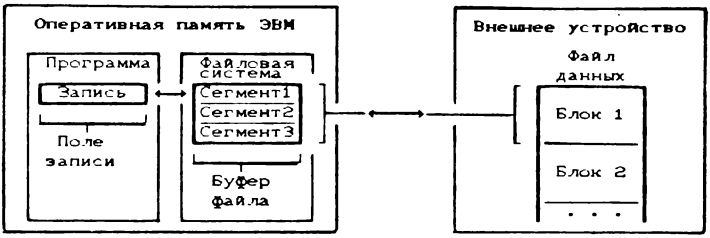


Рис. 5.10

номно расходует память, обеспечивает возможность пересечения записями границ блоков, легко переносится с одного типа внешнего устройства ЭВМ на другое. К ее недостаткам относятся: сравнительно медленный доступ к данным — чтобы выбрать запись, надо предварительно выбрать все ей предшествующие; невозможность произвольного удаления и вставления записей — удалять и добавлять записи можно только в конце файла. Файлы произвольной организации свободны от этих недостатков, но могут создаваться только на дисках и требуют избыточной памяти по сравнению с последовательно организованными.

Обработку любого файла в программе можно начать только тогда, когда он подготовлен к работе. Подготовка файла к обмену осуществляется путем выполнения *операции открытия файла.* В большинстве систем программирования этой операции соответствует команда OPEN (англ. ОТКРЫТЬ). Используются различные модификации команды OPEN: открыть файл для чтения; открыть файл для записи; открыть файл для дополнения и др.

В программе одновременно можно открыть несколько входных и выходных файлов. Если открывается новый файл (несуществующий), то по команде OPEN файловая система создает требуемый файл на указанном томе и подготавливает созданный файл к записи в него данных. Если открывается существующий файл, то файловая система определяет его местоположение и подготавливает для выполнения затребованной операции.

Во всех случаях открытому файлу выделяется специальная область оперативной памяти, называемая *буфером файла* (рис. 5.10). Поскольку обмен данными между внешними устройствами и оперативной памятью осуществляется поблочно, размер буфера, выделяемого файлу, берется равным размеру физического блока внешнего

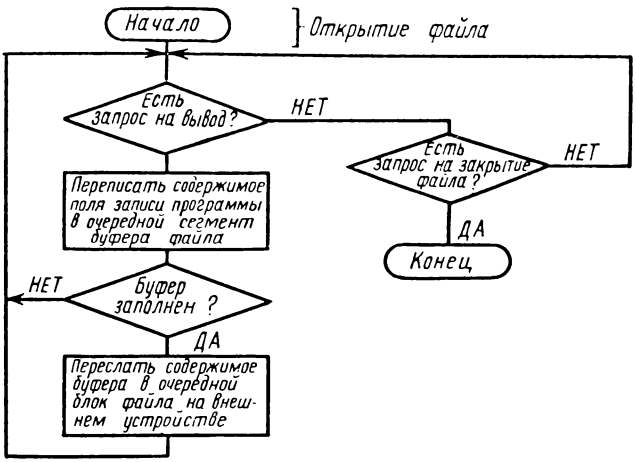


Рис. 5.11

устройства, принимающего (посылающего) данные файла. Части буфера, в которых хранятся записи, называются *сегментами*. Буфер файла предназначается для временного хранения данных, считанных в оперативную память или подготавливаемых для вывода на внешние устройства.

В большинстве систем программирования вывод записи в файл осуществляется по команде PUT, ввод — по команде GET. Но команды PUT и GET не всегда вызывают реальное выполнение операции ввода-вывода, а только в тех случаях, когда команда GET считывает последнюю запись из хранящегося в буфере очередного блока данных файла или когда в результате выполнения команды PUT буфер полностью заполняется.

Обобщенные алгоритмы, реализуемые файловой системой при поблочном вводе и выводе данных последовательно организованного файла, приведены соответственно на рис. 5.11 и 5.12. Начало выполнения обоих алгоритмов связано с открытием файлов, окончание — с выполнением операции закрытия файла CLOSE. Команда CLOSE прекращает работу программы с файлами, выводит для выходных файлов их конечные метки, освобождает буферную память файла. Так как операции обмена данными между буфером файла и внешним устройством выполняются файловой системой автоматически, для программиста сохраняется иллюзия того, что каждая выводимая или вводимая запись поступает непосредственно в файл или из файла.

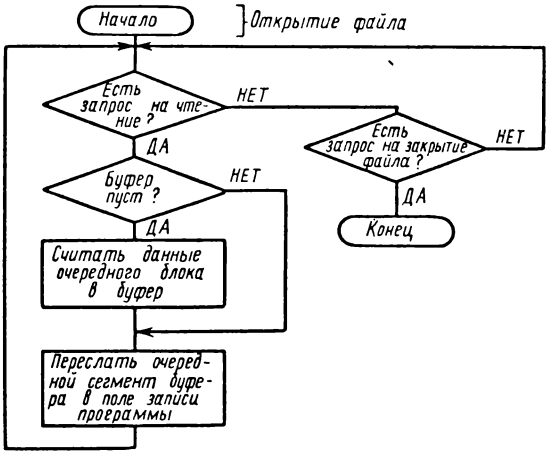


Рис. 5.12

Правила записи команд OPEN, CLOSE, GET, PUT и др., используемых в программах для выполнения операций ввода-вывода, приводятся в описаниях алгоритмических языков. Познакомиться с ними более подробно можно в последующих книгах данной серии.

Вопросы для самопроверки

5.1. Что такое файл данных?

5.2. Приведите примеры имен файлов.

5.3. Для чего используется расширение имени файла?

5.4. Приведите примеры записи имен файлов с использованием символов (*) и (?). Дайте пояснения.

5.5. Назовите основные параметры физической структуры магнитных дисков.

5.6. Составьте расчеты емкости для гибких дисков форматов: D-9; QD-18.

5.7. Что такое кластер?

5.8. Объясните назначение каталога диска. Что такое подкаталог, корневой каталог?

5.9. Что такое спецификация файла?

5.10. Поясните назначение операций форматирования, инициализации, монтирования диска.

5.11. Поясните различие между понятиями «непрерывный файл» и «фрагментированный файл».

5.12. Для чего необходима таблица размещения файлов (FAT)? Используя рис. 5.4, поясните использование FAT для определения местонахождения файла на диске.

5.13. Поясните разницу между устройствами прямого и последовательного доступа.

5.14. Поясните, как осуществляется проверка правильности чтения

данных с магнитной ленты с использованием принципа контроля на четность.

5.15. Используя рис. 5.6, поясните приведенные на нем варианты реализации файловой структуры магнитной ленты. Поясните назначение встречающихся на рис. 5.6 ленточных марок и меток.

5.16. Поясните, каким образом обеспечивается независимость ввода-вывода программы от особенностей конфигурации внешних устройств ЭВМ.

5.17. Как организован последовательный файл? Как осуществляется доступ к записям последовательного файла?

5.18. Как организован файл произвольного доступа? Как осуществляется доступ к записям файла произвольного доступа?

5.19. Объясните назначение команд доступа к файлу: OPEN, PUT, GET, CLOSE.

СПИСОК ЛИТЕРАТУРЫ

1. *Глушков В. М.* Основы безбумажной информатики.— М.: Наука, 1987.
2. Персональные компьютеры Единой системы ЭВМ/ *Запольский А. П., Пыхтин В. Я., Чистяков А. Н., Шкляр В. Б.*— М.: Финансы и статистика, 1988.
3. *Кибернетика. Становление информатики:* Сб. статей.— М.: Наука, 1986.
4. *Мэрфи Дж.* Как устроены и работают электронные цифровые машины: Пер. с англ.— М.: Мир, 1965.
5. *Перегудов М. А., Халамайзер А. Я.* Бок о бок с компьютером.— М.: Высшая школа, 1987.
6. *Пул Л.* Работа на персональном компьютере: Пер. с англ.— М.: Мир, 1986.
7. *Савельев А. Я.* Арифметические и логические основы цифровых автоматов.— М.: Высшая школа, 1984.
8. Современный компьютер: Сб. научн.-попул. статей: Пер. с англ. /Под ред. В. М. Курочкина. — М.: Мир, 1986.
9. *Трейстер Р.* Персональный компьютер фирмы ИБМ: Пер. с англ.— М.: Мир, 1986.
10. *Черемных С. В., Гиглавый А. В., Поляк Ю. Е.* От микропроцессоров к персональным ЭВМ.— М.: Радио и связь, 1988.

Оглавление

Учебное издание

ЭЛЕКТРОННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ

В восьми книгах

Савельев Александр Яковлевич
Сазонов Борис Алексеевич
Скуратович Эдуард Константинович
Когдов Николай Михайлович

Книга 2

ОСНОВЫ ИНФОРМАТИКИ

Заведующая редакцией *Н. И. Хрусталева.* Редактор *С. М. Оводова.* Младший редактор *Г. Г. Бучина.* Художник *Ю. Д. Федичкин.* Художественный редактор *Т. М. Скворцова.* Технический редактор *Л. А. Овчинникова.* Корректор *Г. А. Чечеткина.* Программист *И. А. Грохочинская.* Операторы *С. Р. Луковенкова* и *В. Н. Новоселова.*

ИБ № 8812

Изд. № СТД-698. Сдано в набор 26.02.91. Подп. в печать 10.10.91. Формат 84 × 108/32. Бум. тип. № 2. Гарнитура Таймс. Печать высокая. Объем 8,40 усл. печ. л. 8,82 усл. кр.-отт. 8,20 уч.-изд. л. Тираж 140000 экз. Зак. № 530. Цена 1 р 50 к.

Издательство «Высшая школа», 101430, Москва, ГСП-4, Неглинная ул., д. 29/14.

Набрано на персональном компьютере издательства.

Отклонения по качеству обусловлены состоянием оригиналов, представленных для съемки.

Государственная ассоциация предприятий, объединений
и организаций полиграфической промышленности «АСПОЛ».
Ярославский полиграфкомбинат, 150049, Ярославль, ул. Свободы, 97.

1р. 50к.

Введение в ЭВМ
Основы информатики
Технология подготовки задач для решения на ЭВМ
Языки программирования (ФОРТРАН IV, ФОРТРАН 77, ПЛ/1)
Языки программирования (ПАСКАЛЬ, ПЛ/М)
Средства общения с ЭВМ
Практикум по программированию
Решение прикладных задач