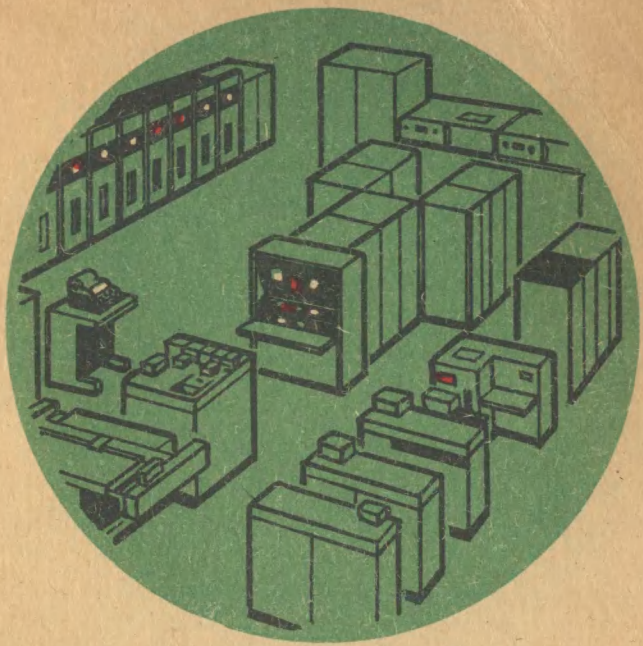
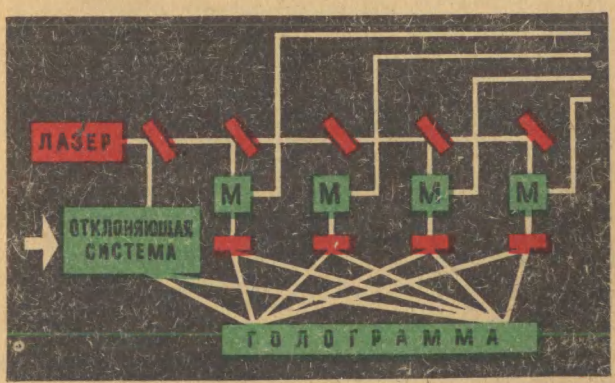


Р. С. ГУТЕР, Ю. Л. ПОЛУНОВ

Математические машины



Р. С. ГУТЕР и Ю. Л. ПОЛУНОВ

Математические МАШИНЫ

ОЧЕРКИ
ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

ПОСОБИЕ ДЛЯ УЧИТЕЛЕЙ

Москва «Просвещение» 1975

Гутер Р. С. и Полунов Ю. Л.

Г 97 Математические машины. Очерки вычислительной техники. Пособие для учителей. М., «Просвещение», 1975

287 с.

Г $\frac{60501\text{-}724}{103(03)\text{-}75}$ Б3-38-24-1975 **518**

ПРЕДИСЛОВИЕ

Наша книга посвящена *математическим* или *вычислительным машинам*. В первую очередь нас здесь интересует, что это *машины*, и лишь во вторую, что они *математические*. Это означает, что основным содержанием книги является описание устройства вычислительных машин, принципов их работы и элементов, из которых они устроены. Но так как эти машины все-таки математические, то мы не могли обойти и чисто математических вопросов, например программирования.

В предисловиях авторы обычно пишут, какому кругу читателей данная книга предназначается. Мы хотим начать с другой стороны, написавши сначала, на кого она не рассчитана. Прежде всего, ее не следует читать специалистам по вычислительной технике. Кроме того, ее нельзя рассматривать ни как учебник, ни как справочник по программированию, ввиду ее неполноты в этом отношении и совсем иного методического подхода, вызванного тем, что в ней на первом плане стояли вопросы устройства машин.

Свою книгу мы предназначаем, в первую очередь, учителям физики и математики и другим специалистам смежных специальностей, которые хотят ознакомиться с принципами устройства и работы современных вычислительных машин и их применением. Полагаем, что она написана достаточно популярно и будет вполне доступна не только учителям, но и школьникам старших классов. Во всяком случае, мы не выходили за пределы школьного курса физики и математики, кроме нескольких мест, не играющих существенной роли.

К сожалению, ограничение объема книги заставило нас исключить материал, относящийся к аналоговым и гибридным машинам, а также описание ряда деталей устройства цифровых машин последних поколений и многие вопросы их применения. Тем не менее мы полагаем, что наша книга

даст возможность достаточно хорошо познакомиться с устройством цифровых машин второго и третьего поколений.

Что касается программирования, то при описании машинных языков мы сочли возможным ограничиться машинами второго поколения, поскольку работа на машинах третьего поколения происходит преимущественно через алгоритмические языки. Для знакомства с ними мы впервые воспользовались параллельным изложением двух наиболее распространенных и близких между собой языков — алгола и фортрана.

Расположение материала и само содержание книги были улучшены благодаря замечаниям члена-корреспондента АН СССР Н. Н. Моисеева, ознакомившегося с первоначальным планом и проспектом книги. Мы выражаем ему свою признательность.

Мы благодарим рецензентов, члена-корреспондента АПН СССР С. И. Шварцбурда и доктора педагогических наук В. М. Монахова, внимательно ознакомившихся с рукописью и высказавших нам ряд полезных замечаний и советов.

Мы благодарим также Т. А. Муратову за неоценимую помощь при подготовке рукописи к изданию и ряд замечаний по существу.

ВВЕДЕНИЕ

Считать приходилось человеку уже очень давно. Самые древние из дошедших до нас источников — египетские папирусы — содержат описания правил вычисления различных величин. При этом, естественно, использовались некоторые вполне определенные способы наименования и записи чисел, т. е. определенная система счисления.

Первые приспособления, которые использовались для облегчения счета, например абак или русские счеты (подробнее см. гл. II), облегчали, собственно, не столько сам процесс счета, сколько запоминание исходных чисел и в особенности промежуточных результатов. По этой причине числа в этих приспособлениях изображались в своей привычной цифровой форме в определенной системе счисления (почти всегда — десятичной, которая к моменту создания упоминаемых приспособлений стала общепризнанной). Такими же были и первые вычислительные машины. Казалось, что иначе и быть не может; однако выяснилось, что может быть и иначе.

В начале XVII века были придуманы логарифмы, затем логарифмическая шкала и в 1622—1623 годах первый вариант логарифмической линейки, которая оказалась вычислительным приспособлением совсем другого типа. Здесь изображением числа служит некоторая точка шкалы или, точнее говоря, отрезок, начало которого всегда фиксировано и совпадает с началом шкалы, а конец определяется изображаемым числом.

Различие в способах представления чисел на русских счетах и логарифмической линейке — в физической природе величин, применяемых для этой цели: на линейке число изображается длиной отрезка, т. е. величиной, имеющей непрерывный характер и могущей принимать любые действительные значения.

Казалось бы, использование непрерывных величин позволяет достичь большой точности в изображении числа.

Однако на самом деле повышения точности не происходит. Наоборот, достигнуть большой точности на логарифмической линейке или используя для изображения числа какие-либо другие непрерывные величины весьма затруднительно.

Точность отсчета числа на логарифмической линейке определяется длиной шкалы. То же относится и к тому случаю, когда для изображения числа применяется любая другая непрерывная величина: точность отсчета числа будет определяться точностью измерения соответствующей величины.

Итак, для изображения чисел в вычислительных машинах и устройствах можно использовать различные принципы. Это различие положено в основу классификации математических машин, которой посвящена первая глава нашей книги.

Глава I

РАЗЛИЧНЫЕ ТИПЫ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

§ 1. СПОСОБЫ ИЗОБРАЖЕНИЯ ЧИСЕЛ

Как было уже сказано во введении, существует два различных способа изображения чисел в вычислительных машинах и приборах.

Первый из них можно условно назвать *цифровым*. Этот способ реализуется в большом количестве различных приспособлений и машин, простейшим из которых являются хорошо всем известные русские счеты. Поэтому при подробном рассмотрении цифрового способа изображения чисел мы будем для иллюстрации использовать прежде всего их.

Мы предполагаем, что читатель знаком с первоначальными сведениями о системах счисления. Ограничимся кратким сообщением о них, отсылая за более подробными сведениям к другим источникам *.

Системой счисления называют способ наименования и записи чисел. Общеупотребительной является *позиционная десятичная* система счисления. Это означает, что одна и та же цифра изображает различные числа в зависимости от занимаемой ею позиции, а каждый следующий слева разряд в десять раз больше предыдущего. Отношение соседних разрядов называется *основанием* системы счисления. Запись числа в данной системе означает его представление в виде суммы степеней основания с коэффициентами, роль которых играют цифры; степени могут быть как положительными, так и отрицательными. Например,

$$73534{,}261 = 7\cdot 10^4 + 3\cdot 10^3 + 5\cdot 10^2 + 3\cdot 10^1 + 4\cdot 10^0 + 2\cdot 10^{-1} + 6\cdot 10^{-2} + 1\cdot 10^{-3}.$$

Легко догадаться, что количество различных цифр в данной системе счисления равно основанию этой системы — различными цифрами обозначаются все числа, меньшие основания, которые в данной системе будут однозначны. Само основание всегда записывается как 10.

Широкое распространение получила *двоичная* система счисления; в ней для изображения числа достаточно лишь двух цифр — 0 и 1. Запись числа в двоичной

* См., например, сборник «Дополнительные главы по курсу математики для факультативных занятий. 7—8 классы», Гутер Р. С. Системы счисления и арифметические основы работы электронных вычислительных машин, 1974. Более кратко те же сведения можно найти в статье Гутер Р. С. Вычислительные машины и системы счисления. «Квант», 1971, № 9, более полно и подробно — в книге Фомин С. В. Системы счисления. Серия «Популярные лекции по математике». М., 1964 (и более поздние издания).

системе много длиннее десятичной: например, число 5 записывается в виде 101, число 9 — 1001, а число 18—10010. Поэтому для записи двоичных чисел часто используется *смешанная двоично-восьмеричная система.*

Возьмем число в восьмеричной системе и переведем каждую цифру этого числа отдельно в двоичную систему. Такая запись и называется двоично-восьмеричной. Для каждой восьмеричной цифры отводится три двоичных разряда, что достаточно, поскольку самая большая восьмеричная цифра 7 в двоичной системе записывается как 111. Оказывается (из-за того, что $8=2^3$), что двоично-восьмеричная запись совпадает с двоичной. Так, например,

$$572_8=101\ 111\ 010=5\cdot 8^2+7\cdot 8^1+2=378_{10},$$ с другой стороны,

$$101111010_2=2^8+2^6+2^5+2^4+2^3+2^1=378_{10}.$$

Двоичное число можно рассматривать поэтому, как двоично-восьмеричное; для записи его разбивают на группы по 3 разряда (триады) и каждую триаду заменяют восьмеричной цифрой. Например,

$$100101001110_2=100\ 101\ 001\ 110=4516_8.$$

Запишем изображаемое число в некоторой выбранной системе счисления, например привычной десятичной. Для изображения числа в данной машине или приспособлении отводится определенное количество разрядов. Каждый разряд представляет собой некоторый *элемент*, обладающий каким-то числом различных устойчивых состояний. При записи числа в десятичной системе счисления таких состояний должно быть десять — по числу различных цифр в данной системе. Различные состояния элемента принимаются за изображения различных цифр.

Обратимся к примерам. На русских счетах элементом для изображения отдельного разряда числа является спица с надетыми на нее десятью косточками. Различные состояния этого элемента определяются количеством опущенных вниз косточек. Они и применяются для изображения различных цифр числа. При этом необходимо иметь в виду, что отдельные состояния элемента должны быть *различимы*: случай, когда одна из косточек находится в промежуточном положении, недопустим и предполагается невозможным.

Другим примером устройства с цифровым способом изображения чисел является арифмометр. Здесь элементом для изображения отдельного разряда служит шестеренка с десятью зубьями. Она может поворачиваться вокруг своей оси не на любой угол, а только на углы, кратные 36°, так, чтобы какой-либо из ее зубов останавливался против окошка. Каждое возможное положение шестеренки соответствует определенной цифре, которая пишется на зубе и может быть прочитана в окошке.

Уже по этим двум примерам можно понять связь между выбранной системой счисления и характером элемента, применяемого для изображения отдельного разряда. Основание системы счисления определяет число различных цифр и тем самым число различных состояний, в которых может находиться элемент. Десять косточек на спице и десять зубцов на шестеренке определяются тем, что нам надо изображать десять различных цифр, что в свою очередь определяется десятичной системой счисления.

Слово «элемент», которое мы выше использовали, следует понимать достаточно широко. Нередко такой элемент может на самом деле представлять собой объединение нескольких. Например, в одних электромеханических вычислительных машинах элементом для изображения десятичной цифры служит количество электрических импульсов, посылаемых по одному проводу, а в других — единственный импульс, посланный по одному из десяти проводов.

Такой элемент можно сконструировать из других, имеющих небольшое число естественных различных состояний. Для различных электрических и электронных элементов число таких состояний обычно равно д в у м. Так, электрическая лампочка может гореть или не гореть, выключатель может быть включен или выключен, конденсатор — заряжен или разряжен, электронная лампа — проводить или не проводить ток.

Из нескольких таких элементов можно скомбинировать один, имеющий нужное число различных состояний. Однако более удобно перейти к системе счисления с другим основанием. Учитывая сказанное выше, естественно выбрать для изображения чисел двоичную систему, в которой используются лишь две различные цифры, 0 и 1. Тогда для изображения числа можно использовать элементы с двумя различными состояниями. Но дело и не только в этом. Для большой вычислительной машины существенно не столько число элементов (разрядов), сколько общее число различимых устойчивых состояний всех элементов, необходимых для изображения чисел.

Легко убедиться, что и с этой точки зрения двоичная система счисления выгоднее десятичной. Например, для машины, работающей в десятичной системе счисления, для изображения любого целого числа в интервале от 1 до 999 требуется три элемента ($10^3 = 1000$), имеющие по 10 состояний, итого 30 различимых состояний всех элементов. В то же время при использовании двоичной системы счисления каждый элемент имеет лишь два состояния, а для изображения целых чисел такого же интервала требуется 10 элементов ($2^{10} = 1024$), т. е. всего 20 различных состояний всех элементов. Высказанное утверждение нетрудно и доказать в общем виде.

В т о р о й способ изображения чисел называют *непрерывным* или *аналоговым*. Здесь для изображения числа применяется некоторая физическая величина, имеющая непрерывный характер, т. е. могущая принимать любое действительное значение. Это может быть длина отрезка или угол поворота вала, ток или напряжение в электрической цепи, температура, давление и другие подобные величины. Простейшим примером вычислительного устройства с непрерывным способом изображения чисел является логарифмическая линейка. Здесь, как мы уже говорили во введении, число изображается длиной отрезка.

Точность представления числа при цифровом способе изображения определяется количеством разрядов, т. е. количеством элементов, имеющихся в данной машине или в данном приспособлении.

Для увеличения точности достаточно увеличить количество отводимых для изображения числа элементов. Так, на русских счетах нужно увеличить количество спиц с косточками, а на арифмометре — количество шестеренок. Конечно, первое сделать много проще, чем второе, но и для второго трудности носят лишь технический, а не принципиальный характер.

Напротив, при непрерывном способе изображения числа точность его представления определяется точностью измерения изображающей число величины. Например, если число изображается напряжением в некоторой электрической цепи, то точность, с которой мы знаем это число, определяется точностью вольтметра, измеряющего соответствующее напряжение. То же относится, естественно, и к любой другой величине, изображающей число. Конечно, в вычислительных машинах с непрерывным способом изображения числа применяются специальные измерительные приборы повышенной точности. Но это требует особых усилий.

§ 2. ПРИНЦИПЫ РАБОТЫ

Математические машины, в которых используется цифровой способ изображения чисел, называют *цифровыми* или *дискретными*. Машины, в которых числа изображаются с помощью непрерывного способа, называют *непрерывными*, *аналоговыми* или *моделирующими*. Различия в способах изображения чисел диктуют, в свою очередь, различия в принципах работы вычислительных машин. Существо этих различий мы и хотим сейчас выяснить.

Цифровые машины вычисляют в известном смысле так же, как это делает человек. Рассмотрим простейшую арифметическую операцию — сложение и простейшее из приспособлений с цифровым способом изображения чисел — русские счеты. Чтобы выполнить на счетах сложение 236 и 142, надо отложить число 236 (рис. 1), а затем на спице, изображающей сотни, опустить вниз одну косточку, на спице, изображающей десятки,— четыре и на спице, изображающей единицы,— две. Короче говоря, дело сводится к сложениям однозначных чисел в каждом разряде отдельно, в результате чего мы и получаем требуемую сумму 378.

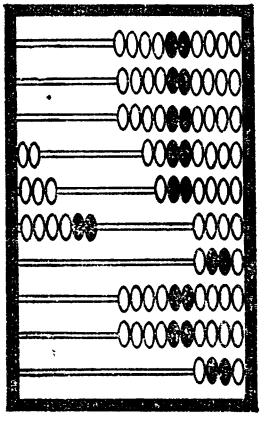


Рис. 1

Несколько иначе придется действовать, если требуется сложить 236 и 147. В разряде единиц мы не в состоянии опустить вниз еще семь косточек — такого количества их просто нет. Поэтому сложение происходит в два приема. Сначала мы опускаем оставшиеся четыре косточки; тогда в этом разряде оказываются опущенными все

десять. Такое состояние элемента эквивалентно нулевому состоянию данного и единице старшего разряда. После того как мы вернем все десять косточек вверх и опустим одну косточку на спице десятков, следует опустить на спице единиц еще три косточки (из семи мы в первый раз опустили лишь четыре). Получим правильную сумму 383.

Таким образом, перенос из разряда в разряд производится на русских счетах вручную. Этот перенос можно несколько ускорить, если заметить, что 7=10—3, т.е. прибавление семерки равносильно вычитанию тройки и прибавлению единицы старшего разряда. Суть операции при этом никак не изменяется.

Аналогично выполняется сложение и в арифмометре, с той только разницей, что в нем перенос в старший разряд выполняется автоматически. Для этого один из зубьев шестеренки, служащей элементом для изображения одного разряда, делается длиннее остальных. При переходе шестеренки из состояния 9 в состояние 0 этот более длинный зуб через посредство вспомогательного механизма поворачивает на один шаг соседнюю шестеренку, изображающую старший разряд (подробнее см. гл. II).

Сказанного, по-видимому, достаточно, чтобы понять основные принципы выполнения арифметических операций в машинах и приспособлениях с цифровым способом изображения чисел: дело сводится к выполнению соответствующих операций над однозначными числами и возможным переносам из разряда в разряд, правильность выполнения которых необходимо обеспечить.

Здесь мы можем снова отметить роль системы счисления, используемой нами для записи числа. Именно система счисления определяет вид «таблицы умножения» и «таблицы сложения» однозначных чисел, которые лежат в основе арифметических операций на цифровых машинах. И снова оказывается, что наиболее простыми, а потому наиболее легко осуществимыми оказываются операции в двоичной системе счисления. Таблица сложения однозначных чисел имеет в ней вид:

$$0+0=0,\ 0+1=1+0=1,\ 1+1=10,$$

а таблица умножения, если ограничиться отличными от нуля множителями, сводится к

$$1\times 1=1.$$

Иначе обстоит дело с выполнением основных операций на вычислительных машинах непрерывного действия. Напомним сначала принципы работы логарифмической линейки.

Как мы уже говорили, на логарифмической линейке число изображается длиной отрезка. При сложении отрезков складываются и их длины. Сложение двух отрезков осуществляется путем приложения начала второго отрезка к концу первого. Поэтому если взять две одинаковые равномерные шкалы, то, сдвигая начальную точку одной из них на величину, равную одному слагаемому, и отклады-

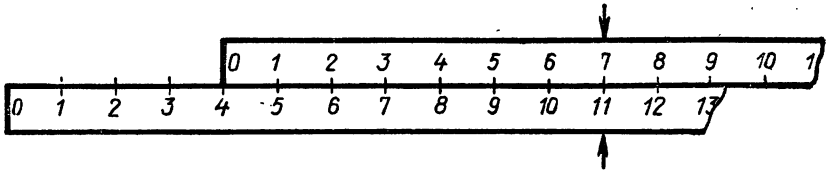


Рис. 2

вая на сдвинутой шкале второе слагаемое, можно на неподвижной шкале найти соответствующую сумму (рис. 2).

Основной шкалой логарифмической линейки является не равномерная, а *логарифмическая шкала*. При ее построении за изображение числа принимается отрезок, длина которого равна логарифму изображаемого числа. Тогда сложению отрезков будет соответствовать уже не сложение чисел, а сложение их логарифмов, что, как известно, соответствует перемножению чисел. Поэтому путем сдвига одной логарифмической шкалы относительно другой можно выполнять умножение и деление (рис. 3).

Если изображать числа с помощью каких-либо электрических величин, то нужно создать электрические схемы, позволяющие выполнять над этими величинами те или иные операции, соответствующие требуемым арифметическим, например складывать токи или вычитать напряжения.

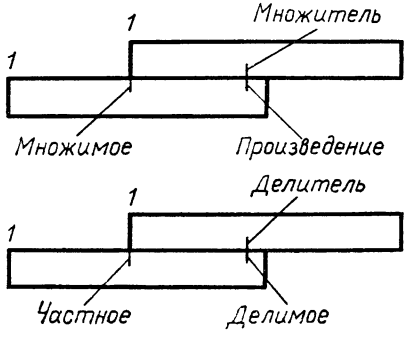


Рис. 3

Легко привести пример схемы, которую можно использовать для умножения и деления и в которой числа изображаются электрическими величинами. Возьмем электрическую цепь, состоящую из источника, например гальванической батареи, и резистора. Известно, что ток в такой цепи подчиняется закону Ома

$$I = \frac{E}{R},$$

где I — ток, E — напряжение и R — сопротивление. Включив в цепь амперметр и вольтметр (рис. 4) и зная величину R, мы можем по показаниям амперметра находить частное от деления E на R, а по показаниям вольтметра — произведение IR, придавая остальным величинам

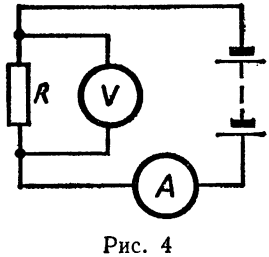


Рис. 4

нужные значения. Таким образом, эта простейшая электрическая схема может быть использована для умножения и деления. Числа в ней изображаются электрическими величинами — напряжением, величиной тока и сопротивлением.

Рассмотренный пример позволяет понять, почему математические машины непрерывного действия называют *моделирующими машинами*. Машины непрерывного действия воспроизводят (*моделируют*) процесс, который описывается данной функцией, уравнением или системой уравнений. Такое моделирование называют *математическим моделированием*, в отличие от физического моделирования, в котором изменяются лишь масштабы, но не физическая природа моделируемого процесса или явления.

Мы не будем подробнее рассматривать эти вопросы, а ограничимся в этой книге только цифровыми машинами.

Глава II

НЕМНОГО ИСТОРИИ

§ 1. НАЧАЛА ИНСТРУМЕНТАЛЬНОГО СЧЕТА

«Числа правят миром»,— говорил Платон. Эти слова особенно справедливы в наше время, когда *вычислительные машины* и *численные методы* успешно применяются практически во всех областях человеческого знания.

Кто и когда изобрел числа? Хотя Прометей в трагедии «Прикованный Прометей» древнегреческого драматурга Эсхила (525—456 гг. до н. э.) утверждает:

> Подумайте, что смертным сделал я:
> Число им изобрел
> И буквы научил соединять...,

понятие ч и с л а возникло задолго до появления письменности. Люди научились считать, в течение сотен веков передавая свой опыт и свои знания из поколения в поколение.

Счет, или более широко *вычисления*, могут быть выполнены в различной форме: существует *устный*, *письменный* и *инструментальный счет (вычисления)*. Средства для инструментального счета в разные времена называли по-разному: счетные доски, абаки, счеты, счетные инструменты, снаряды, приспособления, приборы, машины и, наконец, примерно с середины нашего столетия — вычислительные машины. Им будут посвящены последующие главы этой книги.

В этой главе мы рассмотрим краткую историю развития *средств инструментальных вычислений*; как говорил Г. В. Лейбниц, «Кто хочет ограничиться настоящим без знания прошлого, тот никогда не поймет его».

Древнейшим «счетным инструментом», который сама природа предоставила в распоряжение человека, была его собственная рука. «Понятие числа и фигуры,— пишет Ф. Энгельс,— взято не откуда-нибудь, а только из действительного мира. Десять пальцев, на которых люди учились считать (производить первую арифметическую операцию), представляют собой все, что угодно, только не

продукт свободного творческого разума»*. Имена числительные во многих языках указывают, что у первобытного человека пальцы являются преимущественно *орудием счета*. Кисть руки, *пясть*, является синонимом и фактической основой числительного «пять» у многих народов.

По словам знаменитого русского путешественника Н. Н. Миклухо-Маклая (1846—1888), туземцы Новой Гвинеи считали следующим образом: «. . . папуас загибает один за другим пальцы руки, причем издает определенный звук, например «бе, бе, бе . . .» Досчитав до пяти, он говорит «ибон-бе» (рука). Затем он загибает пальцы другой руки, снова повторяет «бе, бе . . .», пока не доходит до «ибон-али» (две руки). Затем он идет дальше, приговаривая «бе, бе. . .», пока не доходит до «самба-бе» и «самба-али» (одна нога, две ноги)» **.

От пальцевого счета берут начало пятеричная система счисления (одна рука), десятичная (две руки), двадцатеричная (пальцы рук и ног). В гомеровской «Одиссее» часто встречается слово «пятерить», имеющее по смыслу значение «считать» и свидетельствующее о распространении в гомеровскую эпоху пальцевого счета. В другом литературном памятнике — комедии Аристофана «Осы» (конец V — начало IV века до н. э.) говорится: «Подсчитай попросту, не на камешках, а на руках, все подати. . .»

Хорошо был известен пальцевый счет и в Риме. По свидетельству древнеримского историка Плиния-старшего (погибшего в 79 году в Помпее во время извержения Везувия), на главной римской площади — Форуме — была воздвигнута гигантская фигура двуликого бога Януса. Пальцами правой руки он изображал число 300, пальцами левой — 55. Вместе это составляло 355, т. е. число дней в году в соответствии с римским календарем.

Полное описание пальцевого счета составил в средневековой Европе ирландский монах Бе́да Достопочтенный (ок. 673—735 гг.). Среди его трудов есть трактат «О счислении». В нем подробно излагаются способы счета на пальцах, причем этот счет распространялся на все числа вплоть до миллиона. Трактат Беды явился источником, откуда средневековые составители учебников арифметики в течение многих лет черпали свои сведения о пальцевом счете. Наряду со способами простого *подсчета на пальцах* отдельных предметов в этих учебниках излагаются сведения о выполнении *арифметических операций* с помощью пальцев рук.

Издревле употреблялся еще один вид инструментального счета — с помощью деревянных палочек (бирок) с зарубками. Впервые упоминание о способе записи чисел путем нанесения зарубок встречается на барельефе храма фараона Сети I (1350 г. до н. э.) в Абидосе. Здесь изображен бог Тот, отмечающий с помощью зарубок на пальмовой ветви длительность срока правления фараона. В сред-

---

* Энгельс Ф. Анти-Дюринг. М., 1967, с. 33.

** Миклухо-Маклай Н. Путешествия. М.—Л., 1940, т. 1, с. 280.

ние века в России, как и в остальной Европе, бирками пользовались для учета и сбора налога. Бирка разрезалась на две продольные части, одна из которых оставалась у крестьянина, другая —у сборщика налогов. По зарубкам на обеих частях и велся счет уплаты налога, который проверяли складыванием частей бирки. В Англии, например, этот способ записи налогов существовал до конца XVII столетия.

Другие народы (китайцы, индийцы, американские индейцы) использовали для представления чисел веревки с узелками.

§ 2. АБАК, СУАН-ПАН, СЧЕТЫ...

Раннему развитию письменного счета препятствовали два обстоятельства. Во-первых, в первоначальных системах счисления письменные вычисления были сложными. Чтобы убедиться в этом, советуем читателю перемножить $CLVI \times LXXIV$, пользуясь римской системой счисления. Во-вторых, не было материала, на котором удобно производить письменные вычисления: пергамент был изобретен лишь в V веке до н. э., бумага — в начале нашей эры.

Этими обстоятельствами можно объяснить появление специального счетного прибора, известного в древности под названием *абака*. Происхождение термина «абак» не установлено. Большинство историков производят его от семитического корня; согласно этому толкованию абак означает дощечку, покрытую слоем пыли. В своей примитивной форме абак действительно представлял собой такую дощечку. На ней острой палочкой проводили линии, и в получившихся колонках по позиционному принципу размещали какие-нибудь предметы, например камешки или палочки.

Впоследствии абак претерпел ряд изменений. В результате замены камешков косточками или шариками, нанизанными на нити или прутья, получили счеты, суан-пан, соробан; в других странах

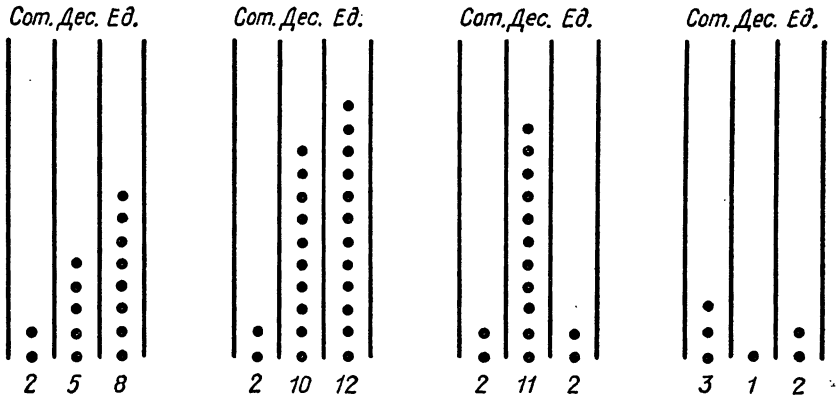


Рис. 5

стали использовать разлинованную таблицу со счетными жетонами (счет на линиях); все это — разновидности абака.

Посмотрим, как выполняются арифметические операции на абаке при десятичной системе счисления. На рисунке 5, не требующем комментариев, показана последовательность выполнения сложения 258+54. Вычитание выполняется изъятием камешков, умножение — как повторное сложение, деление — как повторное вычитание.

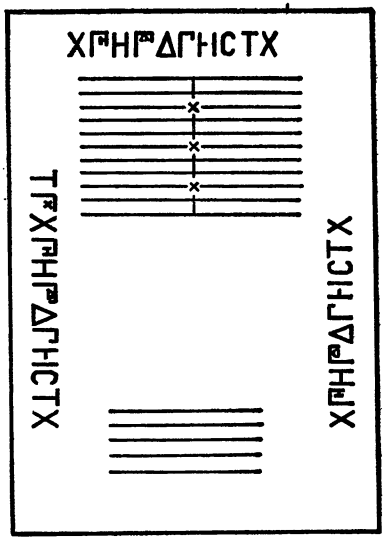


Рис. 6

Известно свидетельство историка Геродота (V век до н. э.), что египтяне пользуются абаком, причем в отличие от греков передвигают камешки не слева направо, а сверху вниз. Отсюда видно, что даже в эпоху Геродота и в Греции, и в Египте абак получил широкое распространение. Историки полагают, что в Грецию абак был завезен финикийцами и стал там «походным инструментом» греческого купца. О коммерческом назначении абака свидетельствует то обстоятельство, что значения, приписываемые камешкам в различных колонках, обычно сообразовывались с отношениями различных денежных единиц. Например, у историка Полибия мы встречаем слова: «Придворные — как камни на счетной доске; захочет счетчик, и они будут стоить один халк, а захочет — так и целый талант» (один талант равнялся 6000 драхмам, а халк составлял 1/48 драхмы). На рисунке 6 приведен древнегреческий абак, найденный в середине прошлого века на острове Саламине. Он представляет собой мраморную доску размерами $1{,}5 \times 0{,}75$ *м*, на которой расположено десять «длинных» колонок (для целых чисел) и четыре «коротких» (для дробей). Буквы, стоящие сверху, справа и слева на рисунке, раскрывают способ пользования абаком.

В Древнем Риме абак назывался *calculi* или *abaculi*. Римский абак изготовляли из камня, бронзы, слоновой кости или цветного стекла. Слово *calculus* означает «галька», «голыш». От этого слова произошло в дальнейшем латинское *calculatore (вычислять)* и *calculus (исчисление)* и наше — *калькуляция*. Сохранился бронзовый римский абак, на котором *calculi* (множ. от *calculus*) передвигались в вертикально прорезанных желобках. Внизу помещали камешки для счета до пяти, а в верхней части имелось отделение для камешка, соответствующего пятерке (рис. 7).

В китайской книге VI века нашей эры описан прибор, состоящий из укрепленных на доске параллельных веревок, на каждой из ко-

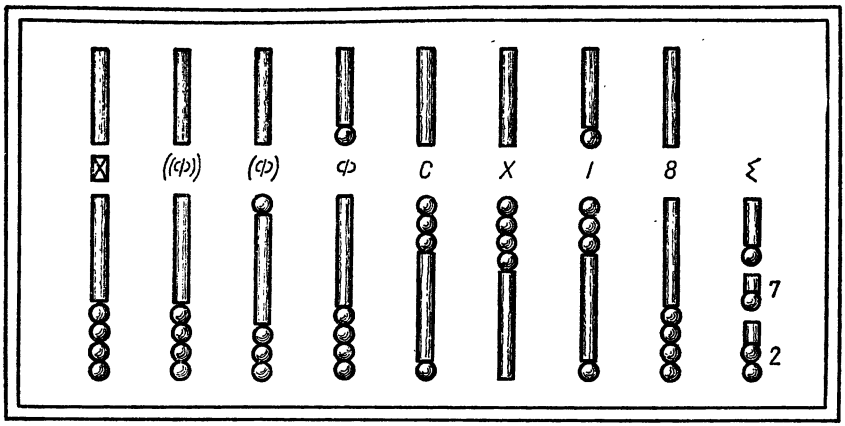


Рис. 7

торых надето 5 косточек, причем последняя имеет другой цвет и означает 5 единиц — китайская разновидность абака, *суан-пан*. Современный тип суан-пана появился не раньше XII столетия (рис. 8, *а*). Две части, на которые он делится перегородкой, носят названия «земля» и «небо». Шарик «земли» означает единицу, а шарик «неба» — 5 единиц.

Японский *соробан* (рис. 8, *б*) происходит от китайского суан-пана, который был завезен в Японию в XV—XVI веках. Соробан проще своего предшественника — на каждой проволочке и в каждом отделении у него на один шарик меньше, чем у суан-пана.

Русские счеты в своей первоначальной форме появились примерно на рубеже XVI—XVII веков. О них мы уже говорили в гл. I (рис. 8, *в*).

Описанные инструменты являются национальными и широко используются в своих странах до сих пор. Опытные счетоводы работают на этих инструментах со скоростью, не уступающей скорости вычислений на настольных счетных машинах.

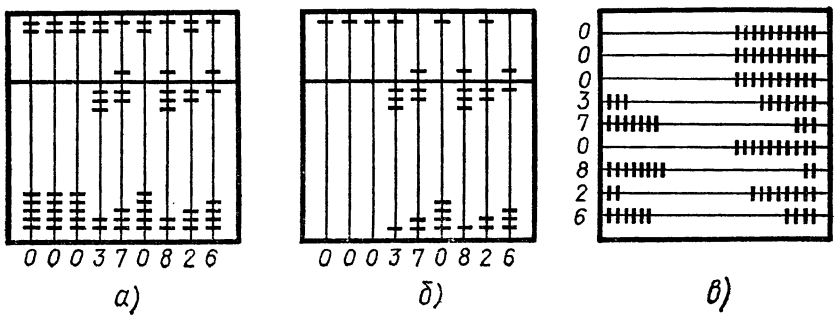


Рис. 8

В средневековой Европе распространение получили два типа абака. Один из них описал в своей книге французский ученый монах Герберт, немало сделавший для развития науки в мрачный период средневековья.

Герберт родился около 945 года в Оверни (центральная Франция) в бедной семье и воспитывался в монастыре городка Орильяка. Стремясь к расширению своих знаний, Герберт едет за Пиренеи, в христианскую часть Испании. Поселившись у епископа города Вих, он много учится, особенно математике. В 970 году в качестве секретаря епископа Герберт отправляется в Рим, где, будучи представленным папе и императору Оттону I, остается на некоторое время учителем сына императора. В 991 году Герберт избирается архиепископом Реймса, но через 6 лет оставляет этот пост и становится учителем 17-летнего императора Оттона III, сына Оттона II. В 999 году Герберт стал римским папою и в 1003 году умер.

Герберт занимался астрономией, логикой, философией и геометрией; он является автором нескольких математических сочинений, одно из которых «Правила вычисления с помощью абака», теологических трактатов и публицистических посланий.

В описании Герберта абак представлял собой гладкую доску, посыпанную голубым песком и разделенную на 30 столбцов, из которых три отводились для дробей, а прочие группировались по три столбца в девять групп. Сверху столбцы и группы завершались дугами (рис. 9). Столбцы в каждой группе обозначались (слева направо) буквами *C* (centum, 100), D (decem, 10) и S (singularis, 1) соответственно. В отличие от древних форм счетной доски в каждый столбец клали не камешки, а особые нумерованные жетоны (сделанные из рога) — «апексы», на которых были обозначены девять первых числовых знаков. Апекс с изображением нуля отсутствовал, поэтому для изображения нуля в соответствующий столбец жетона не клали. Таким образом, 27-разрядное целое число на абаке представлялось как бы сгруппированным по три разряда.

Замена камешков апексами не давала больших преимуществ для вычислений. Апексы имели иное значение в развитии математики: в них можно видеть ближайших предков современных европейских цифр. Значение книги Герберта состоит не только и не столько в том, что он начертил 30-колонный абак, а в том, что он дал правила вычисления на абаке, которые несколько веков «питали» европейскую арифметическую культуру.

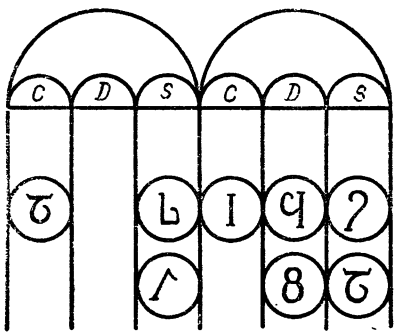


Рис. 9

Другой тип абака получил распространение с конца XV столетия. Он представляет собой горизонтальную разлинованную таблицу, на которой выкладываются спе-

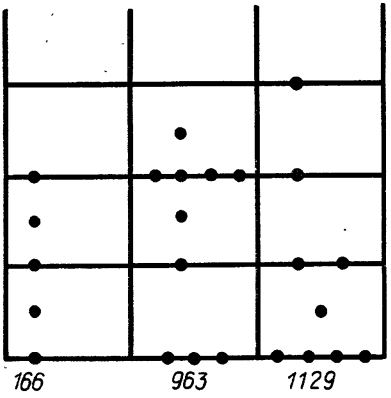


Рис. 10

циальные жетоны. Горизонтальные линии таблицы соответствуют единицам, десяткам, сотням и т. д. На каждую линию кладут до четырех жетонов; жетон, помещенный между двумя линиями, означал пять единиц разряда ближайшей нижней. В вертикальном направлении доска расчерчивалась на несколько столбцов для отдельных слагаемых или множителей. На рисунке 10 показана схема сложения 166+963= =1129.

Счетные таблицы употреблялись до конца XVIII века. В Нюрнберге, например, в XVI—XVII веках изготовлением металлических счетных жетонов занималась целая отрасль промышленности. Счетные таблицы отличались большим разнообразием, начиная от специальных столов и кончая платками и ковриками.

Они были необходимой принадлежностью купца и чиновника, ученого и школяра. Счет на линиях вспоминают герои Шекспира; Мольер в «Мнимом больном» смешил Париж, заставляя больного проверять счета аптекаря, раскладывая на столе жетоны; Лейбниц предпочитал способ счета жетонами арифметическим расчетам на бумаге.

Профессор А. П. Юшкевич приводит в своей книге * обычно приписываемое прусскому королю Фридриху II стихотворение:

Придворные — точь-в-точь жетоны,
все их значенье — в положеньи.
В фаворе значат миллионы,
но лишь нули — в пренебреженьи.

По существу это лишь стихотворный пересказ слов Полибия, цитированных в начале параграфа.

Цифры на апексах были первым появлением арабско-индийских цифр в Европе. Первой книгой по арифметике, в которой эти цифры употреблялись уже систематически, была «Liber abaci» Леонардо Фибоначчи из Пизы, впервые изданная в 1202 году. Начиная с нее, новые цифры постепенно получали все более широкое распространение, проникая в счетоводство торговых домов. В XV веке разгорелась борьба между «абацистами» — сторонниками счета на абаке и «алгоритмиками» — сторонниками счета на линиях, который под воздействием использования арабско-индийских цифр быстро стал развиваться в сторону современных

* Юшкевич А. П. История математики в средние века. М., 1961, с. 360.

алгоритмов письменного счета. Не удивительно поэтому, что широкое распространение бумаги, изобретенной в XII—XIII веках, сделало ненужным абак и обеспечило победу «алгоритмиков», хотя борьба продолжалась еще несколько веков.

§ 3. ДЖОН НЕПЕР И ЕГО ПАЛОЧКИ

Начало XVII столетия знаменуется двумя замечательными достижениями инструментального счета. Оба они связаны, одно непосредственно, а другое косвенно, с именем великого шотландского ученого Джона Непера и направлены на то, чтобы упростить выполнение операций умножения и деления. Насколько эти операции представлялись затруднительными для людей XVI—XVII столетий, свидетельствует запись в дневнике крупного чиновника Британского Адмиралтейства Сэмюэля Пеписа, впоследствии друга Ньютона, сделанная им 4 июля 1662 года: «К пяти часам утра, приведя в порядок свой журнал, я отправляюсь в контору. Вскоре туда приходит мистер Купер, с помощью которого я надеюсь изучить математику. . . (я пытаюсь, прежде всего, выучить таблицу умножения)».

Понятно, какое значение имело изобретение логарифмов. Великий физик, математик и астроном И. Кеплер писал тюбингенскому профессору математики Вильгельму Шиккарду: «. . . Некий шотландский барон, имени которого я не запомнил, выступил с блестящим достижением; он каждую задачу на умножение и деление превращает в чистое сложение и вычитание . . .» Этим шотландским бароном был Джон Непер, опубликовавший в 1614 году свой знаменитый трактат «Mirifici logarithmorum canonis descriptio» («Описание удивительных таблиц логарифмов»).

Джон Непер родился в 1550 году в фамильном замке Мерчистон, близ Эдинбурга, столицы Шотландии. В юности он отличался нелюдимым и застенчивым характером и не слишком крепким здоровьем. О воспитании мальчика больше всего заботился его дядя — епископ Адам Босуэлл. 20 декабря 1563 года Джон поступает в один из колледжей университета Сент-Эндрю. Около 1566 года, после смерти матери, он оказывается в Европе, где с целью совершенствования образования посещает Италию, Данию, Францию. В 1571 году он возвращается в Мерчистон и в 1572 году женится на Элизабет Стирлинг. В 1574 году Непер с женою переезжает в замок в Гартнесе, где продолжает свои научные занятия, начатые еще в университете.

Умер Джон Непер в Мерчистоне 4 апреля 1617 года. По словам английского историка, он «заслуживает звания Великого Человека, более чем любой другой шотландец, когда-либо появившийся на свет».

Дж. Непер

Непер занимался наукой исключительно ради удовлетворения прирожденной жажды знаний и неохотно отдавал свои труды в распоряжение печатного станка. Поэтому знаменитый трактат о логарифмах был напечатан только в 1614 году, хотя по многим свидетельствам Непер пришел к идее логарифма на 20 лет раньше, а первое математическое сочинение Непера «De arte logistica», посвященное некоторым вопросам тригонометрии, алгебры и арифметики, вышло в свет лишь в 1839 году.

В 1617 году, незадолго до смерти Непера, увидело свет его последнее сочинение «Rabdologie seu numerationis, per virgulas labri duo». В предисловии к нему Непер писал: «Я всегда старался, насколько позволяли мои силы и способности, избавиться от трудности и скуки вычислений, докучливость которых обыкновенно отпугивает очень многих от изучения математики». Значение термина rabdologie Непер объясняет как *счет с помощью палочек*. Эти палочки под названием «палочки Непера», как и сам метод, быстро получили широкое распространение в Европе и были одно время даже более популярны, чем логарифмы — главное изобретение Непера.

Непер, вероятно, был знаком с приемом умножения, описанным в известном средневековом трактате Луки Пачоли «Сумма де арифметика» под названием gelosia (этот прием был знаком индийцам задолго до Пачоли). Суть его в следующем. Счетную доску (или просто лист чистой бумаги) расчерчивали в виде сетки прямоугольников, разделенных диагоналями. По сторонам сетки (сверху и справа) записывали сомножители, а промежуточные произведения помещали в прямоугольники так, чтобы диагональ разделяла «единицы» и «десятки» (обычно, «десятки» помещались в верхний треугольник, а «единицы» — в нижний). Для получения произведения осуществляли суммирование «вдоль диагоналей», а результат записывали снизу сетки (младшие разряды) и слева от сетки (старшие разряды).

В качестве примера на рисунке 11 показано выполнение методом gelosia умножения $954 \times 314 = 299\,556$. Цифры шести разрядов произведения 954×314 получаются следующим образом;

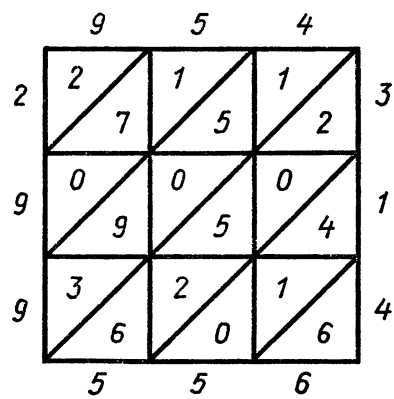


Рис. 11

2; (0+7+1); (3+9+0+5+1); (6+2+5+0+2); (0+1+4); 6

или 2, 8, (18), (15), 5, 6, т. е. 299 556.

По мнению Пачоли, запись выкладок при этом методе напоминает решетчатые оконные ставни, скрывавшие от взоров прохожих сидящих у окон женщин. Такие ставни называли gelosia *(жалюзи)* в связи с другим значением этого слова — ревность.

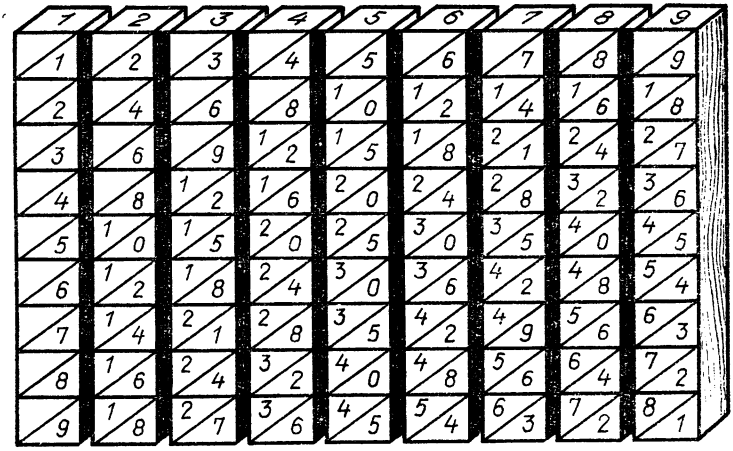


Рис. 12

Непер предложил разрезать «школьную» таблицу умножения* на 9 полосок и разделить числа диагональю на единицы и десятки, как в gelosia. Полоски наклеивались на дощечки (рис. 12), по одной палочке для каждой цифры, и изготовлялись в нескольких экземплярах.

Для умножения, например, 327×8 прикладываем друг к другу палочки для цифр 3, 2 и 7 и после этого обращаемся к восьмой строке. Слева прикладывалась еще одна «единичная» палочка, против цифры 8 на этой палочке мы имеем произведение на 8 цифр 3, 2, 7, из которых составлено множимое. Суммируя числа, как в gelosia, получаем:

$$2;\ (4+1);\ (6+5);\ 6$$
$$\text{или } 2,\ 5,\ (11),\ 6,\ \text{т.е. } 2616.$$

Если множитель многозначный, то отдельные произведения выписывают, как обычно, со смещением на один разряд и затем складывают.

В XVI и XVII столетиях в Европе появилось множество модификаций палочек Непера. В 1668 году вюртембергский иезуит Каспар Шотт предложил заменить палочки Непера цилиндрами, на поверхности которых вдоль образующих нанесены числа с палочек Непера. Цилиндры помещались параллельно друг другу в ящичке, где могли вращаться на проходящих через них осях**. В 1678 году Пьер Пти, французский математик и физик, друг Паскаля, наклеил полоски бумаги с начерченными «палочками» на картонные ленты

---

* Во времена Непера ее ошибочно называли пифагоровой.

** Еще раньше эту же идею использовал в своей механической вычислительной машине уже упоминавшийся нами В. Шиккард (см. следующий параграф).

и заставил их двигаться вдоль оси цилиндра. Устройство получило название барабана Пти. В 1727 году замечательный немецкий механик и энциклопедист Якоб Лейпольд видоизменил барабан Пти, придав ему прямоугольную форму. Год спустя Михаил Фортиус предложил свой прибор, состоявший из ряда подвижных концентрических кругов все с теми же надписями.

(Этот список можно было бы продолжить, но мы упомянем одну из последних переработок. Это «Настольная таблица умножения для сложных вычислений, по Неперу переработал Б. Ф. Тихомиров», изданная в Ленинграде в 1930 году.)

Изобретение логарифмов, которое, по словам Лапласа, «сократив труды астронома, удвоило его жизнь», послужило основой для изобретения замечательного вычислительного инструмента, вот уже 350 лет служащего инженерам всего мира,— *логарифмической линейки*.

§ 4. МЕХАНИЧЕСКАЯ ЭРА

Первая механическая счетная машина была изготовлена в 1623 году; незначительный промышленный выпуск таких машин стал возможным лишь спустя 200 лет. Настоящее распространение и использование механических счетных машин началось лишь в 1880—1890 годах.

Долгое время считалось, что первая механическая *суммирующая* машина была изготовлена в 1642 году Блезом Паскалем (1623—1662), в последующем великим математиком, физиком и философом. Но в 1957 году в библиотеке г. Штуттгарта (ФРГ) среди записей и документов великого математика и астронома Иоганна Кеплера был обнаружен чертеж неизвестной ранее счетной машины. Как удалось установить, это был чертеж машины, сконструированной и изготовленной другом Кеплера, профессором университета в Тюбингене, Вильгельмом Шиккардом.

Вильгельм Шиккард (1592—1636) начал работать в Тюбингене в 1619 году в качестве профессора кафедры восточных языков. Но он интересовался астрономией и, вступив в переписку с Кеплером, все более увлекался точными науками. Уже в 1631 году Шиккард возглавил университетскую кафедру математики и астрономии. Первая счетная машина Шиккарда была построена в середине 1623 года, но сгорела во время пожара в начале 1624 года. В 1636 году В. Шиккард и вся его семья умерли от чумы, и его работы, как и его имя, были забыты вплоть до находки 1957 года. Поэтому машина Шиккарда не оказала влияния на развитие вычислительной техники, хотя многие его идеи и конструктивные решения были позднее переоткрыты и использованы в последующих машинах.

Но и машина Шиккарда оказалась не первой. В 1967 году в Национальной библиотеке в Мадриде были обнаружены два тома неопубликованных рукописей Леонардо да Винчи (1452—1519). Среди чертежей I тома (Codex Madrid I), почти полностью посвященного прикладной механике, имелся эскиз тринадцатиразрядного суммирующего устройства с десятизубыми колесами. По этому

эскизу машина была построена и оказалась работоспособной. Однако ведь и машина Леонардо да Винчи оставалась неизвестной изобретателям вычислительных машин XVII—XVIII веков. Поэтому следует считать правильным, что «биографии» механических вычислительных машин ведутся от Паскаля.

Паскаль

Счетная машина Паскаля была задумана им в 1640 году; семнадцатилетний юноша очень хотел облегчить работу своему отцу, который просиживал дни и ночи над однообразными и утомительными расчетами, будучи интендантом Руана. Первая модель машины была готова через несколько месяцев, но оказалась неудачной. Паскаль сделал около пятидесяти различных моделей, экспериментируя с материалом и формой элементов машины. Исследовались модели из дерева, слоновой кости, меди и других материалов, в одних были прямолинейные стержни и плоские пластинки, в других — передаточные цепи; применялось прямолинейное и круговое движение, конические и цилиндрические зацепления и т. д.

Работа над счетной машиной продолжалась около пяти лет и была завершена в 1645 году. В 1649 году Паскаль получил «королевскую привилегию» (патент), дающую право на изготовление и продажу машины. Некоторое количество таких машин действительно было им изготовлено, и часть из них была продана. Машина Паскаля со снятой крышкой изображена на рисунке 13.

Числа в счетной машине Паскаля изображались с помощью *счетных колес*, в принципе таких же точно, как и в машинах Шиккарда

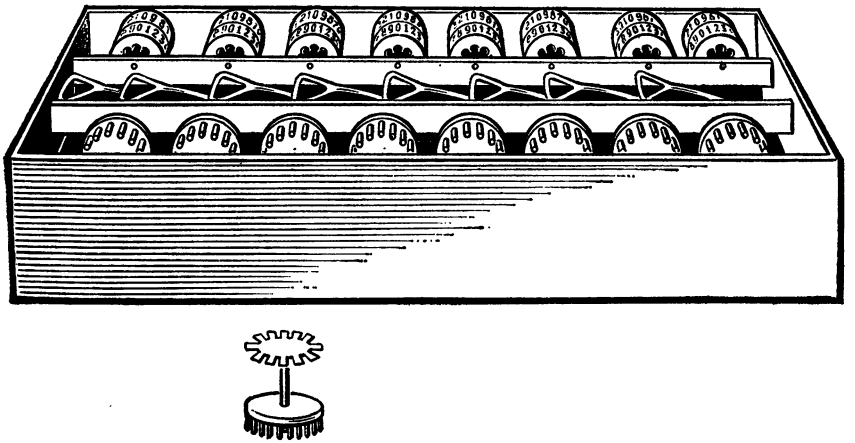


Рис. 13

и Леонардо да Винчи, хотя эти машины наверняка не были известны Паскалю. Десятичное счетное колесо представляло собой шестеренку с десятью зубцами, расположенными через 36° по окружности. Для установки цифры в каждом разряде между соответствующими зубцами установочного колеса вставлялся штифт, после чего колесо поворачивалось до неподвижного упора. Рядом с каждым зубцом гравировалась цифра, которая после поворота колеса была видна в окошко лицевой крышки машины. Таким образом, цифры изображались углами поворота вала, которые могли принимать значения, кратные 36°. При установке следующего слагаемого каждое счетное колесо поворачивалось на дополнительный угол в соответствии с новой цифрой в каждом разряде, в результате чего осуществлялось сложение в каждом разряде.

Интересной частью машины Паскаля является *механизм переноса*, обеспечивающий перенос из младшего разряда в старший. Он поворачивает на 1 зуб колесо старшего разряда при полном обороте колеса младшего. На колесе *1* (рис. 14) имеются стержни *3*, которые при вращении колеса входят в зацепление с зубьями вилки *4*, свободно вращающейся на оси колеса *2* старшего разряда. Вилка несет на себе защелку *5*; в тот момент, когда колесо *1* перейдет от 9 к 0, вилка выскользнет из зацепления и упадет вниз, увлекая защелку. Последняя натолкнется на зуб колеса *2* и повернет его на 1/10 оборота.

В июне 1652 года Паскаль отправляет экземпляр своей машины в Стокгольм шведской королеве Христине. В коллекции шведской

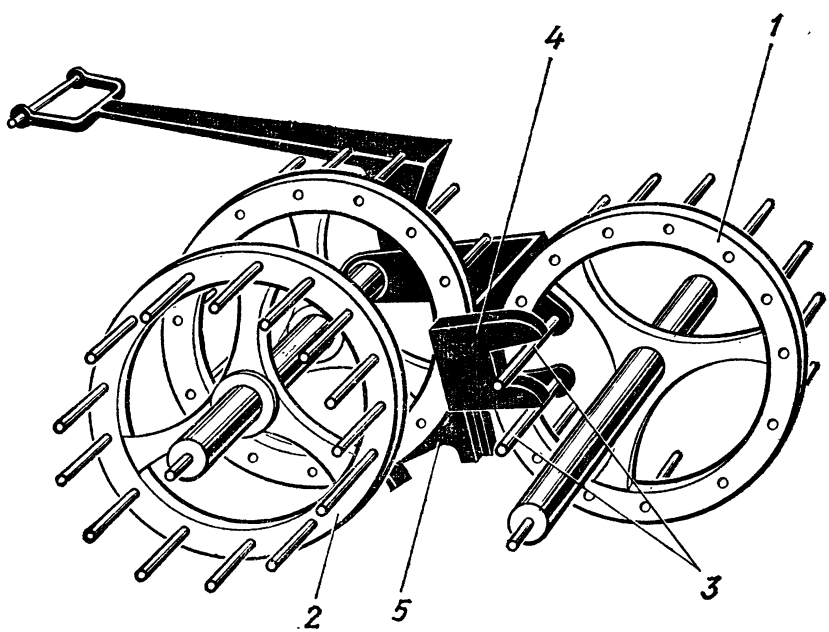


Рис. 14

королевы эту машину мог видеть член дипломатической миссии республиканской Англии, магистр Сэмюэль Морлэнд (1625—1695), который впоследствии сконструировал первую английскую счетную машину. В этой машине, изготовленной в 1666 году, отсутствовал автоматический перенос из разряда в разряд, но зато для каждого колеса существовал счетчик числа полных оборотов. Показания этого счетчика прибавлялись к показаниям старшего разряда.

Упомянутые выше первые счетные машины подвергались дальнейшим усовершенствованиям. Из числа таких усовершенствованных счетных машин можно отметить машину часовщика Людовика XIV Рене Грийе (1678), профессора математики и метеорологии из Гессена Христиана-Людовика Герстена (1722), Л'Епина (1725) и де Буистиссандо (1730).

Новое и несколько необычное для XVIII века применение для счетной машины нашел Хакоб Родриг Перейра (1715—1780). Выходец из Португалии, Перейра получил в Бордо медицинское образование и посвятил свою жизнь обучению глухонемых. Он разработал «графический алфавит» и методику обучения глухонемых чтению и речи. В 1751 году он сконструировал и изготовил счетную машину, назначением которой было обучение глухонемых счету.

От известных к тому времени суммирующих машин она отличалась, прежде всего, тем, что счетные колеса располагались на единственной оси, проходящей через все колеса, что делало машину более компактной. Кроме того, эта машина имела весьма оригинальный механизм переноса десятков.

На нынешней территории Советского Союза первая суммирующая машина была, как гласит сделанная на ней надпись, «изобретена и изготовлена» около 1770 года в Белоруссии «Евной Якобсоном, часовым мастером и механиком в городе Несвиже». Машина Якобсона была девятиразрядной. Как и в других суммирующих машинах, вычитание здесь производилось путем поворота счетных колес в противоположную сторону, а умножение — путем повторного сложения.

Суммирующие счетные машины, как описанные выше, так и не упомянутые нами, изготовлялись в нескольких экземплярах и не получили в XVII—XVIII веках сколько-нибудь серьезного употребления. Во-первых, они не были еще по-настоящему необходимы. Во-вторых, они были капризны, ненадежны и неудобны в эксплуатации. Это происходило не столько из-за недостатков конструкции, сколько вследствие низкого уровня технологических процессов и невысокой вследствие этого точности изготовления и сборки.

Коренное изменение произошло в XIX столетии, когда рост промышленности и транспорта и расширение коммерческой деятельности банков сделали построение быстродействующих и надежных счетных машин актуальной задачей. На патентные бюро, особенно в США, обрушился шквал заявок на такие изобретения. Однако по-настоящему удачная конструкция многоразрядной клавишной суммирующей машины была предложена лишь в 1885 году. Это сделал 24-летний механик Дорр Э. Фелт, назвавший свою машину комптометром.

Организовав в 1887 году вместе с чикагским предпринимателем Р. Таррантом компанию по производству комптометров, Фелт создал первую, практически использовавшуюся в США счетную ма-

шину. Ее преимуществами является совмещение операций установки и счета и клавишный привод для установки чисел. Вместе с тем существенным недостатком машины Фелта является невозможность контролировать правильность ввода чисел и отсутствие печатающего механизма. Последний недостаток Фелт пытался исправить. Одну из своих конструкций он снабдил печатающим механизмом. Но эта конструкция оказалась не слишком удачной, и слава создателя первой успешной печатающей суммирующей машины принадлежит его соотечественнику Уильяму С. Бэрроузу.

Бэрроуз родился в 1857 году в семье неудачливого механика и еще мальчиком был отдан учеником бухгалтера в один из провинциальных банков. Тяжелые условия жизни привели к заболеванию туберкулезом, в результате чего ему пришлось оставить бухгалтерию. В 1882 году, работая механиком на фабрике в Сен-Луисе, он начинает работать над «бухгалтерской машиной». В конце 1885 года ему удалось создать машину, которая печатала вводимые числа, суммировала их и затем печатала результат.

Фирма, созданная для производства «бухгалтерских машин», принесла Бэрроузу богатство и славу, которыми он не успел воспользоваться, скончавшись в 1898 году от туберкулеза.

Элемент счетного механизма машины Бэрроуза показан на рисунке 15. При наборе числа нажатые клавиши западают; это дает возможность по зажатым клавишам проверить правильность набора. Для осуществления сложения необходимо повернуть приводной рычаг. Эти машины выпускались свыше 60 лет и в последних моделях приводной рычаг был заменен электромотором.

Использование суммирующих машин для умножения возможно, но неудобно и затруднительно. Для получения частных произведе-

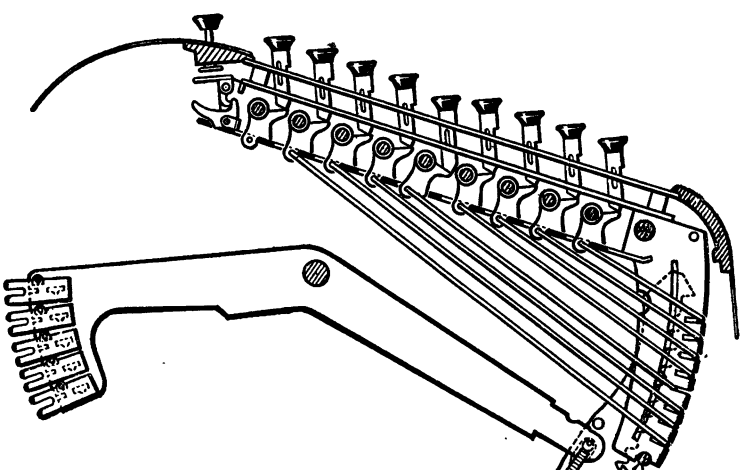


Рис. 15

ний приходится многократно вводить множимое в машину. При этом легко ошибиться либо при вводе множимого, либо при счете числа введенных слагаемых. Полученное частное произведение надо зафиксировать отдельно, очистив машину для получения следующего. Наконец, после получения всех частных произведений их надо вводить в машину заново со сдвигом для окончательного сложения. Создание машин, приспособленных для выполнения умножения или всех четырех арифметических действий (арифмометров), требует преодоления всех указанных недостатков.

Лейбниц

Арифмометр, приспособленный для выполнения всех арифметических действий, был впервые предложен в 1670 году великим немецким ученым Готфридом Вильгельмом Лейбницем (1646—1716). Имя Лейбница и его работы в области математики, механики, философии, логики, права и других наук достаточно хорошо известны, поэтому мы не станем останавливаться ни на его научных заслугах, ни на сведениях из его биографии и ограничимся лишь его работами в области вычислительной техники.

Лейбниц предложил использовать вместо установочного колеса ступенчатый валик — цилиндр, вдоль которого имеется десять выступов различной длины. Устанавливая десятичное счетное колесо в различных местах валика, мы можем, поворачивая валик на 360°, поворачивать колесо на нужное, всегда одно и то же число зубьев. Первые варианты машины Лейбница были мало удачными, но Лейбниц неоднократно возвращался к своей машине, работая над ее усовершенствованием около 40 лет: описание последнего варианта машины было опубликовано им в 1710 году.

Благодаря использованию ступенчатого валика, установка множимого выполняется в машине Лейбница лишь один раз путем передвижения счетного колеса вдоль валика так, чтобы обеспечить зацепление с нужным числом зубцов. Передвижение каретки обеспечивало требуемый сдвиг частичных произведений. Таким образом, арифмометр Лейбница содержал уже почти все принципы работы позднейших механических арифмометров. Несмотря на это, он не получил широкого распространения вследствие высокой стоимости, чрезвычайных трудностей изготовления и больших требований к точности, которые трудно было удовлетворить в XVII веке. Но ступенчатый валик, как важнейший элемент для представления числа, сохранился в механических арифмометрах вплоть до настоящего времени, благодаря удобству эксплуатации и технологичности изготовления.

Разновидности ступенчатого валика использовались в XVIII столетии в арифмометрах Стэнхоупа, Гана и Мюллера.

Чарльз Стэнхоуп (1753—1816) происходил из старинной английской семьи. Он увлекался политической деятельностью и научными исследованиями; уже в 1770 году получил награду Шведской Академии наук за статью о маятнике. В 1775, 1777 и 1780 годах он изобретает счетные машины, из которых последняя является усовершенствованием суммирующей машины Морлэнда, а две первые предназначены для выполнения всех четырех действий. В них использовались модифицированные ступенчатые валики Лейбница.

Приблизительно в те же годы изобрел свою машину швабский пастор, астроном-любитель и механик Филипп-Матеус Ган. Столкнувшись с необходимостью сложных и громоздких вычислений при расчете астрономических приборов и часов, он вспомнил описание арифмометра Лейбница и поставил перед собой цель создать аналогичную машину. Ган изготовил удачную и компактную конструкцию, расположив ступенчатые валики вертикально. Эта конструция была усовершенствована в 1784 году военным инженером Иоганном Хейльфрихом Мюллером.

К числу усовершенствований, введенных Мюллером, относится звонок, сигнализирующий о переполнении счетчика, используемый и в наше время в арифмометрах и пишущих машинках. Мюллером же была впервые высказана идея *разностной машины*, о которой мы расскажем в следующем параграфе. Брошюра с описанием этой идеи, изданная во Франкфурте, не получила широкой известности.

Вычислительными машинами занимался также великий русский математик и механик XIX века П. Л. Чебышев (1821—1894). Среди многочисленных изобретенных им механизмов имеется также арифмометр, сконструированный в 1878 году. По тому времени это была одна из самых оригинальных вычислительных машин. Чебышев создал для этого арифмометра весьма оригинальный механизм переноса, действовавший непрерывно.

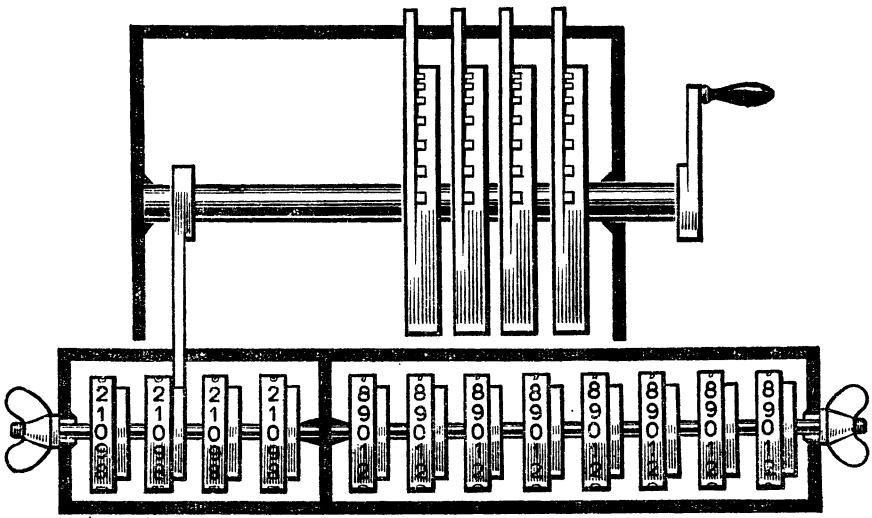


Рис. 16

Еще один способ фиксирования вводимого слагаемого был предложен в 1876 году петербургским чиновником В. Т. Однером, который изобрел *зубчатое колесо с переменным числом зубцов*. Колесо Однера было использовано в арифмометрах, которые начали выпускаться в России с 1894 года и выпускались, с некоторой модернизацией, до середины шестидесятых годов. Арифмометры «Феликс» (рис. 16), являющиеся прямыми потомками арифмометров Однера, и сейчас еще работают во многих учреждениях.

Чебышев

И ступенчатый валик Лейбница, и колесо с переменным числом зубцов Однера нашли свое применение в современных электромеханических арифмометрах, выпускавшихся в больших количествах до последнего времени. От своих предков современные арифмометры, или, как их часто называют, *клавишные машины*, отличаются тем, что их не требуется вращать вручную. Все они снабжены электроприводами и работают автоматически, делая нужное число оборотов и передвигая каретку без участия оператора. Только за последние пять — семь лет электромеханические арифмометры активно вытесняются настольными электронными вычислительными машинами.

Механическая эра отразилась и в истории машин непрерывного действия. В XIX веке было предложено несколько различных приборов для измерения длин *(лонгиметров)* и площадей *(планиметров)*. Из последних следует отметить очень удачный *полярный планиметр Амслера*, предложенный в 1854 году швейцарским математиком Я. Амслером (1823—1912). С некоторыми модификациями он применяется и до сих пор.

Наиболее интересным и важным изобретением в этой области явился механический интегратор для решения дифференциальных уравнений, построенный в 1912 году великим русским математиком и инженером А. Н. Крыловым (1863—1945). Это была первая интегрирующая машина непрерывного действия, позволяющая решать дифференциальные уравнения до четвертого порядка. Из-за отсутствия в то время других подходящих двигателей она приводилась в движение специально сконструированным часовым механизмом.

Крылов

Чарльз Бэббедж

§ 5. ПИОНЕРЫ АВТОМАТИЗАЦИИ ВЫЧИСЛЕНИЙ

Механические вычислительные машины, о которых шла речь в предыдущем параграфе, были р у ч н ы м и, т. е. требовали участия оператора в процессе вычислений. Для каждой операции нужно было ввести в машину исходные данные и привести в движение счетные элементы машины для выполнения операции. Время от времени нужно было считывать и записывать полученные результаты и контролировать правильность хода вычислений. Ясно, что для такой машины ускорение выполнения операций мало отразится на общем времени вычислений, так как большая часть времени по-прежнему будет уходить на ручную работу оператора по установке данных и считыванию результатов.

Нельзя ли создать *автоматическую вычислительную машину*, которая осуществляла бы требуемые вычисления без участия человека? Во второй половине XX века этот вопрос звучит риторически. Но в XIX веке сама постановка этого вопроса, не говоря уже о положительном его решении, требовала незаурядной творческой дерзости, глубокой интуиции и богатого научно-инженерного воображения. Первым поставил этот вопрос и сделал серьезные шаги в обосновании положительного ответа на него замечательный английский ученый, инженер и изобретатель Чарльз Бэббедж.

Чарльз Бэббедж родился в 1791 году в семье богатого лондонского банкира. Он закончил университет в Кембридже, получив в 1817 году ученую степень магистра, и специализировался в области математики. Впрочем, Бэббедж был весьма разносторонним человеком, и его перу принадлежит большое число научных работ в самых различных областях: математики, прикладной статистики, теории магнетизма, геологии, ботаники, экономики и т. д. Его книга «Экономика машин и производства», изданная в 1832 году, содержала идеи, которые спустя много лет составили основу таких современных научных дисциплин, как исследование операций, системный анализ, научное управление производством. Эту книгу высоко ценили не только современные ему экономисты. Ее знал и неоднократно цитировал (в «Капитале» и других работах) Карл Маркс.

«Главным делом жизни» Бэббеджа, по словам самого ученого, были вычислительные машины. Но и здесь сказалась присущая Бэббеджу широта взглядов: изучая возможности изготовления необходимых для его машин деталей и узлов и непосредственно участвуя в их конструировании и изготовлении, Бэббедж сделал ряд

выдающихся изобретений в области машиностроения. Он создал поперечно-строгальный и токарно-револьверный станки, различные калибры, резцы и пресс-формы, предложил метод изготовления зубчатых колес литьем под давлением и т. п.

Занимаясь расчетами, Бэббедж неоднократно обнаруживал большое число ошибок в логарифмических, астрономических и навигационных таблицах, которые возникали и вследствие ошибочных расчетов, и вследствие ошибок печатания. Поэтому он задался целью создать машину, которая будет автоматически вычислять и печатать таблицы функций.

Разностная машина, над которой Бэббедж начал работать с 1820 года, предназначалась для табулирования функций, точнее, приближающих их многочленов, с помощью хорошо известного метода разностей. Познакомимся с ним на простом примере.

Допустим, что требуется вычислить таблицу третьих степеней членов натурального ряда, т. е. табулировать функцию

$$N=n^3 \quad (n=1,\ 2,\ 3,\ \ldots).$$

Пусть такая таблица уже вычислена (табл. 1, колонки (1)—(2)). Вычтем из каждого последующего значения N предыдущее. Мы получим последовательные значения *первых разностей* (колонка (3)). Проделав ту же операцию с первыми разностями, получим *вторые разности* Δ^2 (колонка (4)) и, наконец, *третьи* Δ^3 (колонка (5)).

Как видно из полученной таблицы, третьи разности оказываются постоянными: колонка (5) состоит из одного и того же числа 6. Это не случайность, а следствие важной теоремы: *если функция есть многочлен k-й степени, то в таблице с постоянным шагом ее k-е разности постоянны.*

В таблице 1 мы вычисляли разности, зная значения функции. Нетрудно понять, что можно действовать в противоположном направлении — вычислять значения функции по известным разностям, для чего достаточно задать первую строку таблицы 1. Следовательно, надо было сделать такую машину, которая по заданным разностям до некоторого порядка могла бы вычислять значения

Таблица 1

1	2	3	4	5
n	N	Δ	Δ^2	Δ^3
1	1	7	12	6
2	8	19	18	6
3	27	37	24	6
4	64	61	30	6
5	125	91	36	...
6	216	127	...	
7	343	...		
...	...			

функции, для чего достаточно было выполнять лишь операции сложения (или вычитания, так как разности могли оказаться отрицательными).

Числа в разностной машине Бэббеджа изображались с помощью тех же десятичных счетных колес, которые применялись в машине Паскаля. Колеса собирались в *регистры*, расположенные на одной вертикальной оси, которые и служили для изображения многоразрядного числа. Основным механизмом был механизм сложения.

Модель, которую Бэббедж изготовил собственноручно в 1820—1822 годах, могла табулировать функции с постоянными вторыми разностями с точностью восьми знаков и вызвала большой интерес. Английское правительство согласилось финансировать постройку большой разностной машины, которая должна была табулировать с точностью 20 знаков функции с постоянными шестыми разностями.

Работа над изготовлением разностной машины продолжалась свыше десяти лет, но так и не была завершена. Отчасти в этом было повинно правительство, несколько раз прекращавшее финансирование и досаждавшее автору упреками и подталкиванием, отчасти и сам автор, непрерывно переделывавший одни и те же узлы и конструкции.

Описание этой машины, появившееся в 1834 году в одном из научных журналов, побудило шведа Георга Шютца и его сына Эдварда начать разработку своего варианта разностной машины. Эта машина была построена в 1854 году и табулировала с точностью 9 знаков функции с постоянными четвертыми разностями. По заказу английского правительства была построена в 1859—1863 годах копия этой машины. Кроме этих, в XIX веке было построено еще несколько разностных машин — шведом Вибергом, американцем Грантом и др.

Хотя разностная машина в процессе вычислений не требовала вмешательства оператора, это было, по современной терминологии, специализированное вычислительное устройство с фиксированной программой. Бэббедж не захотел ограничиться таким уровнем автоматизации и пошел дальше. В 1834 году он изобрел *универсальную вычислительную машину с программным управлением*, которую назвал *аналитической машиной*. Можно полагать, что это изобретение было одной из причин, побудивших его прекратить работу над разностной машиной.

Бэббедж испытал бы полное удовлетворение, узнав, что структура вновь изобретенных почти через столетие универсальных цифровых вычислительных машин по существу повторяет структуру его машины!

Аналитическая машина имела следующие составные части:

1. «Склад» для хранения чисел (по современной терминологии «накопитель», или «запоминающее устройство», «память»).

2. «Мельницу» для производства арифметических действий над числами («арифметическое устройство»).

3. Устройство, управляющее в определенной последовательности операциями машины * (сейчас — «устройство управления»).

4. Устройства ввода и вывода данных.

Для хранения чисел Бэббедж предложил использовать регистры из десятичных счетных колес. Перенос чисел из памяти в другие устройства машины предполагалось осуществлять с помощью реек, которые должны были зацепляться с зубцами на колесах. Каждая рейка продвигалась до тех пор, пока колесо не занимало нулевое положение. Движение передавалось стержнями и связями в арифметическое устройство, где посредством другой рейки оно использовалось для перемещения в нужное положение одного из колес регистра.

Бэббедж считал, что запоминающее устройство должно иметь емкость в 1000 чисел по 50 десятичных знаков, «... для того, чтобы иметь некоторый запас по отношению к наибольшему числу, которое может потребоваться». Для сравнения укажем, что запоминающее устройство одной из первых английских ЭВМ (ЭДСАК) имело объем 250 десятиразрядных чисел.

Особое внимание Бэббедж уделял конструированию арифметического устройства. Здесь ему удалось сделать одно из наиболее выдающихся своих изобретений: систему предварительного переноса (по современной терминологии — систему сквозного переноса).

Схема *сквозного переноса*, предложенная Бэббеджем, оказалась настолько удачной, что она используется сейчас во всех цифровых вычислительных машинах, как электромеханических, так и электронных. Поэтому мы отложим описание этой схемы до соответствующего места в гл. III.

Умножение и деление в *аналитической машине* выполнялось последовательными сложениями и вычитаниями соответственно. Время на производство арифметических операций оценивалось Бэббеджем так: сложение или вычитание — 1 *сек*; умножение (двух пятидесятиразрядных чисел) — 1 *мин*; деление (сторазрядное число на пятидесятиразрядное) — 1 *мин*.

Бэббедж предполагал указывать алгебраический знак числа особым зубчатым колесом, расположенным над регистром, но не соединенным с другими колесами устройством переноса. Если это колесо показывало четное число, то знак должен был считаться положительным, в случае нечетного числа — отрицательным. При умножении знак образовывался сложением, при делении — вычитанием чисел на знаковых колесах.

Для устройства управления Бэббедж предложил применить механизм, аналогичный механизму ткацкого станка Жаккара. Идея Бэббеджа заключалась в том, чтобы заставить два жаккаровских механизма с цепочкой карт в каждом управлять действиями машины.

---

* Бэббедж не дал ему названия.

Один механизм с «картами операций» *(управляющими картами)* должен был соединяться с арифметическим устройством и приводить его в состояние готовности для выполнения той или иной арифметической операции, в зависимости от отверстий, пробитых в соответствующей карте. Второй механизм должен был управлять переносом чисел из «склада» в «мельницу» и обратно.

Таким образом, с помощью жаккаровских карт — прообраза современных перфокарт, Бэббедж предполагал осуществить автоматическое управление процессом механических вычислений.

Аналитическая машина не была построена, но Бэббедж сделал более 200 чертежей ее различных узлов, около 30 вариантов общей компоновки машины и изготовил за свой счет некоторые устройства. Уже после смерти ученого, в 1871 году, его сын, генерал-майор Генри Провост Бэббедж, достроил арифметическое устройство («мельницу»).

Таким образом, история универсальных вычислительных машин с программным управлением начинается с *аналитической машины* Бэббеджа. Историю же программирования следует начинать с имени его верной ученицы и помощницы, дочери великого английского поэта Дж. Г. Байрона, Августы Ады Лавлейс.

В 1840 году, по приглашению итальянских математиков, Бэббедж посетил Турин, где прочел цикл лекций о своей аналитической машине. Эти лекции законспектировал, а затем издал отдельной статьей военный инженер Л. Ф. Менабреа, впоследствии генерал в армии Гарибальди, а затем премьер-министр Италии. Эта статья была переведена на английский язык Адой Лавлейс и издана ею в 1843 году вместе с обширными комментариями, превышающими в несколько раз текст самой статьи.

Некоторые вопросы программирования были рассмотрены самим Бэббеджем, например идея условной передачи управления, о которой мы будем подробно говорить в гл. V. Но большинство идей и принципов программирования для аналитической машины, имеющих значение и для современных универсальных вычислительных машин, было рассмотрено в «Комментариях» Ады Лавлейс. Ей принадлежат некоторые термины, употребляемые программистами и сейчас, например *рабочие ячейки*. Она же ввела одно из центральных понятий программирования — *цикл*, о чем нам придется достаточно подробно говорить в главах, посвященных программированию (гл. V).

К сожалению, «Комментарии» оказались единственной научной работой замечательной женщины; в 1852 году в возрасте 37 лет она умерла и была похоронена рядом с могилой ее великого отца.

Следующий важный шаг на пути автоматизации вычислений был сделан примерно через 20 лет после смерти Бэббеджа американцем Германом Холлеритом (1860—1929), который изобрел электромеханические машины для вычислений с помощью перфокарт, получившие название *счетно-аналитических машин*.

Закончив в 1879 году Горную школу при Колумбийском университете, Холлерит поступил в статистическое управление при Министерстве внутренних дел США, где принимал участие в обработке данных по переписи 1880 года. Позже, работая с 1882 года в Массачусетском технологическом институте и затем в Бюро патентов,

он начал разрабатывать машины для механизации обработки данных переписи.

Система Холлерита, основные патенты на которую были получены им в 1884—1889 годах, включала перфокарту, перфоратор, позволяющий пробивать отверстия в нужных местах перфокарты, сортировальную машину и табулятор. Основная идея Холлерита состояла в том, чтобы изображать различные характеристики отверстиями в определенных местах перфокарты. Последняя представляет собой кусок картона определенного размера с определенным числом позиций. Современная перфокарта подробно описана в гл. IV. Первые карты Холлерита имели такую же форму, но другое число строк и колонок.

Пробивка отверстий осуществлялась вручную на перфораторе, позволявшем подготовлять 500—700 карт в день. Сортировальная машина позволяла распределить карты на группы, в зависимости от места пробитого отверстия. Она представляла собой несколько закрытых крышками ящиков. Карта продвигалась вручную между набором штырей, насаженных на пружины, и резервуаром, наполненным ртутью. Если штырь попадал в отверстие, он касался ртути и замыкал электрическую цепь. При этом поднималась крышка определенного ящика, куда и попадала карта. Табулятор работал аналогично сортировке, но в нем обнаруженное отверстие использовалось для увеличения на 1 показания определенного, связанного с данным штырем счетчика.

Таким образом, с помощью сортировальной машины можно было распределить колоду перфокарт на группы, обладающие различными требуемыми признаками, а при помощи табулятора — подсчитать число карт в каждой группе. Это дало возможность применить эти машины для обработки данных переписи населения, что позволило, например, провести обработку данных переписи 1890 года втрое быстрее предыдущей.

Дальнейшее усовершенствование счетно-аналитических машин позволило во много раз ускорить их работу и возложить на табулятор более сложные обязанности. Тем не менее и сейчас комплекты счетно-аналитических машин применяются, главным образом, для обработки статистических данных.

Впрочем, можно упомянуть, что в 1920—1940 годы делались успешные попытки применить счетно-аналитические машины для научно-технических расчетов. Основные достижения в этой области связаны с именем английского астронома и вычислителя Л. Дж. Комри. Разумеется, речь шла уже о модернизированных машинах, без ртути, и с печатающими устройствами.

§ 6. НЕДОЛГИЙ ВЕК РЕЛЕЙНЫХ МАШИН

Еще в 1937 году Говард Айкен, работая над своей докторской диссертацией по физике, требовавшей сложных расчетов, начал придумывать различные машины для автоматического решения частных задач, но затем пришел к идее автоматической универсальной вычислительной машины. Эта машина получила название «вычислительной машины с автоматическим управлением последовательностью операций», но больше известна под именем «Марк-1». Она была построена и передана Гарвардскому университету (США) в 1944 году.

Случайно познакомившись, спустя три года после начала работы над машиной, с идеями Бэббеджа и убедившись во многих совпадениях, Айкен сказал: «Живи Бэббедж на 75 лет позже, я остался бы безработным!»

Как и в аналитической машине Бэббеджа, числа в «Марк-1» хранились в регистрах из десятичных счетных колес. Регистры были 24-разрядными (из них один разряд использовался для записи знака) и имели механизмы переноса десятков. Поэтому они могли использоваться не только для хранения чисел, но и для операций над ними, выполняя одновременно обязанности и ячейки памяти, и элемента арифметического устройства. Таких регистров было в машине 72. Кроме них, машина имела еще 60 регистров «пассивной памяти», куда можно было вручную вводить числа перед началом вычислений, но они должны были оставаться там неизменными.

Для управления операциями в «Марк-1» использовались уже электромеханические элементы — *реле-переключатели*. Щетка (считывающий контакт) пробегала по десяти неподвижным контактам, присоединяясь к каждому колесу регистра таким образом, что можно было получить электрический эквивалент числа, хранящегося в данном разряде. При сложении устанавливалось соединение между щетками одного накопителя и механизмом переключения второго. Скорость выполнения операций оказывалась при этом больше, чем для чисто механических элементов, например сложение требовало около 0,3 секунды, умножение — 5,7 секунды.

Работой машины управляли *команды*, вводимые с перфоленты; механизм считывания был таким, как на счетно-аналитических машинах. Некоторые команды определяли тип операции, другие сообщали, из каких регистров следует брать числа и куда направлять результат. Таким образом, здесь объединялись оба типа бэббеджевских карт — операционные и адресные. Заметим, что первоначальная система команд «Марк-1» не содержала команды условного перехода, что весьма ограничивало возможности машины. Условный переход был включен в систему команд позднее, быть может, вследствие знакомства Айкена с работами Бэббеджа.

В 1945 году группа Айкена начала работу над машиной «Марк-2», которая была чисто *релейной вычислительной машиной*, где и элементами для представления чисел служили электромеханические реле. Так как для реле характерны два устойчивых состояния, а идея отказаться от десятичной системы счисления еще не приходила в голову конструкторам, числа в «Марк-2» записывались в смешанной двоично-десятичной системе: каждая десятичная цифра представлялась в двоичной системе и хранилась в группе из четырех реле.

Числа представлялись в плавающей форме, т. е. в виде $N = N_0 \times 10^p$, с 10-значной мантиссой N_0 и порядком p, заключенным в интервале — $15 \leqslant p \leqslant 15$. Таким образом, 40 реле требовалось для записи мантиссы, еще 4 для записи порядка и 2—для записи знаков порядка и мантиссы; поэтому регистр (ячейка памяти) машины «Марк-2» состояла из 46 реле. Машина имела 100 таких регистров.

Суммато́р, в котором выполняется сложение, в отличие от сумматора «Марк-1» отделен от памяти. Скорость работы «Марк-2» составляет уже около 8 сложений или 4 умножения в секунду. Кроме того, «Марк-2» содержит специализированные устройства для вычисления некоторых элементарных функций.

Наряду с работами группы Айкена приблизительно в то же время велась работа других групп, в результате которой было создано еще несколько электромеханических релейных машин. Так, еще в 1939 году была закончена и в 1940 году демонстрировалась релейная машина американского математика Дж. Штибитца «Модель-I», которая выполняла четыре арифметических действия над комплексными числами. В этой машине использовалось смешанное двоично-пятеричное представление чисел, допускающее простую аппаратную реализацию.

Дальнейшая успешная разработка малых специализированных машин на тех же принципах привела к созданию в 1944—1946 годах универсальной релейной вычислительной машины «Модель-V», содержащей около 9 тысяч реле. Скорость работы этой машины составляла 3 сложения или 1 умножение в секунду.

Одной из наиболее совершенных релейных вычислительных машин была советская релейная вычислительная машина РВМ-1, сконструированная в начале пятидесятых годов выдающимся инженером Н. И. Бессоновым (1906—1963) и построенная в 1956 году. Машина имела 5500 реле. Числа представлялись в ней в плавающей форме в двоичной системе счисления, ячейка памяти имела 34 двоичных разряда. Запоминающее устройство имело 50 ячеек активной памяти и 128 ячеек пассивной, а скорость работы составляла 50 сложений или 20 умножений в секунду. Эта машина успешно работала до 1966 года.

Главным недостатком релейных вычислительных машин являлось отсутствие хранимой в памяти программы, что обусловливалось небольшим объемом оперативной памяти, и невысокая скорость работы, вызванная низким быстродействием электромеханических релейных переключателей. Поэтому релейные машины были быстро вытеснены электронными. В истории вычислительной техники они будут теперь занимать почетное место *первых действовавших автоматически универсальных вычислительных машин с программным управлением.*

§ 7. «ЭЛЕКТРОННЫЙ МОЗГ»

Работа над первой электронной вычислительной машиной, в которой для изображения чисел и для осуществления операций над ними использовались электронные элементы, началась в США в середине 1943 года. Машина строилась по заказу артиллерийского управления и предназначалась для расчета баллистических таблиц. Руководили работой американские ученые Дж. В. Моучли и Д. П. Эккерт.

Постройка машины была завершена в конце 1945 года; машина получила названия ЭНИАК (ENIAC — *Electronic Numerical Integrator and Computer*, электронный цифровой интегратор и вычислитель). Это сооружение содержало свыше 18 тысяч электронных ламп и полутора тысяч реле и потребляло мощность около 150 *квт*.

Использование электронных ламп вместо механических и электромеханических элементов позволило резко увеличить скорость выполнения операций: ЭНИАК тратит на умножение 2,8 *мсек*, а на сложение 0,2 *мсек* (*мсек* (миллисекунда) $= 10^{-3}$ *сек*). По принципам работы ЭНИАК имеет много общего с «Марк-1» и «Марк-2». Числа здесь представляются в двоично-десятичной системе, и запоминающие регистры снабжены системой сквозного переноса, благодаря чему могут использоваться для сложения или вычитания. Другие арифметические операции выполняются в специализированных блоках.

Основным элементом для запоминания двоичной цифры служила *триггерная ячейка* или *триггер* — схема, которая может находиться в одном из двух устойчивых состояний. Эта схема была разработана в 1918 году известным русским радиотехником М. А. Бонч-Бруевичем и независимо от него в 1919 году американскими специалистами У. Икклзом и Ф. Джорданом; мы познакомимся подробно с этими схемами и их использованием в гл. III.

Управление автоматической работой ЭНИАКа с помощью вводимой программы, как на релейных машинах, оказалось невозможным из-за слишком большого различия в скорости выполнения операций и скорости (надо бы сказать — медленности) ввода перфоленты или перфокарты; у релейных машин эти скорости с о и з м е р и м ы. Поэтому конструкторы использовали для задания программы *штеккерно-коммутационный способ*, распространенный в счетно-аналитических машинах. Такой способ требует много времени для предварительной подготовки, состоящей в выполнении соединений на коммутационной доске, но зато позволяет реализовать высокие способности ЭНИАКа.

Начиная с 1944 года в работе над созданием электронных вычислительных машин принял участие один из крупнейших американских математиков Джон фон Нейман. Он родился в 1903 году в Будапеште. Получив степень доктора математики в Будапештском университете и диплом химика в Швейцарской высшей технической школе в Цюрихе, фон Нейман переехал в США в 1930 году и с 1933 года до своей смерти в 1957 году работал научным сотрудником знаменитого института перспективных исследований в Принстоне.

В 1946 году была опубликована весьма важная для дальнейшего развития вычислительной техники статья Дж. фон Неймана, Г. Голдстайна и А. Беркса «Предварительное рассмотрение логической конструкции электронного вычислительного устройства». В этой статье были высказаны две основные идеи, которые используются сейчас во всех электронных вычислительных машинах: использование двоичной системы счисления и принцип хранимой программы.

Удобство двоичной системы для работы с электронными элементами, имеющими два устойчивых состояния, читателю должно быть уже достаточно ясно. Хранимая в памяти программа позволяла преодолеть важнейший недостаток ЭНИАКа — затраты времени на набор и подготовку программы на коммутационной доске. Программу, как и исходные числовые данные, предлагалось хранить в памяти машины. Отдельные команды извлекались из памяти устройством управления, расшифровывались и использовались для извлечения чисел из памяти, выполнения требуемых операций и отсылки результатов в память.

Хранение программы в памяти давало и еще одно очень важное преимущество. Для записи команды в память ей необходимо придать числовой вид. Это позволяет производить преобразование команд в процессе работы машины. Такие преобразования делают возможным *переадресацию* и *формирование команд* самой машиной. В свою очередь возможность формирования команд позволила создать алгоритмические языки и передать не только выполнение, но и заметную часть составления программы самой машине.

Статья «Предварительное рассмотрение логической конструкции. . .» оказала большое влияние на новые разработки и проекты. Началась постройка машины ЭДВАК, использующей высказанные идеи и принципы. Но эта машина была завершена лишь в 1950 году, а годом раньше, в 1949 г., под руководством проф. М. В. Уилкса в Кембриджском университете была построена английская машина ЭДСАК, которая оказалась, таким образом, первой электронной вычислительной машиной с хранимой программой. Скорость выполнения арифметических операций на машине ЭДСАК характеризовалась такими данными: умножение — 8,5 *мсек*, сложение — 70 *мксек* (1 *мксек* (микросекунда) $= 10^{-6}$ *сек*).

Заметим, что скорость вычислительной машины начала уже определяться не только временем работы арифметического устройства, но и временем обращения к памяти, которое существенно зависело от физической реализации запоминающих элементов. Память машины ЭНИАК использовала триггерные ячейки, которые были быстродействующими, но очень дорогими и их поэтому было мало (всего 20 регистров). Для машины с хранимой программой необходимо было иметь большую память. ЭДСАК имела память на ртутных линиях задержки емкостью 512 чисел.

Машина «Джониак», построенная в 1946—1952 годах и названная так в честь фон Неймана, имела память на электроннолучевой трубке специальной конструкции. Позднее здесь был применен для запоминающего устройства магнитный барабан, впервые использованный в 1947 году в небольшой английской вычислительной машине, построенной под руководством Э. и К. Бут. Магнитная лента была впервые использована в 1951 году на машине ЮНИВАК, построенной Моучли и Эккертом. Это была первая большая вычислительная машина, построенная не по специальному заказу, а для продажи.

В начале пятидесятых годов были сконструированы и изготовлены первые советские электронные вычислительные машины. Первой из них была построена БЭСМ (быстродействующая электронная счетная машина), которая разрабатывалась и строилась под руководством акад. С. А. Лебедева и была завершена в 1951—1952 годах. Затем были созданы машины М-2 (1952 г., под руководством И. С. Брука), «Стрела» (1953 г., под руководством Ю. И. Базилевского) и «Урал» (1954 г., под руководством Б. И. Рамеева).

Принятая классификация, в основу которой положен физико-технологический принцип, делит электронные вычислительные машины на различные *поколения*. Все перечисленные выше электронные вычислительные машины, как зарубежные, так и советские, относятся к *первому поколению*. В машинах первого поколения основной физический элемент, используемый в счетных цепях и схемах,— электронные вакуумные лампы.

Уже в середине пятидесятых годов появились «первые ласточки» *второго поколения* машин, в которых вакуумные лампы были заменены полупроводниковыми элементами — *транзисторами*. Это не значит, конечно, что весь парк машин был тут же полностью заменен. Но к началу шестидесятых годов выпуск машин первого поколения полностью прекратился, а к середине все они были сняты и с эксплуатации.

Наша книга (во всяком случае, основные главы III—VI) посвящена именно машинам второго поколения, которые в настоящее время составляют подавляющее большинство парка наших вычислительных машин, находящихся в эксплуатации. Но небольшое число эксплуатируемых машин принадлежит уже к *третьему поколению*.

Для машин третьего поколения характерно изменение в технологии изготовления полупроводниковых элементов. Появившаяся *интегральная технология* позволила объединить в одном электронном приборе несколько вентилей или триггеров. Использование *интегральных схем* является отличительным признаком машин этого поколения, которые, видимо, скоро полностью сменят машины второго поколения.

Интегральная технология не стоит на месте, и ее развитие позволяет наметить контуры *четвертого поколения* и даже *пятого*, разумеется, гораздо более смутно и расплывчато. Сейчас уже изготовляются *средние интегральные схемы*, содержащие до 50 вентилей, ячеек и несколько десятков триггеров, которые могут быть использованы в качестве отдельных *операционных схем* — регистров, счетчиков, дешифраторов и т. п. Это и есть элементно-технологическая основа будущих машин четвертого поколения, первые экземпляры которых уже появились на свет. Что касается пятого поколения, то, видимо, оно будет связано с *большими интегральными схемами*, которые смогут выполнять роль отдельных устройств.

Конечно, переход от одного поколения к другому не состоит только в изменении элементно-технологической базы; он непременно сопровождается и более глубокими изменениями в структуре маши-

ны, принципах соединения различных устройств, практике эксплуатации, способах и системах программирования, возможностях применения и т. п. Но некоторые основные положения уже стабилизируются. И видимо, немало серьезного в шутке крупного специалиста в области программирования, голландского математика Э. Дикстры: «Для того чтобы появился настоящий джентльмен, необходимо, по мнению англичан, хотя бы три предшествующих благородных поколения . . . Очевидно, это справедливо и для вычислительных машин».

Читатель, по-видимому, заметил, что по мере приближения к нашему времени параграфы этой главы все более «худели». Это произошло не случайно. В начальных параграфах мы хотели не только познакомить читателей с историей развития «счетной техники», но и рассказать о принципах работы некоторых элементарных приспособлений и машин, которые находят еще значительное распространение в вычислительной практике и, без сомнения, относятся к предмету нашей книги. Для того чтобы рассказать об устройстве и использовании современных вычислительных машин, нам понадобилось написать не один параграф, а целую книгу — ту самую, которая и находится сейчас перед глазами читателя.

Глава III

ЦИФРОВЫЕ МАШИНЫ. ПЕРВОНАЧАЛЬНЫЕ СВЕДЕНИЯ

§ 1. БЛОК-СХЕМА ЦИФРОВОЙ ВЫЧИСЛИТЕЛЬНОЙ МАШИНЫ

Применение цифрового способа изображения чисел диктует свои принципы устройства, о которых мы уже бегло говорили в гл. I.

Современные быстродействующие универсальные вычислительные машины, о которых будет идти речь в настоящей главе, обычно называют электронными, поскольку они построены с применением электронных элементов. Однако основное их отличие вовсе не в том, что они используют электронные элементы, а в том, что они *цифровые*. С математической точки зрения основные принципы работы цифровых вычислительных машин одни и те же, как для современных электронных вычислительных машин любого поколения, так и для релейных машин Говарда Айкена или Н. И. Бессонова, и для чисто механической *аналитической машины* Чарльза Бэббеджа, о которой мы достаточно подробно говорили в гл. II.

Цифровая вычислительная машина должна содержать следующие основные устройства:

а) арифметические устройства *(арифметика)*;
б) запоминающее устройство *(память)*;
в) устройство управления *(управление)*;
г) устройство ввода данных *(ввод)*;
д) устройство вывода результатов вычислений *(вывод)*.

Кроме того, машина снабжается также *пультом* ручного управления и сигнализации. Соединение перечисленных устройств между собой — *блок-схема* цифровой вычислительной машины *— показана на рисунке 17.

Рассмотрим кратко назначение и роль упомянутых устройств, представляющих собой *функционально независимые части машины*.

Арифметическое устройство машины служит для выполнения арифметических, логических или других операций над числами, на

* Как говорилось в гл. II., *Аналитическая машина* Бэббеджа содержала все основные устройства современных универсальных вычислительных машин. Теперь читатель может убедиться, что это действительно так.

которые распадается любой вычислительный процесс. Основой арифметического устройства является *сумматор*, осуществляющий сложение двух чисел. Выполнение всех других операций сводится к сложению (однократному или многократному) и некоторым другим вспомогательным действиям, например сдвигам, нужным при выполнении умножения. Характеристикой арифметического устройства служит его быстродействие, т. е. время, затрачиваемое на каждую операцию или количество операций в единицу времени, а также состав возможных операций, выполняемых устройством.

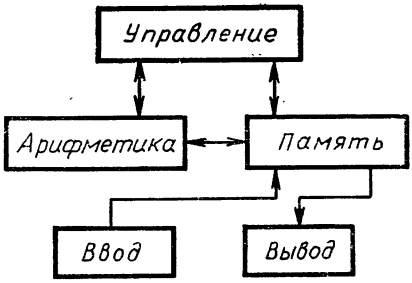


Рис. 17

Память машины служит для хранения всей требуемой информации, как исходных данных, так и промежуточных и окончательных результатов вычислений. Кроме того, в современных машинах память хранит также и записанный в особой форме алгоритм решения задачи; подробнее об этом речь будет идти в следующем параграфе. Память воспринимает информацию, передаваемую из других устройств, например из *арифметики*, или поступающую в машину извне через *ввод*, и выдает во все другие устройства информацию*, требуемую для протекания вычислительного процесса.

Характеристиками памяти машины являются ее *объем (емкость)* и *быстродействие (скорость* или *время обращения)*.

Быстродействие памяти определяется временем обращения, т. е. временем, необходимым для записи в память или считывания из памяти. Естественными требованиями к памяти являются возможно большая скорость, т. е. возможно меньшее время обращения, и одновременно возможно больший объем. Легко понять, что эти требования противоречивы: чем больше емкость памяти, тем больше времени тратится на выполнение операции записи или считывания. Противоречие разрешается разделением запоминающего устройства на *оперативную* (или *внутреннюю*) память и *внешнюю* память.

Оперативная память машины делится на отдельные *ячейки*, состоящие из определенного, постоянного для данной машины числа *разрядов*. Каждый из разрядов может принимать одно из двух возможных значений, 0 или 1. Таким образом, содержимое ячейки памяти или любая его часть могут восприниматься как двоичное число. Содержимое отдельной ячейки памяти принято называть *машинным словом*, так что объем памяти характеризуется количеством машинных слов, которые могут быть в ней записаны.

---

* Мы оставляем пока этот термин без определения. В дальнейшем из конкретных примеров будет ясно, что он означает в том или ином случае.

Для удобства обращения ячейки оперативной памяти занумерованы; номер ячейки называется ее *адресом*. Оперативная память непосредственно общается с остальными устройствами машины, в частности с *управлением* и с *арифметикой*. Числа, участвующие в арифметической (или другого типа) операции, отыскиваются по их адресам и передаются в нужные устройства. Ячейка памяти устроена так, что при считывании из нее машинного слова содержимое ячейки сохраняется и может быть снова считано из той же ячейки любое число раз. Оно изменяется лишь в том случае, если в данную ячейку записывается новое машинное слово; тогда предыдущее автоматически стирается.

Время обращения к оперативной памяти составляет единицы микросекунд или даже доли микросекунды. Вместе с тем объем ее сравнительно невелик. На машинах первого поколения он составлял 2^{11}—2^{12} (2048—4096) машинных слов; на машинах второго поколения от 2^{13} до 2^{15}, а иногда и 2^{16}. Как правило, оперативная память бывает выполнена на ферритах.

Другая часть запоминающего устройства — *внешняя память* — выполняется с помощью магнитной записи на магнитных барабанах, магнитных лентах (типа магнитофонных) или магнитных дисках. Внешняя память характеризуется существенно меньшим быстродействием, зато большим объемом. Каждая машина имеет не менее четырех барабанов, а емкость каждого из них равна двум или четырем объемам оперативной памяти. На магнитной ленте записывается от 200 до 500 тысяч слов, причем машина может иметь от 4 до 16 магнитофонов. Емкость диска промежуточная между емкостью магнитного барабана и ленты.

Устройства внешней памяти непосредственно не участвуют в обмене информацией с другими устройствами машины. Весь обмен идет только через оперативную память, даже если речь идет о переписывании содержимого одной ленты на другую. При этом обмен производится не отдельными машинными словами, как между оперативной памятью и *арифметикой*, а целыми группами слов.

Ввод служит для преобразования исходных данных в удобную для машины форму и ввода всей требуемой информации в память машины. Информация из ввода поступает непосредственно лишь в оперативную память. В случае необходимости записи каких-либо данных во внешнюю память это приходится делать через оперативную, используя общие приемы обмена информацией между оперативной и внешней памятью. *Вывод* предназначается для преобразования результатов вычислений в форму, удобную для человеческого восприятия и дальнейшего использования. Устройства ввода и вывода, так же как и отдельные устройства внешней памяти (барабаны, ленты, диски), относятся к *периферийным устройствам* цифровых вычислительных машин.

Устройство управления обеспечивает определенное взаимодействие всех устройств вычислительной машины между собой, *автоматически* управляя всем вычислительным процессом. Важной частью

устройства управления является *счетчик команд* (или, как его часто называют, *командный регистр адреса)*, с назначением которого мы познакомимся в следующем параграфе. Автоматическое управление процессом решения задачи достигается на основе применения принципа *программного управления*, предложенного еще в 1834 году Чарльзом Бэббеджем (см. гл. II). Подробному изложению сути программного управления мы и посвятим следующий параграф.

§ 2. ПРОГРАММА И КОМАНДЫ

Процесс решения задачи состоит в осуществлении некоторого *алгоритма* — определенной предписанной последовательности операций. При этом под операцией следует понимать не только арифметические или алгебраические операции, но также и действия другого типа: сравнение чисел, переход к новой операции, условные действия (которые следует или не следует производить в зависимости от выполнения или невыполнения некоторых условий) и т. п.

Запись алгоритма может быть более или менее подробной в зависимости от того, какие операции можно считать элементами алгоритма, например отдельные арифметические операции или вычисление по формуле. В первом случае, скажем, алгоритм вычисления величины $y=(a+bx)/(ax+b)$ придется записать так:

1) вычислить $b\times x=R_1$; перейти к следующей операции;
2) вычислить $a+R_1=R_2$; перейти к следующей операции;
3) вычислить $a\times x=R_3$; перейти к следующей операции;
4) вычислить $R_3+b=R_4$; перейти к следующей операции;
5) вычислить $R_2:R_4=y$; закончить вычисления (R_1—R_4— промежуточные результаты), тогда как во втором случае алгоритм будет состоять из одного шага:

1) вычислить $(a+bx)/(ax+b)=y$.

Будем считать элементами алгоритма отдельные арифметические операции. Тогда алгоритм решения квадратного уравнения $ax^2+bx+c=0$ запишется в виде:

1) вычислить $b\times b=R_1$; перейти к следующей операции;
2) вычислить $a\times c=R_2$; перейти к следующей операции;
3) вычислить $4\times R_2=R_3$; перейти к следующей операции;
4) вычислить $R_1-R_3=D$; перейти к следующей операции;
5) проверить $D\geqslant 0$; если неравенство справедливо, то перейти к следующей операции; в противном случае перейти к операции 13);
6) вычислить $\sqrt{D}=R_4$; перейти к следующей операции;
7) вычислить $a+a=R_5$; перейти к следующей операции;
8) вычислить $R_4:R_5=R_6$; перейти к следующей операции;
9) вычислить $b:R_5=R_7$; перейти к следующей операции;
10) вычислить $R_6-R_7=x_1$; перейти к следующей операции;
11) вычислить $R_6+R_7=R_8$; перейти к следующей операции;
12) вычислить $-R_8=x_2$; закончить вычисления;
13) вычислить $-D=R_4$; перейти к следующей операции;
14) вычислить $\sqrt{R_4}=R_5$; перейти к следующей операции;

15) вычислить $a+a=R_6$; перейти к следующей операции;
16) вычислить $b : R_6=R_7$; перейти к следующей операции;
17) вычислить $-R_7=\alpha$; перейти к следующей операции;
18) вычислить $R_5 : R_6=\beta$; перейти к следующей операции;
19) вычислить $-\beta=\gamma$; закончить вычисления.

Приведенный алгоритм позволяет вычислить корни квадратного уравнения x_1, x_2, в случае когда они действительные, или действительную и мнимую части комплексных корней $\alpha+\beta i$, $\alpha+\gamma i$. В число элементарных операций этого алгоритма входят не только арифметические или алгебраические операции, но также и проверка справедливости неравенства. Кроме того, каждая операция содержит указание о том, что делать дальше — переходить к следующей операции, или к какой-либо другой, или закончить вычисления.

Чтобы поручить осуществление алгоритма автоматической цифровой машине, следует, прежде всего, представить его в виде последовательности таких элементарных операций, каждая из которых может быть выполнена машиной, т. е. в виде *последовательности элементарных операций машины.*

Запись алгоритма решения данной задачи в виде последовательности элементарных операций данной машины называют *программой.* Точнее следует говорить о программе решения данной задачи, написанной для данной машины. *Программа* состоит из последовательности *команд*, каждая из которых содержит информацию об отдельной элементарной операции. Команда определяет работу машины в течение промежутка времени, необходимого для ее выполнения.

Структура машинной команды определяется характером содержащейся в ней информации. Прежде всего, команда должна содержать сообщение о том, какая именно операция в данный момент должна быть выполнена. Кроме этого, необходимо указать, над какими числами следует эту операцию совершить.

При работе на арифмометре или клавишных вычислительных машинах мы сами сообщаем машине, т. е. набираем на ее клавишах, нужные числа. В программно-управляемой машине все исходные и промежуточные данные хранятся в ячейках памяти; поэтому в команде следует указать не сами числа, а а д р е с а ячеек памяти, в которых эти числа хранятся. В этом состоит *адресный принцип*, являющийся главной составной частью программного управления.

Соответственно сказанному выше команда в современной цифровой вычислительной машине делится на две части — *операционную* и *адресную*. Операционная часть содержит информацию о требуемой операции. Для этой цели каждой элементарной операции машины присвоен определенный номер, который называется *кодом операции*. Определенные, всегда одни и те же, разряды ячейки памяти при записи команды отводятся для операционной части команды. В этих разрядах записывается условное число — код операции. По этому коду *управление* настраивает *арифметику* машины на выполнение требуемой операции.

Другая часть команды — адресная; для нее также отводятся определенные разряды ячейки, в которых и записываются адреса ячеек памяти, содержащих требуемые в данной операции числа. Количество адресов в команде, вообще говоря, может быть различным, но для каждой конкретной машины оно всегда одно и то же; просто существуют машины различной *адресности*, которая и определяется числом адресов в команде.

Большинство цифровых вычислительных машин имеет так называемое *естественное управление*: после выполнения команды, взятой из некоторой ячейки с адресом $Я$, машина автоматически переходит к выполнению команды из ячейки с адресом $Я+1$. Именно здесь и участвует упомянутый в предыдущем параграфе *счетчик команд* (или *командный регистр адреса*). Если адрес какой-либо ячейки памяти попадает в счетчик команд, то содержимое этой ячейки вызывается в устройство управления и расшифровывается как очередная команда, которая и выполняется. После этого к содержимому счетчика команд автоматически прибавляется единица, чем и осуществляется естественное управление.

Уместно здесь же пояснить часто используемый в программировании термин. О ячейке, адрес которой записан в данный момент в счетчике команд, т. е. о ячейке, из которой в данный момент вызвана команда для исполнения, говорят, что в этой ячейке *находится управление*. Пользуясь этим термином, можно сказать, что естественное управление автоматически передает управление следующей ячейке.

Машины с естественным управлением можно сделать трехадресными. Такими и являются советские машины второго поколения БЭСМ-4, М-220, М-222. В арифметических и прочих аналогичных командах трехадресных машин первый и второй адреса являются адресами ячеек памяти, содержащих числа, над которыми выполняется операция. Третий адрес указывает ячейку, предназначенную для записи результата. После завершения операции регистры арифметического устройства, в том числе и сумматор, свободны от чисел, относящихся к предыдущей операции, и если ее результат потребуется для следующей операции, то его нужно снова вызвать из ячейки, куда он был записан.

Строятся также машины с меньшим числом адресов, двухадресные и одноадресные. Двухадресными являются, например, советские вычислительные машины Минск-22, Минск-32. В двухадресных машинах в команде, кроме кода операции, указываются только два адреса, которые могут использоваться по-разному, в зависимости от того, какая модификация операции рассматривается. Пусть, например, речь идет о сложении. Возможно несколько модификаций сложения. В одном случае два числа в команде означают адреса слагаемых; сумма остается в сумматоре и может быть взята оттуда для дальнейших вычислений. В другом случае запись суммы в память производится по второму адресу, так что находившееся там слагаемое стирается. В третьей модификации два адреса

команды означают адрес второго слагаемого и суммы; первое слагаемое берется из сумматора. Существуют и другие модификации, на которых мы не останавливаемся. Естественно, в такой машине код операции должен содержать указание не только на то, какая операция выполняется, но и какая ее модификация имеется в виду.

К одноадресным вычислительным машинам относятся машины второго поколения: Урал-11, Урал-14 и наиболее мощная из машин второго поколения БЭСМ-6, а также машины третьего поколения типа Ряд (ЕС 1010—ЕС 1050). Здесь в команде, кроме кода операции, указывается адрес лишь одной ячейки памяти. Для арифметической операции этот адрес всегда означает адрес одной из компонент (слагаемого, сомножителя и т. п.). Имеется в виду, что вторая компонента операции уже находится в сумматоре машины; здесь же остается и результат операции после ее завершения.

Мы можем теперь коротко описать последовательность действий цифровой вычислительной машины при решении некоторой задачи. Составленная программа набивается на внешнем носителе (перфокарты или перфолента) вместе с исходными числовыми данными и через устройства ввода вводится в оперативную память. Затем в счетчик команд вручную с пульта управления или автоматически с устройства ввода записывается адрес ячейки, с которой начинается программа. Устройство управления вызывает содержимое этой ячейки и расшифровывает его, выделяя код операции и адреса. По коду арифметическое устройство настраивается на выполнение требуемой операции, а по адресам из памяти вызываются в арифметическое устройство требуемые числа. После записи полученного результата управление передается следующей ячейке, из которой вызывается следующая команда, и так далее. Последняя команда осуществляет остановку машины, но перед этим должны встретиться команды, обращающиеся к устройству вывода, которые выводят полученные результаты на печать, перфорацию и т. п.

§ 3. ЭЛЕМЕНТЫ ЦИФРОВЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

В § 1 мы познакомились с основными устройствами цифровой вычислительной машины и их назначением. Теперь нам предстоит «углубиться» в каждый прямоугольник блок-схемы и разобрать принципы построения и работы этих устройств. Подобно аналоговым вычислительным машинам, состоящим из блоков, предназначенных для осуществления тех или иных операций, цифровые вычислительные машины состоят из отдельных *элементов*, обеспечивающих выполнение элементарных операций.

Заметим, кстати, что термин *элементарная операция* оказывается неоднозначным. Под этими словами вообще следует понимать простейшую, не расчленяемую далее операцию. Однако в разных ситуациях расчленение ограничивается различными уровнями.

Как уже говорилось в предыдущем параграфе, при записи алгоритма можно считать элементарной операцией вычисление по формуле или отдельную арифметиеекую операцию. Мы остановились там на понимании элементарной операции как отдельного арифметического действия. Но с точки зрения фактического исполнения эта операция поддается дальнейшему расчленению; например, сложение двух многозначных чисел расчленяется на сложение цифр в отдельных разрядах и переносы из разряда в разряд, не говоря уже о необходимости выполнения некоторых вспомогательных действий (вроде переноса слагаемого из *памяти* в *арифметику*) при выполнении сложения в цифровой машине.

В дальнейшем всюду под *элементарной операцией машины* мы будем понимать операцию, соответствующую команде программы и имеющую определенный код в системе команд машины (см. также § 1—2 гл. V). Части, на которые расчленяется эта операция, когда мы рассматриваем ее машинную реализацию, будем называть *элементами* или *тактами* элементарной машинной операции. В таком случае следует говорить, что элементы машины предназначены для осуществления элементов (или тактов) машинных операций.

Машинные элементы делятся на *логические*, *запоминающие* и *вспомогательные*. Из логических элементов строят *операционные схемы*, обеспечивающие выполнение арифметических и иных операций над машинными словами. Запоминающие элементы предназначаются для хранения информации. Из них организуются запоминающие устройства. Наконец, вспомогательные элементы предназначены для формирования стандартных сигналов и согласования режимов работы операционных схем. Эта классификация, разумеется, несколько условна, поскольку одни и те же элементы могут использоваться различным образом. Так, запоминающий элемент — *триггер* — может быть построен из логических элементов, а операционная схема — *регистр* — может рассматриваться как запоминающее устройство.

Современные цифровые вычислительные машины работают в двоичной системе счисления, преимущества и недостатки которой мы рассмотрели в главе I. Для изображения чисел в этой системе требуются лишь две цифры и поэтому для их записи используются элементы, имеющие ровно два устойчивых состояния; одно из них принимается за изображение нуля, а другое—за изображение единицы. Запоминающий элемент должен хранить одну двоичную цифру, а арифметическое устройство — осуществлять арифметические операции в двоичной системе, где они обладают особой простотой и наглядностью. Вообще, каждое машинное слово, какую бы роль оно не играло, представляет собой набор двоичных цифр, т. е. нулей и единиц.

Результат любой операции, выполняемой в машине, есть двоичное число. Поэтому операционную схему можно рассматривать как некоторый функциональный преобразователь, на входах и выходах

которого — двоичные числа. Еще удобнее считать аргументами и значениями функции отдельные разряды двоичных чисел. В этом случае наш функциональный преобразователь приобретает вид устройства с несколькими входами и выходами, причем на каждый вход подается и с каждого выхода снимается двоичная цифра — 0 или 1.

Таким образом, речь идет о переменных и функциях, принимающих лишь значения 0 или 1. Такие переменные и функции носят название *логических* или *булевских* — по имени английского математика и логика Джорджа Буля (1815—1864), являвшегося создателем современной символической логики. Подробным изучением булевских функций занимается область математической логики, которую называют *алгеброй логики* или *исчислением высказываний.*

Каждому высказыванию * ставится в соответствие значение ее *функции истинности*, равное 1, когда данное высказывание истинно, и 0, когда оно ложно. Исчисление высказываний занимается различными способами комбинаций высказываний и вычислением значений функций истинности сложных высказываний (комбинаций) по известным значениям функций истинности элементарных.

Аргументы логических функций могут принимать лишь два различных значения. Поэтому для функции любого числа переменных существует конечное число различных наборов аргументов, и логическую функцию можно задать с помощью таблицы, в которой будут перечислены все возможные наборы аргументов и соответствующие значения функции. Например, функция трех переменных $z=f\ (a,\ b, c)$ может быть задана таблицей, содержащей 8 строк (см. табл. 1); функция n переменных имеет 2^n различных наборов аргументов.

Таблица 1

Аргументы			Функция
a	b	c	z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Объем таблицы, задающей функцию, можно сократить (не менее чем вдвое), если приводить не все наборы аргументов, а только

* Точное формальное определение этого термина затруднительно и мало полезно; читатель может ограничиться интуитивным представлением о смысле этого слова; важно лишь, что должны иметь смысл фразы: *высказывание истинно* или *высказывание ложно.*

те, для которых функция равна 1 (или 0). Несмотря на это, для большого числа аргументов табличное представление может оказаться громоздким и плохо обозримым. Удобнее сложные логические функции выражать через более простые и хорошо изученные функции небольшого числа переменных при помощи формул. Этим мы и займемся в следующем параграфе.

§ 4. ЭЛЕМЕНТАРНЫЕ ЛОГИЧЕСКИЕ ФУНКЦИИ И ЛОГИЧЕСКИЕ ЭЛЕМЕНТЫ МАШИНЫ

Рассмотрим сначала основные логические функции двух аргументов.

1. Логическая функция *и*.

Значения функции *и* от двух аргументов a и b определяются таблицей 2.

Таблица 2

Аргументы		Функция *и*
a	b	
0	0	0
0	1	0
1	0	0
1	1	1

Из таблицы видно, что значения функции *и* совпадают с произведением аргументов. Поэтому ее называют также *логическим умножением*; еще одно название для этой операции, употребляемое в математической логике, — *конъюнкция*. Нетрудно определить функцию *и* для любого конечного числа аргументов: она равна 1 тогда и только тогда, когда все сомножители равны единице, как и должно быть для произведения.

Логическое умножение обозначается знаком $\wedge$. Легко показать, что эта операция подчиняется переместительному и сочетательному законам:

$$a \wedge b = b \wedge a \qquad \text{(коммутативность)},$$
$$(a \wedge b) \wedge c = a \wedge (b \wedge c) \qquad \text{(ассоциативность)}.$$

2. Логическая функция *или*.

Эта функция для двух аргументов задается таблицей 3.

Ее называют *логическим сложением* или *дизъюнкцией* и обозначают знаком $\vee$. Нужно только отметить, что логическая сумма единиц отличается от их арифметической суммы. Эта операция также подчиняется переместительному и сочетательному законам (коммутативна и ассоциативна):

$$a \vee b = b \vee a,$$
$$(a \vee b) \vee c = a \vee (b \vee c).$$

Таблица 3

Аргументы		Функция *или*
a	b	
0	0	0
0	1	1
1	0	1
1	1	1

Кроме того, для операций логического сложения и умножения справедлив распределительный *(дистрибутивный)* закон:

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c).$$

Логическая сумма нескольких слагаемых равна нулю тогда и только тогда, когда все слагаемые равны нулю.

3. Логическая функция *не*.

Функция «*не*» зависит от одного аргумента и определяется таблицей 4.

Таблица 4

Аргумент	Функция
0	1
1	0

Она обозначается чертой, поставленной над аргументом, или знаком $\neg$ перед ним, т. е. $\bar{a}$ или $\neg a$. И тот, и другой знаки читаются, как *не a*.

Подобно арифметическим операциям, логические операции имеют различное «старшинство», что отражается на порядке их выполнения в выражениях, содержащих символы различных операций: прежде всего выполняется операция *не*, затем *и* и только потом *или*. Здесь наблюдается полная аналогия с алгебраическими выражениями. При вычислении по формуле $a^2b^2+c^2$ сначала выполняются возведения в степень, затем умножение $a^2 \cdot b^2$ и только потом сложение. Аналогично в формуле $\neg a \wedge \neg b \vee \neg c$ сначала выполняются отрицания, затем конъюнкция, а потом дизъюнкция. В тех случаях, когда нужно изменить этот порядок, применяются круглые скобки, правила использования которых в логических выражениях не отличаются от тех же правил для алгебраических формул.

С помощью введенных логических функций легко заменить громоздкие таблицы простыми формулами. Так, функцию $z=f(a, b, c)$ трех аргументов, заданную таблицей 1 на стр. 52, можно выразить простой формулой

$$z = \bar{a} \wedge \bar{c} \vee a \wedge b.$$

Аналогичное представление возможно и для других логических функций. Пусть, например, логическая функция Ф трех аргументов $u=\Phi(a, b, c)$ задается наборами значений аргументов, для которых она обращается в 1:

1) $a=1$, $b=0$, $c=1$;
2) $a=1$, $b=1$, $c=0$;
3) $a=0$, $b=0$, $c=1$.

Поставим в соответствие каждому набору логическое произведение аргументов, в котором сомножитель, равный нулю, берется со знаком отрицания. Мы получим три функции:

$$F_1(a, b, c) = a \wedge \bar{b} \wedge c,$$
$$F_2(a, b, c) = a \wedge b \wedge \bar{c},$$
$$F_3(a, b, c) = \bar{a} \wedge \bar{b} \wedge c.$$

Образовав теперь логическую сумму полученных трех произведений, мы придем к формуле:

$$\Phi = F_1 \vee F_2 \vee F_3 = a \wedge \bar{b} \wedge c \vee a \wedge b \wedge \bar{c} \vee \bar{a} \wedge \bar{b} \wedge c.$$

Непосредственной проверкой легко убедиться, что полученная формула дает нам именно заданную функцию. Действительно, для любого из заданных наборов аргументов соответствующее слагаемое равно 1, так как представляет собой, по построению, произведение единиц. Поэтому и сумма равна 1. Напротив, для всех остальных наборов в каждом слагаемом найдется хотя бы один нулевой сомножитель, так что все слагаемые, а значит, и сумма равны 0.

Точно так же строится формула для функции, заданной перечислением всех наборов аргументов, для которых она равна нулю. Пусть, например, эти наборы суть

1) $a=0$, $b=1$, $c=1$;
2) $a=1$, $b=0$, $c=0$;
3) $a=0$, $b=0$, $c=0$.

Построим суммы $a \vee \bar{b} \vee \bar{c}$, $\bar{a} \vee b \vee c$, $a \vee b \vee c$ и образуем их логическое произведение

$$F = (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee b \vee c) \wedge (a \vee b \vee c).$$

Как и выше, легко убедиться, что эта функция обращается в нуль на заданных трех наборах аргументов, и только на них.

Таким образом мы убедились, что любую логическую функцию можно представить в виде комбинации функций *и*, *или* и *не*. Такая система логических функций, достаточная для получения любых других, называется *функционально полной*. Эта система не является единственной функционально полной; возможны и другие. В дальнейшем нам потребуются еще две функционально полные системы, каждая из которых состоит из единственной функции.

Одной из таких функций является логическая функция «*штрих Шеффера*». Она задается таблицей 5. Ее выражение в предыдущей системе имеет вид $F_1(a, b) = \neg(a \wedge b) = \neg a \vee \neg b$.

Другой функцией является *функция Пирса*, которая задает-

Таблица 5

Аргументы		Функция $F_1(a, b)$
a	b	
0	0	1
0	1	1
1	0	1
1	1	0

Таблица 6

Аргументы		Функция $F_2(a, b)$
a	b	
0	0	1
0	1	0
1	0	0
1	1	0

ся таблицей 6 и выражается через *и—или — не* в виде $F_2(a, b) = \neg(a \vee b) = \neg a \wedge \neg b$. Каждая из рассмотренных функций образует функционально полную систему, т. е. любая логическая функция может быть выражена через одну или через другую из них.

В качестве примера использования логических функций рассмотрим функциональную схему запоминающего элемента — *триггера*, играющего особую роль в вычислительной технике. Триггер широко используется в операционных схемах цифровых машин; он предназначен для хранения одной двоичной единицы информации, т. е. одного двоичного разряда. Мы рассмотрим три варианта триггера.

1. Триггер с раздельными входами

Функциональная схема триггера с раздельными входами показана на рисунке 18. На каждый из двух входов A и B могут подаваться входные сигналы в виде кратковременных импульсов. Наличие импульса на входе или на одном из двух выходов a, b будем считать единицей, а его отсутствие — нулем. Изображенная схема может быть описана уравнениями

$$\overline{b \vee A} = a,$$
$$\overline{a \vee B} = b.$$

Зависимость между входными и выходными величинами показана в таблице 7.

Таблица 7

Входы		Выходы	
A	B	a	b
0	0	1	0
		0	1
0	1	1	0
1	0	0	1
1	1	0	0

Из таблицы видно, что при отсутствии импульсных сигналов на обоих входах выходные величины определяются неоднозначно: уравнениям удовлетворяют как $a=0$, $b=1$, так и $a=1$, $b=0$. При действии входных импульсов на обоих входах выходы определяются однозначно, но после прекращения действия входных импульсов такая комбинация выходных величин сохраниться не может. Поэтому соответствующую комбинацию входных импульсов не используют, считая ее запрещенной.

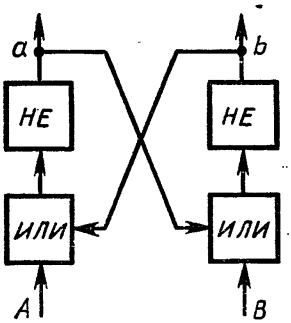


Рис. 18

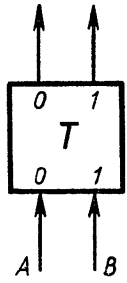


Рис. 19

Устойчивыми состояниями триггера являются $a=1$, $b=0$ и $a=0$, $b=1$. При отсутствии входных сигналов триггер будет находиться в одном из этих состояний, запоминая поданную на вход комбинацию. Одно из этих состояний будем называть *нулевым*, а второе *единичным*.

Графический символ для изображения триггера с раздельными входами показан на рисунке 19. Триггер имеет *нулевой* и *единичный входы*, а также *нулевой* и *единичный выходы*, причем обозначения вводятся так, чтобы после действия импульса на нулевой (соответственно единичный) вход триггер переходил в нулевое (единичное) состояние, при котором сигнал на нулевом (единичном) выходе есть 1, а на единичном (нулевом) — 0.

2. Триггер со счетным входом

Входной сигнал в этой схеме подается на единственный вход C (рис. 20), называемый *счетным входом*, и автоматически передается либо в точку A, либо в точку B, переводя триггер из одного состояния в противоположное. Кроме логических элементов, схема содержит элемент задержки τ, благодаря которому сигналы на входе A или B появляются после прекращения действия сигнала на входе C. Условное обозначение такого триггера в операционных схемах приведено на рисунке 21.

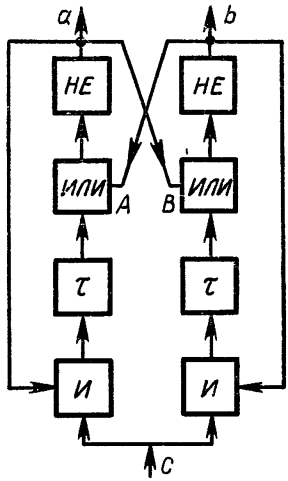


Рис. 20

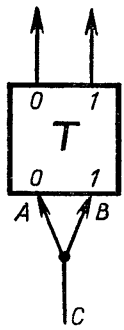


Рис. 21

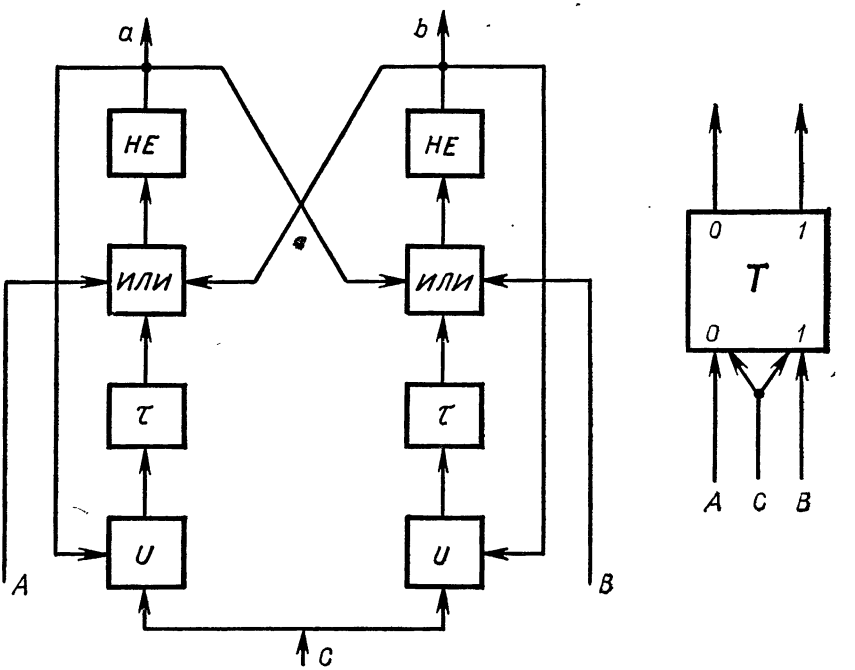


Рис. 22

Если на триггер со счетным входом подать последовательно два импульса, то первый из них переведет его в противоположное (например, из нулевого в единичное) состояние, а второй — возвратит в первоначальное. При большом числе импульсов конечное состояние определяется четностью или нечетностью их количества. Таким образом, триггер со счетным входом осуществляет счет «по модулю 2», который имеет место при работе с двоичными разрядами без переноса:

$$0+1=1+0=1; \quad 0+0=1+1=0.$$

3. Комбинированный триггер

Схема триггера, показанная на рисунке 22 (вместе с условным обозначением), сочетает в себе свойства обеих предыдущих схем. Она имеет как раздельные, так и счетные входы.

§ 5. ФИЗИЧЕСКИЕ ЭЛЕМЕНТЫ, ИСПОЛЬЗУЕМЫЕ ДЛЯ КОНСТРУИРОВАНИЯ ЛОГИЧЕСКИХ

В предыдущем параграфе мы познакомились с понятием логических элементов машины и с некоторыми примерами их использования. Теперь следовало бы перейти к конструированию

самих этих элементов. Однако предварительно мы должны познакомиться с физическими элементами, которые здесь используются. Им и посвящен настоящий параграф.

Самым простым электрическим элементом с двумя устойчивыми состояниями является *переключатель*, который может быть замкнут или разомкнут. В первых цифровых вычислительных машинах в качестве переключателей использовались *электромеханические реле*. Но их быстродействие невелико, так как подвижные механические части весьма инерционны. Кроме того, реле имеют ограниченное число надежных срабатываний и для обеспечения правильной работы должны часто заменяться.

Довольно быстро на смену электромеханическим переключателям пришли более быстрые и надежные *электронные*, построенные на основе вакуумных электронных ламп.

Простейшая электронная лампа — вакуумный *диод* — имеет два электрода: *анод*, сделанный из фольги, и стержневой *катод*, который нагревается нитью накала. Если анод лампы присоединить к положительному полюсу источника, а катод — к отрицательному, то свободные электроны, вылетающие из катода, устремятся к аноду и создадут ток в цепи. Если же изменить полярность включения источника, то электрическое поле будет отталкивать электроны от анода и тока в цепи не будет. Таким образом, диод проводит ток лишь в одном направлении и его можно рассматривать как переключатель: если к аноду приложен плюс, а к катоду минус, то он замкнут, если наоборот — разомкнут.

Если между анодом и катодом поместить еще один электрод — *сетку* (рис. 23), то получится трехэлектродная лампа — *триод*. Когда на сетку подается положительное относительно катода напряжение, то она притягивает электроны, чем усиливает анодный ток. Сетка ближе к катоду, чем анод, и небольшое изменение напряжения на сетке значительно увеличивает анодный ток; триод усиливает сигнал, поданный на сетку. Если же на сетку подается отрицательное напряжение, то сетка будет отталкивать электроны катода и ток в анодной цепи будет отсутствовать, несмотря на приложенное к аноду положительное напряжение.

Триод может быть использован в качестве переключательного элемента благодаря наличию двух режимов его работы:

а) Режим насыщения. При некотором положительном напряжении на сетке все электроны, вылетающие из катода и образующие так называемый пространственный заряд, будут притягиваться к аноду; дальнейшее увеличение напряжения на сетке не приведет уже к увеличению анодного тока. Это *состояние насыщения* соответствует замыканию переключателя;

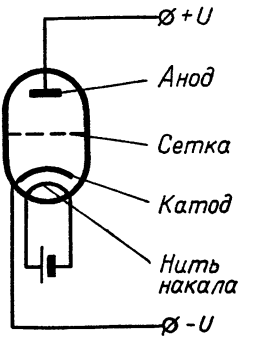


Рис. 23

б) Режим отсечки. При некотором отрицательном напряжении на сетке анодный ток полностью прекращается, «отсекается». Это состояние отсечки соответствует размыканию переключателя.

Электронные переключатели на лампах обладают высоким быстродействием и могут обеспечить большую скорость переключения. Их недостатки: большое потребление энергии и не очень высокая надежность — свойство катода испускать электроны со временем ухудшается, и лампа выходит из строя.

Машины второго поколения вместо электронных ламп стали использовать полупроводниковые диоды и триоды *(транзисторы)*. В этих приборах ток создается направленным движением свободных носителей заряда внутри кристаллической решетки полупроводникового материала.

Германий или *кремний* — обычно используемые полупроводниковые вещества — в кристаллическом состоянии и чистом виде (без примесей) являются изоляторами. Но небольшое количество специальных примесей вызывает появление в кристаллах свободных электрических частиц, способных перемещаться под действием электрического поля, т. е. создающих проводимость.

И германий и кремний четырехвалентны, и имеют четыре валентных электрона, находящихся на внешних орбитах. Если добавить в кристалл некоторое количество пятивалентного элемента, например мышьяка, то атомы последнего займут места некоторых атомов исходного элемента. Пятые валентные электроны этих атомов слабее связаны с ядром, чем остальные. Благодаря этому в кристалле появится некоторое количество свободных электронов, носителей отрицательного заряда, способных перемещаться под влиянием внешнего электрического поля.

Напротив, если в кристалл четырехвалентного вещества ввести примесь трехвалентного, например индия, то в точке, где атом исходного вещества заменится атомом индия, образуется недостаток электронов, «дырка», равносильная наличию свободного положительного заряда. Под действием электрического поля «дырка» будет перемещаться в сторону, противоположную реальному перемещению электронов, ее заполняющих. Это явление называют *дырочной* проводимостью.

Таким образом, добавлением соответствующих примесей можно создавать кристаллы с электронной проводимостью (полупроводники *n-типа*) или с дырочной проводимостью (полупроводники *p-типа*).

Соединим полупроводники с разными типами проводимости и рассмотрим *электронно-дырочный переход* (*p—n-переход*) на границе (рис. 24). Вблизи границы происходит взаимный обмен (диффузия) свободных электронов и дырок из одной области в другую, в результате чего вблизи перехода в *n*-области создается положительный, а в *p*-области отрицательный заряды. Создаваемое этими зарядами поле будет препятствовать дальнейшей диффузии. Говорят, что здесь создается *потенциальный барьер*.

Рассмотрим, что произойдет при подключении к такому соединению внешнего источника э. д. с. Пусть плюс источника подключен к *p*-области, а минус — к *n*-области (рис. 25). Тогда электроны и дырки смогут под действием внешнего электрического тока преодолеть потенциальный барьер и, двигаясь навстречу через переход, создадут значительный *прямой ток*. Если же плюс источника подключен к *n*-области, а минус — к *p*-области (рис. 26), то внешнее поле будет оттягивать электроны и дырки от границы и увеличивать потенциальный барьер. *Обратный ток* через переход будет в этом случае очень мал.

Отсюда ясно, что описанная система двух полупроводниковых элементов является *диодом*, проводящим ток в одном направлении, причем *p*-область играет роль катода, а *n*-область — анода. Вольт-амперная характеристика (зависимость тока через *p*—*n*-переход от приложенного напряжения) диода показана на рисунке 27. При малых положительных напряжениях (положительное смещение диода), пока потенциальный барьер не преодолевается внешним полем, ток невелик; начиная с некоторого значения ток резко возрастает — ветвь вольт-амперной характеристики становится почти вертикальной, сопротивление — очень малым. При отрицательном напряжении (изменении полярности источника, отрицательное смещение) ток через переход $i_{обр}$ очень мал, соответственно сопротивление $r_{обр}$ велико. Дальнейшее увеличение напряжения по абсолютной вели-

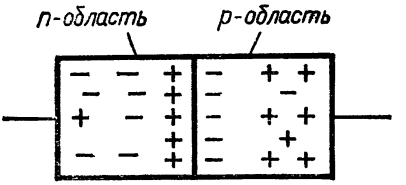


Рис. 24

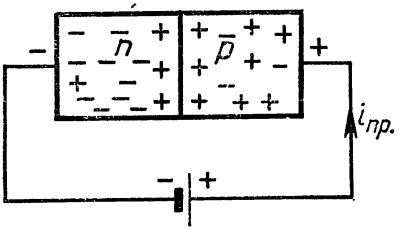


Рис. 25

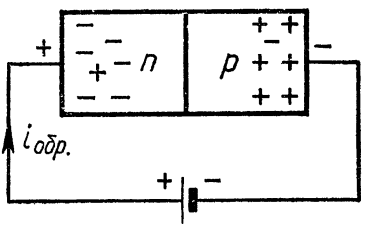


Рис. 26

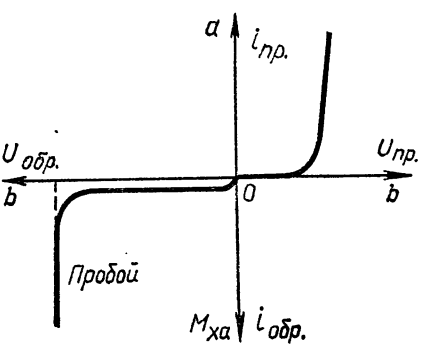


Рис. 27

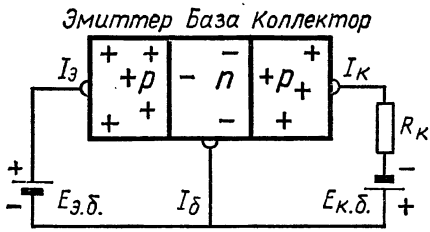


Рис. 28

чине может привести к разрушению структуры полупроводника (*пробой*); при этом ток лавинообразно увеличивается.

Способность полупроводникового диода проводить ток в одном направлении позволяет использовать его в качестве электронного ключа. Правда, его вольт-амперная характеристика отличается от характеристики идеального ключа, но на практике этим расхождением обычно можно пренебречь.

Другой тип полупроводниковых приборов — *транзистор* — представляет собой сочетание двух p — n-переходов. Этот аналог вакуумного триода имеет две p-области и одну n-область и носит название p — n — p-транзистора. Схематически он изображен на рисунке 28. При этом n-область называется *базой* прибора, одна из p-областей — *эмиттером*, другая — *коллектором*.

Переход эмиттер — база смещен напряжением $E_{э.б.}$ в прямом направлении, переход база — коллектор — в обратном направлении напряжением $E_{к.б.}$. В цепь коллектора включена нагрузка — резистор $R_к$. За счет прямого смещения через переход э — б протекает прямой ток и дырки переходят в базу. Здесь они диффундируют к переходу б — к и под действием поля внешнего источника $E_{к.б.}$ переходят в коллектор, в результате чего по нагрузке протекает ток $I_к$. Ток $I_к$ в цепи коллектора почти равен току $I_э$ эмиттера. Коэффициент усиления по току, т. е. отношение изменения тока коллектора к вызвавшему его изменению тока эмиттера, $\alpha=\Delta I_к/\Delta I_э$ есть величина порядка 0,9÷0,99. Изменяя напряжение $E_{э.б.}$, можно изменять количество дырок, «впрыскиваемых» в область коллектора, и тем изменять ток в нагрузке $R_к$.

Итак, полупроводниковый p — n — p-транзистор является аналогом вакуумного триода, причем эмиттер играет роль катода, коллектор — роль анода, а база — роль сетки. Подобным же образом работает и n — p — n-транзистор, имеющий две n-области и одну p-область. Схема его включения отличается от рассмотренной лишь полярностью источников питания. Условные обозначения диода, p — n — p- и n — p — n-транзисторов показаны на рисунке 29.

Транзистор, включенный по схеме рисунка 30, *а* не может усиливать ток, поскольку $\alpha<1$, но его можно использовать как усилитель напряжения. Коэффициент усиления по напряжению есть отношение напряжения $u_н$ на нагрузке к входному напряжению $E_{э.б.}$, $k_н=u_н/E_{э.б.}$. Так как $u_н=I_к R_к$, а $E_{э.б.}=I_э R_{э.б.}$, где $R_{э.б.}$ — входное сопротивление транзистора, т. е. сопротивление перехода э — б, смещенного в прямом направлении, то

$$k_н=\frac{I_к R_к}{I_э R_{э.б.}}=\alpha\frac{R_к}{R_{э.б.}}.$$

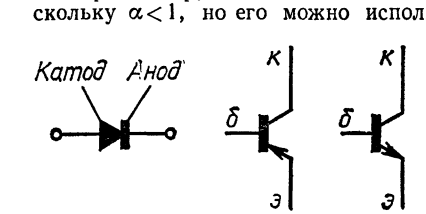


Рис. 29

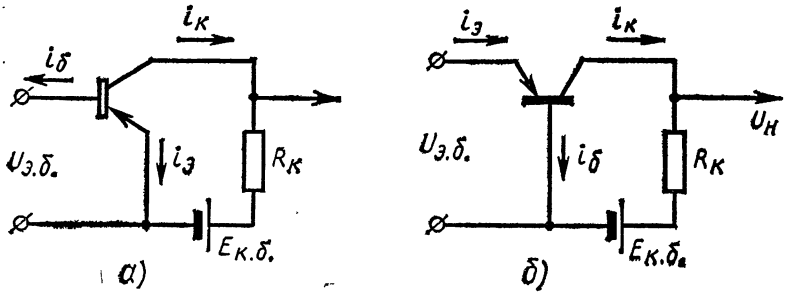


Рис. 30

Выбирая сопротивление нагрузки нужным образом большее $R_{э.б}$, можно получить достаточно большое значение коэффициента усиления по напряжению $k_н$.

Чаще, чем схема рисунка 30, *а*, называемая схемой включения *с общей базой*, используется другая схема включения транзистора, которую называют схемой *с общим эмиттером*. Она показана на рисунке 30, *б*. Здесь коэффициент усиления по току равен $k_i=\Delta I_к/\Delta I_б$. По закону Кирхгофа $i_э=i_б+i_к$ и $\Delta I_б=\Delta I_э-\Delta I_к=(1-\alpha)I_э$. Поэтому

$$k_i=\Delta I_к/\Delta I_б=\alpha\Delta I_э/(1-\alpha)\,\Delta I_э=\alpha/(1-\alpha)\approx 1/(1-\alpha).$$

Из $\alpha\approx 0{,}9\div 0{,}99$ следует $k_i\approx 10\div 100$. Следовательно, такая схема позволяет получить усиление по току. Нетрудно показать, что коэффициент усиления по напряжению остается прежним.

Основой практического применения полупроводниковых триодов является схема усилителя на транзисторах. Такая схема изображена на рисунке 31. В ней используется источник питания, отрицательный полюс которого подключен через нагрузку $R_к$ к коллектору, а положительный заземлен. Входным сигналом является напряжение, подаваемое между базой и заземленным эмиттером, выходное напряжение снимается между коллектором и эмиттером.

Подобно полупроводниковым диодам, транзисторы, как и вакуумные триоды, могут быть применены в качестве электронных переключателей. Для этого используются режимы насыщения и отсечки, вполне аналогичные соответствующим режимам вакуумных триодов.

Режим отсечки соответствует разомкнутому ключу, т. е. случаю, когда триод заперт и не проводит тока. В вакуумном триоде это достигается подачей на сетку отрицательного напряжения. В полупроводниковом триоде необходимо «отсечь» ток через переход эмиттер — база, для этого нужно сместить его входным напряжением в обратном направлении, в результате чего прекратится ввод дырок в базу.

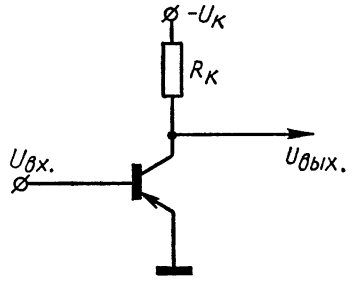


Рис. 31

Аналогично режим насыщения соответствует замкнутому ключу, т. е. случаю, когда триод полностью отперт

и проводит ток. В вакуумном триоде это достигается подачей определенного положительного напряжения на сетку. В полупроводниковом триоде надо увеличить напряжение, смещающее переход эмиттер — база в прямом направлении. При насыщении дырок в базе окажется настолько много, что электрическое поле будет не в состоянии все их втянуть в коллектор: рост тока коллектора прекратится.

Быстродействие транзисторного переключателя различно в зависимости от направления переключения и зависит от тока $I_б$, подаваемого в базу, но во всех случаях достаточно велико.

Уже в начале шестидесятых годов ламповые диоды и триоды были почти полностью вытеснены полупроводниковыми приборами. Последние потребляют во много раз меньшую мощность, занимают примерно в 100 раз меньший объем и значительно надежнее, чем ламповые элементы. Поэтому при рассмотрении логических элементов мы будем исходить из использования полупроводниковых схем.

§ 6. ЛОГИЧЕСКИЕ ЭЛЕМЕНТЫ ЦИФРОВЫХ МАШИН

Сформулируем основные требования к логическим элементам цифровых вычислительных машин.

1. *Высокое быстродействие.* Суть этого требования вполне ясна, так как именно быстродействие элементов определяет в конечном счете быстродействие вычислительной машины. Для логических элементов существенную роль в этом отношении играет время переключения.

2. Достаточно *большая нагрузочная способность.* Нагрузочная способность определяет число других элементов, которые можно подключать к выходу данного элемента, сохраняя нормальные условия его работы.

3. Высокая *помехоустойчивость.* Логические элементы должны быть устойчивы к помехам, которые всегда присутствуют в электрических цепях,— ложные сигналы, вызванные наводками от параллельных цепей, шумы вследствие случайных флуктуаций зарядов в физических элементах и т. п.

4. *Малая потребляемая мощность.* Потребляемая элементами мощность рассеивается, главным образом, в виде тепла. Большое тепловыделение нарушает режимы работы схем и ограничивает «плотность упаковки» элементов, что ведет к увеличению размеров элементов, а значит, и машины в целом.

5. Как можно *меньшее количество физических элементов.* Схемы, требующие меньшего количества физических элементов, оказываются эффективнее как с экономической точки зрения, так и с точки зрения надежности и долговечности.

В § 4 указывалось, что для выполнения любых логических функций достаточно иметь элементы, осуществляющие функции, составляющие некоторую функционально полную систему, например

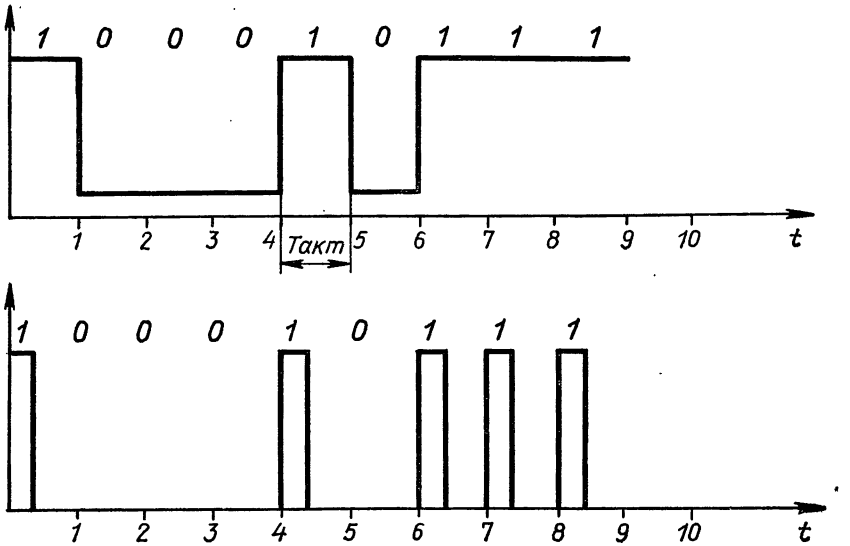


Рис. 32

функции *и*, *или*, *не*. На практике, однако, нередко бывает выгодно использовать и другие схемы.

Физическими аналогами двоичных цифр в цифровых машинах служат электрические сигналы в соответствующих точках схем, способные принимать два хорошо различимых значения. В схемах цифровых устройств эти сигналы изменяются и воспринимаются не непрерывно, а в некоторые дискретные моменты времени. Временной интервал между этими моментами называется *тактом* *. Цифровые устройства обычно содержат специальный блок, вырабатывающий синхронизирующие сигналы, следующие через равные интервалы времени и отмечающие указанные моменты.

Понятие *такта* позволяет разграничить два способа представления информации в цифровой машине: *потенциальный* и *импульсный*. При потенциальном способе цифрам 0 и 1 соответствуют высокий и низкий уровень напряжения в данной точке; потенциальным называется сигнал, меняющий свою величину о д и н р а з за такт. При импульсном способе цифрам 0 и 1 соответствует, например, отсутствие и наличие импульса; импульсом называется сигнал, изменяющий свою величину д в а ж д ы в течение такта. На рисунке 32 показано изменение напряжения при потенциальном и импульсном изображениях одного и того же слова 100010111.

Схема элемента *или* с двумя входами приведена на рисунке 33. Логическими переменными являются входные сигналы $u^1_{вх}$ и $u^2_{вх}$,

* Обращаем внимание читателя, что приведенное определение согласуется с определением такта в § 3 этой главы.

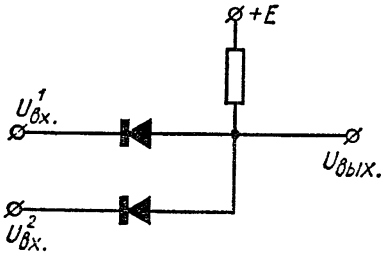


Рис. 33

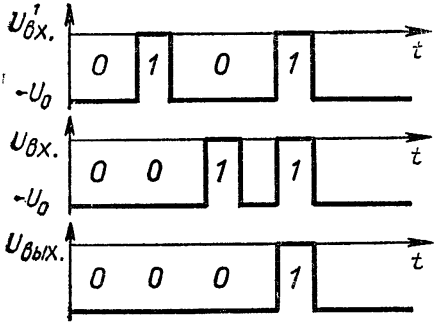


Рис. 34

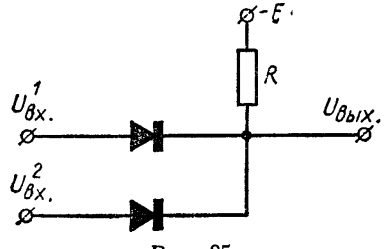


Рис. 35

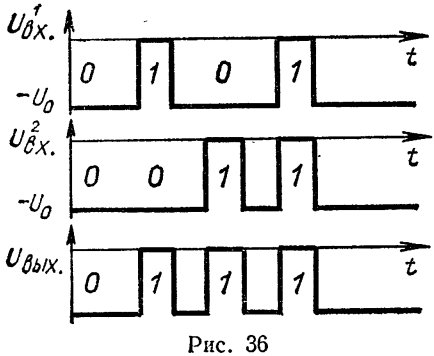


Рис. 36

имеющие вид прямоугольных импульсов положительной полярности относительно некоторого уровня — u_0 (та же схема применима и в случае потенциальных сигналов). Наличие импульса изображает 1, отсутствие—0.

Работа схемы иллюстрируется временно́й диаграммой (рис. 34). При подаче хотя бы на один вход схемы положительного импульса (его амплитуда выбрана равной $|u_0|$) замыкается цепь тока через диод и резистор R. Так как сопротивление R обычно много больше прямого сопротивления диода, то практически все напряжение падает на резисторе и с выхода снимается импульс амплитуды $|u_0|$, т. е. выдается сигнал 1. При отсутствии входных импульсов $u_{вых}$ примет значение $-u_0$ (ибо обычно $|u_0|<|-E|$), что соответствует сигналу 0. Такую схему называют *собирательной*.

Аналогично строится и *схема совпадения* для осуществления операции *и* (рис. 35). При одновременной подаче импульсов, означающих 1, на оба входа, оба диода запираются и на выходе схемы появляется импульс амплитуды $|u_0|$, что соответствует сигналу 1. Если хотя бы на одном входе импульс отсутствует (0), то соответствующий диод отпирается, и образуется цепь из резистора и диода с очень малым прямым сопротивлением. Потенциал на выходе падает тогда до— u_0, что соответствует сигналу 0 (см. временную диаграмму на рис. 36).

Функция *не* требует для своей реализации транзистора. Эта схема, называемая *инвертором*, показана на рисунке 37. При

отсутствии входных импульсов выходное напряжение схемы равно $u_{вых} = -E_к$, так как транзистор заперт положительным смещением $+E_{см}$. Это напряжение соответствует 1 на выходе. При поступлении на вход соответствующего 1 отрицательного импульса, амплитуда которого достаточна для преодоления смещения, транзистор открывается и напряжение на его выходе становится близким к нулю, что соответствует 0 на выходе схемы.

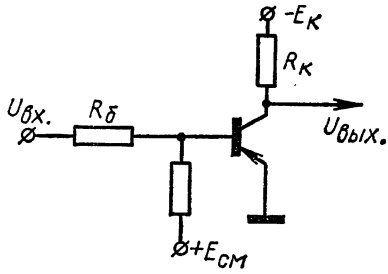


Рис. 37

Рассмотренные схемы просты, обладают нужным быстродействием и экономичны, однако они неспособны усиливать сигналы. Поэтому нагрузочная способность таких схем мала. Этот недостаток стремятся исправить, дополняя схемы транзисторными усилителями. В таком случае выгоднее уже реализовать более сложные функции, например функцию Шеффера или Пирса.

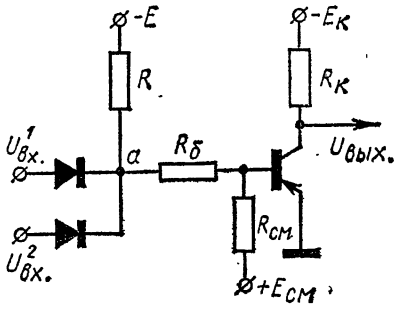


Рис. 38

В качестве примера разберем диодно-транзисторную схему, реализующую функцию Пирса (рис. 38). Как известно из § 4, выход этой функции должен быть нулем во всех случаях, кроме того, когда нулями являются оба входа.

Действительно, при отсутствии входных импульсов в точке a схемы установится отрицательное напряжение $-u_a$. База триода будет находиться под отрицательным напряжением относительно эмиттера (при $|u_a| > +E_{см}$), триод будет открыт и через сопротивление $R_б$ потечет прямой ток. Параметры схемы рассчитывают так, чтобы триод находился в режиме насыщения. Тогда на выходе установится напряжение, близкое к нулевому, что соответствует 1. Если на вход подан хотя бы один положительный импульс, то потенциал точки a возрастет до нуля и транзистор закроется положительным смещением $E_{см}$. На выходе усилителя установится напряжение $-u_0$, соответствующее сигналу 0.

Новая схема содержит усилитель и потому много лучше по нагрузочной способности. Тем не менее она проигрывает в быстродействии, содержит гораздо больше физических элементов и требует большей мощности. Для устранения этих недостатков используют режим далекий от насыщения.

Работу транзисторов в режиме, далеком от режима насыщения, обеспечивают *схемы с переключателями токов*. Принцип действия

таких схем показан на рисунке 39. В зависимости от того, какой ключ (I или II) замкнут, ток от источника поступает либо в левое, либо в правое плечо схемы. При поступлении тока в левое плечо на резисторе $R_{лев}$ создается падение напряжения. Эту ситуацию принимаем на состояние, изображающее 0; падение напряжения на резисторе $R_{пр}$ — за изображение 1. В качестве ключей I и II используются транзисторы, находящиеся либо в режиме отсечки, либо в состоянии, когда транзистор открыт небольшим входным сигналом и далек от режима насыщения (см. например, рисунок 40).

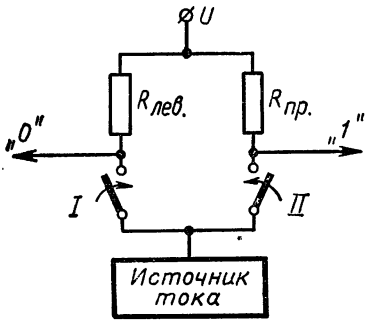


Рис. 39

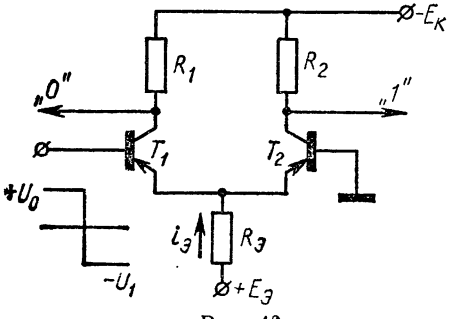


Рис. 40

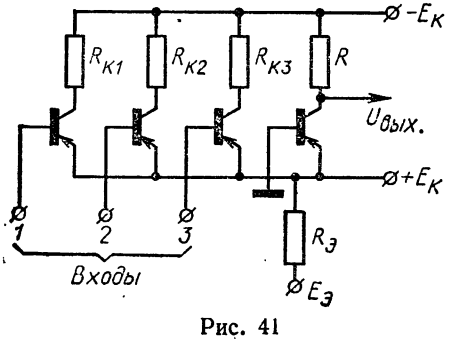


Рис. 41

Величина общего эмиттерного сопротивления $R_э$ двух транзисторов выбирается достаточно большой. База транзистора T_2 заземлена, а на базу T_1 подается управляющее напряжение; сигналы $+u_0$ или $-u_1$, симметричные относительно потенциала земли. Если на базу T_1 подается напряжение u_1, то транзистор T_1 открыт и через него протекает ток, определяемый сопротивлениями резисторов и внутренним сопротивлением триода. Так как $R_э \gg R_1 + (r_{эб} + r_{эк})$, то $i_э \approx E_э / R_э$. Транзистор T_2 закрыт.

Если потенциал базы T_1 положителен, то T_1 закрыт, а эмиттерный ток протекает через T_2. Таким образом, симметричным относительно нуля сигналом на базу T_1 можно осуществить переключение тока либо в ветвь T_1, либо в ветвь T_2. Минимум переключательного напряжения определяется тем небольшим напряжением, на которое эмиттер должен быть положительнее базы, чтобы транзистор открылся. При этом он остается далеко от режима насыщения.

Наибольшее распространение в современных машинах

получили логические элементы на основе диодно-транзисторных схем, а также схем с переключателями тока. Схема с переключателем, реализующая логическую операцию *и* на три входа, приведена на рисунке 41. Она достаточно проста, и в ее работе читатель сможет разобраться самостоятельно.

В заключение приведем принципиальную схему триггера с раздельными входами, о котором шла речь в предыдущем параграфе (рис. 42). Здесь роль инвертора (*не*) играет транзистор, роль собирательных схем (*или*) — входные диоды и цепь коллектор T_1 — база T_2. Временна́я диаграмма работы триггера приведена на рисунке 43.

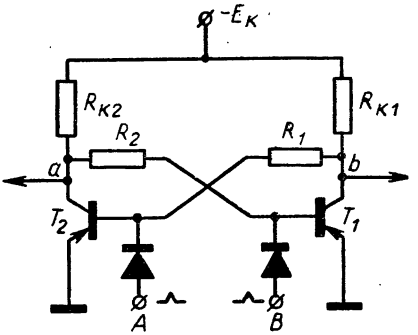


Рис. 42

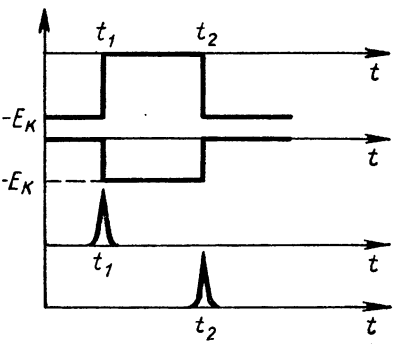


Рис. 43

Пусть транзистор T_1 находится в открытом состоянии. Тогда через него проходит значительный ток и напряжение в точке *b* близко к нулевому уровню. Через резистор R_1 это напряжение передается на базу транзистора T_2 и поддерживает его в запертом состоянии; напряжение в точке *a*, следовательно, равно — $E_к$. Условимся это состояние триггера считать единичным.

Допустим, что в момент t_1 на базу T_1 подан положительный импульс (вход *B*). Импульс запрет транзистор T_1, его коллекторный ток уменьшится и потенциал коллектора (точки *b*) упадет от нуля до $-E_к$. Благодаря обратной связи низкий потенциал через резистор R_1 передается на базу транзистора T_2 и отпирает его; через резистор R_2 начинает проходить большой коллекторный ток, и напряжение в точке *a* повышается от $-E_к$ до нуля. Триггер переходит в новое, нулевое состояние и будет находиться в нем до тех пор, пока импульс на входе *A* не переведет его снова в единичное.

§ 7. ПРЕДСТАВЛЕНИЕ ЧИСЕЛ И КОМАНД В МАШИНЕ

Прежде чем переходить к рассмотрению операционных схем, используемых в различных устройствах цифровой вычислительной машины, мы подробнее познакомимся с тем, как представляются в машине в виде машинных слов команды и числа. Конечно, отдель-

ные детали такого представления выглядят в различных машинах по-разному, но мы остановимся не на деталях, а на общих принципах, которые во всех машинах одни и те же.

Начнем с рассмотрения команд. Как уже говорилось в § 2, машинное слово, изображающее команду, делится на *операционную* и *адресную* части. Разряды, отведенные для записи операционной части команды, содержат только записанный в двоичной системе *код операции*.

Число разрядов, отводимых для операционной части команды на машинах различных типов, различно. Оно зависит от общего числа разрядов в ячейке памяти, от адресности машины, а также и от числа различных элементарных операций, входящих в систему команд данной машины. Изменяется также и число разрядов, отводимых для записи адресной части и для записи каждого адреса в отдельности.

Рассмотрим структуру машинного слова при записи команды для нескольких конкретных вычислительных машин различной адресности.

Трехадресные вычислительные машины БЭСМ-4, М-220, М-222 имеют одинаковую ячейку памяти, состоящую из 45 двоичных разрядов, которые нумеруются справа налево (другие машины второго поколения имеют более длинную ячейку памяти). Первые 36 разрядов справа отведены для адресной части, в которой записываются три адреса по 12 разрядов: в разрядах 36—25 — I адрес, в разрядах 24—13 — II адрес и в разрядах 12—1—III адрес (рис. 44). Таким образом, каждый адрес есть двенадцатиразрядное двоичное, или, что то же самое, четырехразрядное восьмеричное число. Этого достаточно для прямой адресации $2^{12}=4096$ ячеек памяти. Именно такой объем памяти и имела машина первого поколения М-20, по образцу которой сделаны и рассматриваемые машины второго поколения.

Для указания большего адреса в этих машинах применяется *присоединенная адресация*: в машине имеется специальный *регистр приращений*, в котором записываются дополнительные три разряда (триада) для каждого адреса, которые присоединяются к двенадцатиразрядному адресу в команде в качестве старших трех разрядов, образуя, таким образом, пятнадцатиразрядный адрес. С его помощью можно адресовать уже 2^{15} ячеек. Можно представить дело так: память машины состоит из нескольких *кубов* по 4096 ячеек каждый. Адрес в команде означает адрес некоторой ячейки внутри куба,

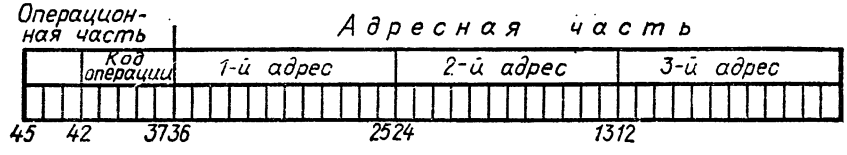


Рис. 44

а содержимое соответствующей триады в регистре приращений — номер куба, из которого эта ячейка берется.

Левые 9 разрядов (45—37) отводятся для операционной части команды. Из них разряды 45—43 используются для работы с индекс-регистром * и только разряды 42—37 содержат код операции. Следовательно, код операции здесь является шестизарядным двоичным или двухразрядным восьмеричным числом, что позволяет иметь $2^6=64$ различных кода и соответственно столько же различных элементарных операций машины.

Отметим, кстати, что в некоторых случаях бывает удобно временно отказаться от адресного принципа и использовать адрес в команде не как адрес ячейки памяти, а как некоторое условное число. В этом случае можно объединять одним и тем же кодом несколько элементарных операций, которые будут различаться условным числом, записываемым в определенном адресе команды.

Эти принципы распределения разрядов машинного слова при записи команды, собственно, в достаточной степени универсальны и используются во всех машинах. Заметим только, что при уменьшении числа адресов в команде можно удлинять адреса и отводить больше разрядов на операционную часть. Кроме того, в машине может быть несколько индекс-регистров и наряду с признаками работы с индекс-регистром в команде может быть указан его номер.

Ячейка памяти двухадресной машины Минск-22 имеет 37 разрядов (0—36), которые нумеруются слева направо. Нулевой разряд является знаковым. Разряды 0—6 отводятся для записи кода операции, который, следовательно, может принимать значения от —77 до +77. Шесть разрядов (7—12) отводятся для записи номера индекс-регистра. Правые 24 разряда составляют адресную часть и содержат два адреса, по 12 разрядов каждый (разряды 13—24 и 25—36). Это дает возможность прямой адресации 2^{12} ячеек памяти. Оперативная память машины Минск-22 имеет два куба по $2^{12}=4096$ ячеек памяти. Адрес в команде указывает номер ячейки в кубе; номер куба указывается вместе с номером индекс-регистра: 7 и 8 разряды, которые могут содержать 0 или 1, означают номера кубов, соответствующих ячейкам первого и второго адреса команды. Ясно, что это эквивалентно присоединенной адресации, о которой мы говорили выше в связи с трехадресными машинами.

Говоря об одноадресных машинах, мы остановимся на машине БЭСМ-6. Ячейка памяти этой машины имеет 48 разрядов. Однако, поскольку в команде указывается только один адрес, то ее можно сделать короткой. В машине БЭСМ-6 команда сделана 24-разрядной и поэтому в ячейке памяти записываются одновременно две команды; одна занимает разряды 48—25, а вторая — разряды 24—1. При вы-

---

* Об индексных регистрах и работе с ними мы будем говорить в § 5 гл. V. Строго говоря, работа с индексными регистрами относится не к операционной, а к адресной части команды; тем не менее обычно к адресной части команды относят лишь непосредственно указанные в ней адреса. Мы будем поступать так же.

борке из памяти по счетчику команд в устройство управления вызываются обе команды одновременно. Сначала расшифровывается и выполняется команда, записанная слева (разряды 48—25), а затем справа (разряды 24—1).

Машина БЭСМ-6 имеет команды двух различных структур — *короткоадресные* (I структура) и *длинноадресные* (II структура). В длинноадресных командах II структуры правые 15 разрядов (1—15) отводятся для записи адреса, так что в них можно непосредственно адресовать 2^{15} ячеек оперативной памяти. Следующие четыре разряда (16—19) отводятся для записи кода операции. Их может быть, следовательно, $2^4=16$, но в машине не все возможные коды фактически используются. 20-й разряд содержит признак структуры команды. Для длинноадресных команд этот признак равен 1. Обычно его относят к коду операции и объединяют в одну восьмеричную цифру с 19 разрядом, так что первая цифра кода в команде II структуры всегда 2 или 3. Левые разряды 21—24 содержат номер индекс-регистра, с которым команда работает.

Короткоадресные команды I структуры отводят для записи адреса только 12 правых разрядов (1—12). 20-й разряд, содержащий признак структуры команды, равен 0. Для кода операции в команде I структуры отводится 6 разрядов — разряды 13—18, так что здесь код операции есть двузначное восьмеричное число; обычно ему приписывают впереди 0 из 20-го разряда и пишут, например 037 или 024. Разряды 21—24, как и в предыдущем случае, используются для записи номера индекс-регистра, а 19-й разряд — для удлинения адреса: если в нем стоит 0, то перед двенадцатиразрядным адресом, стоящим в команде, приписывается триада 000, а если в нем стоит 1, то приписывается триада 111. Таким образом, короткоадресным командам непосредственно доступны две группы по 4096 ячеек памяти с первой цифрой 0 или 7. Доступ в остальные ячейки можно получить только через индекс-регистр. Описанное распределение разрядов показано на рисунке 45.

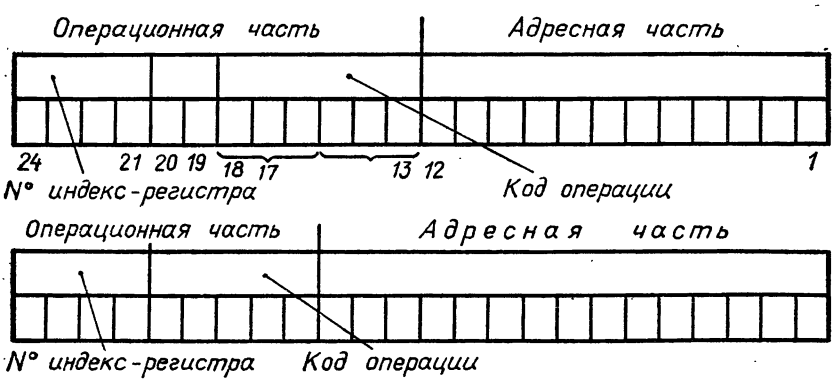


Рис. 45

Рассмотрим теперь представление чисел в ячейке памяти машины. Первые электронные вычислительные машины, и у нас и за рубежом, были машинами с фиксированной запятой. Это означает, что положение запятой, отделяющей целую часть числа от дробной, было фиксировано в определенном месте ячейки памяти, между определенными разрядами. Обычно запятую помещали перед первой значащей цифрой числа, так что в ячейке памяти машины записывалось число, меньшее единицы. Определенное число разрядов в ячейке позволяло сохранять для всех чисел одинаковую а б с о л ю т н у ю точность, но относительные погрешности оказывались в этом случае для разных чисел весьма различными.

Использование фиксированной запятой облегчало конструирование и изготовление машины, все разряды в этом случае (кроме з н а к о в о г о) работают одинаково. Однако необходимость выбирать масштабы для рассматриваемых переменных и следить за тем, чтобы не только исходные, но и все промежуточные и окончательные результаты разумным образом помещались в выбранной разрядной сетке, вызывала серьезные затруднения. Поэтому от фиксированной запятой довольно быстро отказались и перешли к форме записи *с плавающей запятой*.

Числа в форме *с плавающей запятой* записываются в виде

$$N = N_0 \cdot 10^p, \qquad (1)$$

где 10 означает основание системы счисления; следовательно, эта запись имеет смысл в любой системе счисления. Число N_0 называют *мантиссой* числа N, а p — его *порядком*. Сохраняя у всех чисел одинаковое число значащих цифр мантиссы, мы сохраняем для них тем самым постоянную о т н о с и т е л ь н у ю точность, что в большинстве случаев более важно и более удобно, нежели сохранение абсолютной точности.

Запись числа в виде (1) является, очевидно, неоднозначной. Действительно,

$$12{,}57 = 1{,}257 \cdot 10^1 = 0{,}1257 \cdot 10^2 = 0{,}01257 \cdot 10^3 = \ldots$$

Из всех записей вида (1) только одна удовлетворяет условию

$$1 > |N_0| \geqslant 0{,}1, \qquad (2)$$

где $0{,}1 = 10^{-1}$, а 10 — основание системы счисления, так что запись (2) снова имеет один и тот же смысл в любой системе счисления: мантисса числа меньше единицы, но ее первая цифра отлична от нуля. Число, представленное в виде (1) с мантиссой, удовлетворяющей условию (2), называют *нормализованным*.

В машинах с плавающей запятой число записывается в виде (1), и машинное слово, представляющее число в ячейке памяти, содержит мантиссу и порядок этого числа, причем обычно это число является нормализованным, т. е. мантисса удовлетворяет условию (2), хотя в отдельных случаях это может быть и не так. В соответствии с этим ячейка памяти делится на две части: в одной записывает-

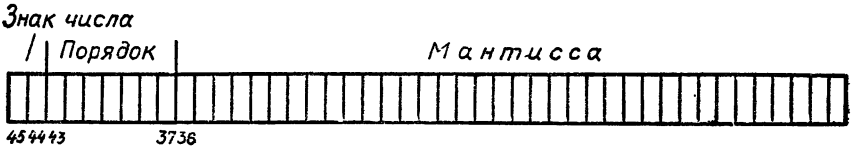


Рис. 46

ся порядок, а в другой — мантисса. Нередко это деление ячейки совпадает с делением ее на операционную и адресную части при записи команды.

Так обстоит дело, например, в трехадресных машинах, о которых мы говорили в начале этого параграфа. Правые 36 разрядов, образующие адресную часть слова при записи команды, при записи числа содержат мантиссу. Эта длина соответствует примерно 10—11 десятичным знакам, что для большинства задач можно считать приемлемой точностью. Разряды 45—37, являющиеся в команде операционной частью, при записи числа образуют п о р я д к о в у ю часть. При этом 45-й разряд фактически при записи числа не используется— в некоторых случаях в нем записывается специальная метка; 44-й разряд содержит *знак числа*: 0 изображает знак плюс, а 1 — знак минус. Разряды 43—37 отводятся для записи *условного порядка числа* (рис. 46).

Условный порядок числа определяется следующим образом. Двоичный порядок p записывается как двузначное восьмеричное число (шесть разрядов). *Условный порядок* (иногда говорят *машинный порядок*) p' по определению равен:

$$p' = 100_8 + p. \qquad (3)$$

Так как для записи условного порядка отводится 7 разрядов, то вообще $177_8 \geqslant p' \geqslant 0$. На самом деле, нулевой условный порядок применяется только для записи нуля: если в результате арифметических действий получается число с нулевым условным порядком, то его мантисса также сбрасывается и принимается равной нулю. Поэтому для чисел, отличных от нуля, имеем $177_8 \geqslant p' \geqslant 1$, откуда для истинного порядка получаем $77_8 \geqslant p \geqslant -77_8$. Это неравенство определяет интервал допустимых нормализованных чисел

$$2^{64} - 1 \geqslant |N| \geqslant 2^{-64}.$$

Числа, выходящие за пределы этого диапазона, в машине записаны быть не могут. Числа, меньшие 2^{-64}, принимаются равными нулю *(машинный нуль)*. При получении больших чисел предусмотрена остановка по переполнению разрядной сетки, так называемый *аварийный останов* (сокращенно *авост*).

Сравнив внимательно величину p истинного порядка и запись условного порядка в ячейке памяти, мы заметим, что при $p \geqslant 0$ в 43-м разряде стоит 1, а при $p < 0$ там стоит 0. Действительно, при

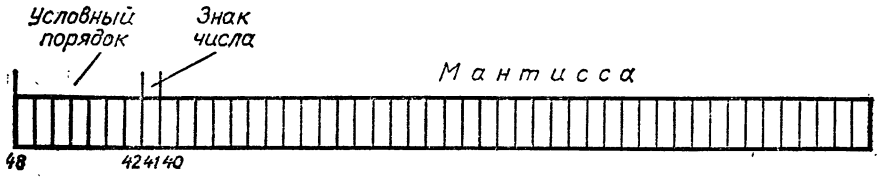


Рис. 47

$p \geqslant 0$ имеем $p' \geqslant 100$, а при $p < 0$ будет $p' < 100$. Поэтому можно рассматривать 43-й разряд как содержащий *знак порядка*, только здесь 1 означает знак плюс, а 0 — знак минус. При $p \geqslant 0$ порядок записан в разрядах 42—37 обычным образом (с присоединением 1 в 43 разряде), а при $p < 0$ в разрядах 42—37 записано число, дополняющее истинный порядок до 100_8. Так, порядок $p = -1$ изобразится в 42—37 разрядах числом 77, порядок $p = -12$ числом 66 и т. п.

В 48-разрядной ячейке памяти одноадресной машины БЭСМ-6 записывается одно число. Разряды разбиваются так (рис. 47): правые 40 разрядов (40—1) отводятся для записи мантиссы. Далее 41-й разряд содержит знак числа; как обычно, 0 означает знак плюс, 1 — знак минус. Разряды 48—42 отведены для записи условного порядка $p' = 100_8 + p$ точно таким же способом, как это только что было описано для машины БЭСМ-4. Принято говорить, что отрицательный порядок изображается в *дополнительном коде*.

Использование дополнительного кода основано на роли дополнений в арифметических действиях, что легко иллюстрируется следующим примером. Пусть требуется, например, произвести вычитание 456—187. Операцию можно выполнить так: 456—187=456—(1000—813)=(456+813)—1000=1269—1000=269. Как видно, вычитание сводится к сложению с десятичным дополнением, т. е. с числом, дополняющим данное до степени основания.

Аналогично *двоичным дополнением* называют число, дополняющее данное до целой степени двух (основания), и в двоичной системе вычитание также сводится к сложению с двоичным дополнением; нужно только не забывать вычитать степень основания, до которого мы дополняем, но в машине с фиксированным числом разрядов это произойдет само собой, так как эта единица потеряется, выйдя за пределы разрядной сетки.

Дополнительный код, благодаря удобству использования в арифметических действиях, широко используется не только при записи порядков, но также и при записи отрицательных мантисс. При этом изображением отрицательного числа в дополнительном коде называется его представление в виде двоичного дополнения, т. е. числа, дополняющего данное до соответствующей степени двух.

Оказывается, что в двоичной системе переход к дополнительному коду весьма прост. Действительно, рассмотрим для простоты, девятизначное двоичное число, например 100 101 011, и найдем выражение для его дополнения до 2^{10}. Для этого надо произвести вычитание:

$$1\,000\,000\,000 - 100\,101\,011 = 011\,010\,101.$$

Тот же результат (дополнительный код) получится, если в данном числе все нули заменить единицами, а все единицы — нулями и к младшему разряду прибавить единицу.

Кроме дополнительного кода, для представления отрицательных чисел в машинах иногда используется и так называемый *обратный*, или *инверсный*, код, который получается, если заменить все нули в записи числа единицами, а единицы — нулями. Как ясно из предыдущего, обратный код отличается от дополнительного единицей младшего разряда.

§ 8. ОПЕРАЦИОННЫЕ СХЕМЫ. РЕГИСТРЫ И ДЕШИФРАТОРЫ

Теперь после всех предварительных обсуждений мы можем перейти к рассмотрению *операционных схем*, используемых в различных устройствах цифровых вычислительных машин. В настоящем параграфе будут рассмотрены два типа операционных схем — регистры и дешифраторы.

Регистром называется устройство, предназначенное для запоминания машинных слов и простейших их преобразований. Регистр состоит из триггеров, количество которых соответствует количеству разрядов в машинном слове. Перечислим операции, выполняемые регистрами.

1. *Гашение* (или *сброс*) регистра, т. е. установка всех его разрядов в нуль.

2. Прием машинного слова из другого устройства (регистра памяти, арифметического устройства и т. п.) или передача в другое устройство.

Передачу слова можно осуществлять параллельно и последовательно. При последовательной передаче все разряды слова передаются последовательно во времени один за другим по одной цепи; при параллельной передаче слова все его разряды передаются одновременно, каждый по своей цепи. Говорят, что слово передается в последовательном коде или в параллельном коде.

3. Преобразование последовательного кода слова в параллельный или обратно.

4. Преобразование прямого кода числа в дополнительный или в обратный код и наоборот.

5. Сдвиг слова влево или вправо на нужное число разрядов.

6. Логическое сложение или логическое умножение машинных слов.

7. Поразрядное сложение.

Регистры могут быть как универсальными, так и рассчитанными на выполнение каких-либо отдельных операций из перечисленных. Рассмотрим блок-схемы регистров различного назначения, вне зависимости от системы элементов, на которых они построены.

На рисунке 48 приведена блок-схема регистра для приема машинного слова в параллельном коде. К единичным входам триггеров

присоединяются цепи передачи разрядных сигналов (считаем, что старшинство разрядов возрастает справа налево). С общей шины «Установка 0» на нулевые входы подается сигнал сброса регистра; после подачи этого сигнала все триггеры регистра будут находиться в нулевом состоянии до тех пор, пока на входных цепях не появятся сигналы входного слова.

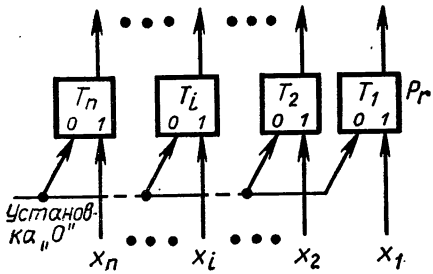


Рис. 48

В процессе передачи информации из одного регистра в другой можно осуществить не только перепись машинного слова, а и содержательную переработку машинных слов. Например, с помощью схемы, приведенной на рисунке 49, выполняется логическое сложение двух слов.

Для многих различных целей, например для умножения многозначных двоичных чисел, необходимо иметь возможность осуществлять *сдвиг* машинного слова, т. е. перемещение в регистре всех разрядов слова на одинаковое число разрядов влево или вправо. Одна из функциональных схем сдвигающего регистра показана на рисунке 50. Данный сдвигающий регистр называется *реверсивным*, так как в нем можно выполнить как сдвиг вправо (в сторону младших разрядов), так и сдвиг влево (в сторону старших разрядов).

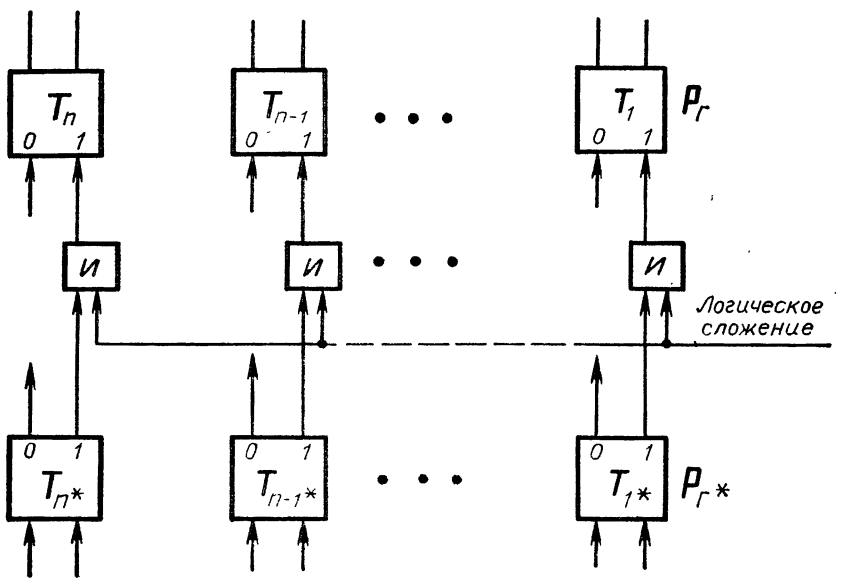


Рис. 49

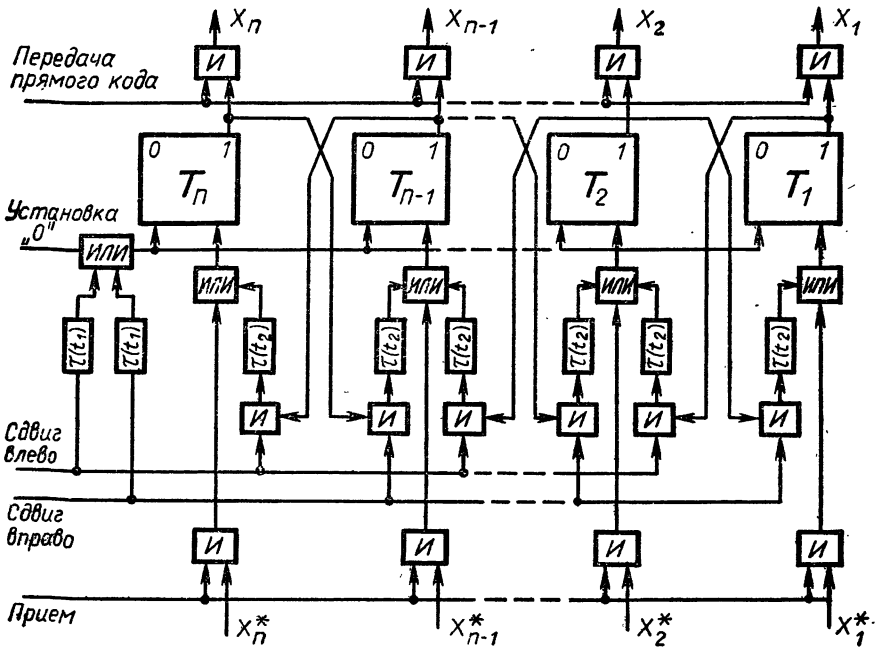


Рис. 50

Установка регистра в нуль производится сигналом «Установка 0», поступающим на шину сброса. Операция сдвига осуществляется следующим образом. Сигнал «Сдвиг влево» поступает по шине управления на клапаны *и* и кратковременно запоминается на элементах задержки $\tau(t_2)$, если на второй вход клапана *и* поступает с расположенного справа триггера сигнал 1. Если с этого триггера поступает сигнал 0, то клапан *и* не открывается и в элемент $\tau(t_2)$ импульс не поступает. Тот же сигнал «Сдвиг влево» поступает через небольшую задержку $\tau(t_1)$ в шину «Установка 0» и гасит содержимое регистра. После этого слово, разряды которого были кратковременно запомнены в элементах задержки $\tau(t_2)$, поступает на единичные установочные входы триггеров регистра и записывается в них. Если на $\tau(t_2)$ была запомнена 1, то соответствующий триггер устанавливается в положение 1; если же в элементе задержки сигнал отсутствовал, то соответствующий триггер останется в положении 0.

Описанная работа схемы обеспечивает сдвиг машинного слова на один разряд влево. При сдвиге на несколько разрядов необходимо подать последовательно требуемое число управляющих импульсов на шину «Сдвиг влево». Аналогично осуществляется сдвиг вправо.

Еще один тип операционных схем, которые будут рассмотрены в настоящем параграфе — *дешифраторы*. Дешифратором называют операционную схему, которая каждой комбинации входных

переменных ставит в соответствие выходной сигнал на одном определенном выходе схемы.

В общем случае дешифратор имеет n входов и 2^n выходов (n-разрядное двоичное число может принимать 2^n различных значений) и служит для преобразования кода слова в управляющий сигнал на одном из выходов. Так, например, код операции преобразуется дешифратором в управляющий сигнал, настраивающий арифметическое устройство на выполнение требуемой операции. Для простоты ограничимся случаем $n=3$. Тогда три двоичных переменных на входе можно рассматривать как триаду и задачей дешифратора будет выдать выходной сигнал на том из восьми выходов, номер которого изображается входной триадой, например на 5, если входная триада имеет вид 101.

Блок-схема *линейного* (или *матричного*) дешифратора на три входа показана на рисунке 51. Дешифратор содержит 8 элементов *и* с тремя входами каждый, на которые подаются все возможные комбинации прямых и инверсных значений разрядов входного слова. Легко проследить, что в каждом случае только на одном из элементов *и* все три входных сигнала будут равны единицам; только этот элемент и выдаст выходной сигнал в нужную цепь. Элементы *и* можно собирать группами в каскад из нескольких элементов. В зависимости от способа каскадного включения такие дешифраторы называют *прямоугольными* или *пирамидальными*.

В прямоугольном дешифраторе входное слово разбивается на части — *подслова* — и для каждого подслова на матричном дешифраторе образуются все выходные значения, которые называют

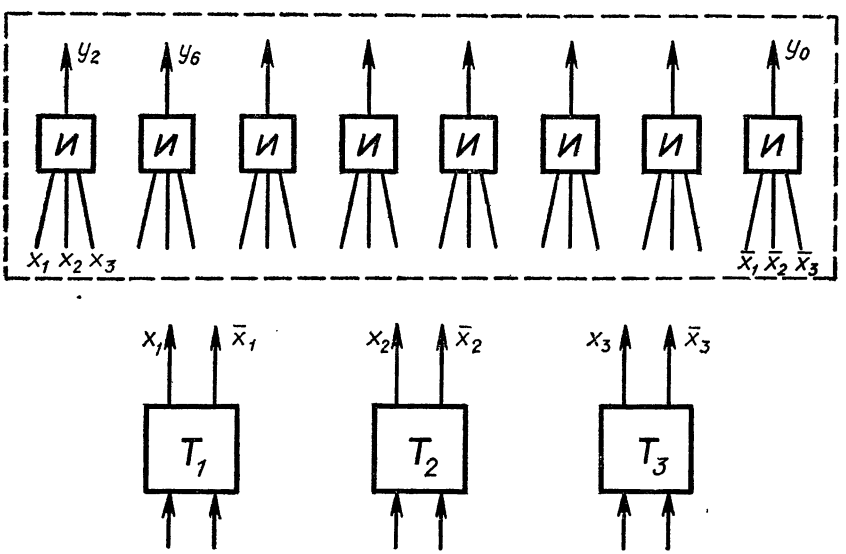


Рис. 51

частичными. Эта группа линейных дешифраторов, число которых равно числу подслов, образует первый каскад прямоугольного дешифратора. В каждом из последующих каскадов выполняется операция логического умножения частичных выходных значений.

§ 9. ОПЕРАЦИОННЫЕ СХЕМЫ. СЧЕТЧИКИ И СУММАТОРЫ

Счетчиками называют устройства, регистрирующие число импульсов, поступивших на вход. Двоичный счетчик, работающий в двоичной системе счисления, состоит из нескольких триггеров, соединенных между собой таким образом, что выполняется счет единичных входных импульсов. Счетчики делятся на *простые* и *реверсивные*. Простой счетчик считает сигналы, поступающие с одним знаком и имеет переход в одном (прямом) направлении. Реверсивный счетчик может принимать сигналы как суммируемые, так и вычитаемые и имеет переходы и в прямом, и в обратном направлениях. Кроме того, счетчик может реализовать переход из крайнего состояния (все единицы) в нулевое, что используется для выдачи сигнала переполнения.

Входной сигнал, принимаемый на триггер младшего разряда счетчика, изменяет его состояние, т. е. переводит из нулевого состояния в единичное или из единичного в нулевое. Таким образом, в этом разряде реализуется сложение (или вычитание) по модулю 2. В последующих разрядах аналогичное действие производит сигнал переноса — из младшего разряда в старший при сложении или из старшего в младший *(заем)* при вычитании. Такой счетчик легко осуществляется на триггерах со счетным входом (см. § 3), схема его показана на рисунке 52.

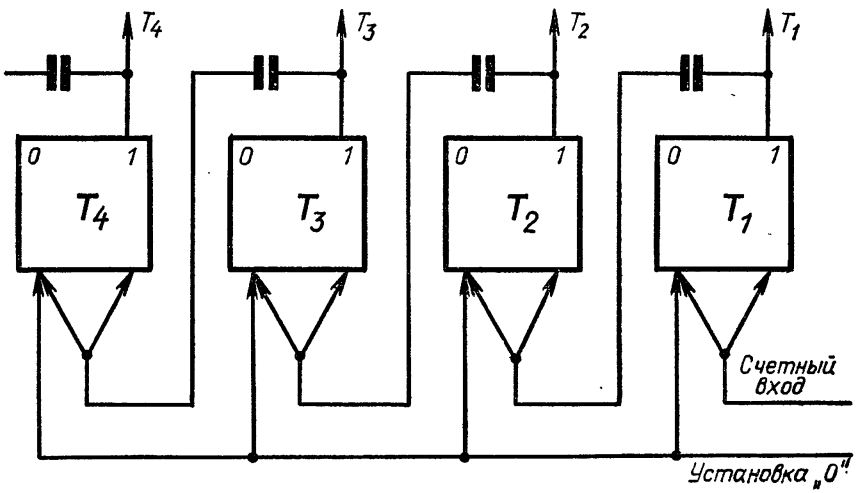


Рис. 52

Приведенная схема реализует суммирующий счетчик, сигнал переноса в котором образуется при дифференцировании перепада выходного напряжения триггера при переходе из единичного состояния в нулевое. Такой счетчик называют счетчиком с *последовательным переносом*: если в счетчике записано несколько единиц подряд, то сигнал переноса проходит все эти триггеры последовательно один за другим. Кроме таких счетчиков, существуют счетчики со *сквозным*, *групповым* и *частично-групповым* переносами.

Функциональная схема реверсионного счетчика со сквозным переносом приведена на рисунке 53. Здесь сигнал переноса поступает на шину сквозного переноса, в которую включены схемы *и* и *или*, управляемые прямыми или инверсными выходами триггеров. Если, например, в триггере младшего разряда записан нуль, то управляемый им клапан *и* будет закрыт и входной сигнал по шине «Сложение» поступит на счетный вход этого триггера; если же там записана единица, то схема *и* выдаст сигнал в следующий разряд, так что будут перевернуты одновременно оба триггера и из 01 получится 10; при нескольких единицах подряд все соответствующие клапаны *и* будут открыты и входной сигнал сразу дойдет до требуемого разряда. Время срабатывания такого счетчика много меньше, чем у счетчика с последовательным переносом. Оно определяется лишь длительностью переходного процесса в триггере и временем задержки τ.

Элементы задержки необходимы в схеме, чтобы обеспечить условия для прохождения сигналов через клапаны *и* по шине сквозного переноса.

В счетчике с групповым переносом входной сигнал подается параллельно на клапаны всех разрядов. Число входов у схем *и* увеличивается с возрастанием порядкового номера триггера и для многоразрядного счетчика может оказаться довольно большим. Вместе с тем в реальных элементах число входов клапана невелико и нагрузочная способность выходов триггеров ограничена. Поэтому разрядность счетчика с групповым переносом обычно ограничивается тремя-четырьмя разрядами. При большем числе разрядов счетчик разбивают на группы. Внутри каждой из групп строят цепи группового переноса, а перенос между группами реализуют методом сквозного переноса. Такой способ образования сигналов переноса называется *частично-групповым*. Счетчики с групповым и частично-групповым переносом являются наиболее быстродействующими.

Последним типом операционных схем, к рассмотрению которых мы переходим, являются *сумматоры*, выполняющие арифметическое суммирование двоичных чисел. Сумматор служит, прежде всего, центральным узлом арифметического устройства цифровой вычислительной машины, однако он находит применение также и в других устройствах машины.

Многоразрядный двоичный сумматор, предназначенный для сложения многоразрядных двоичных чисел, представляет собой комбинацию *одноразрядных* суммирующих схем, с рассмотрения кото-

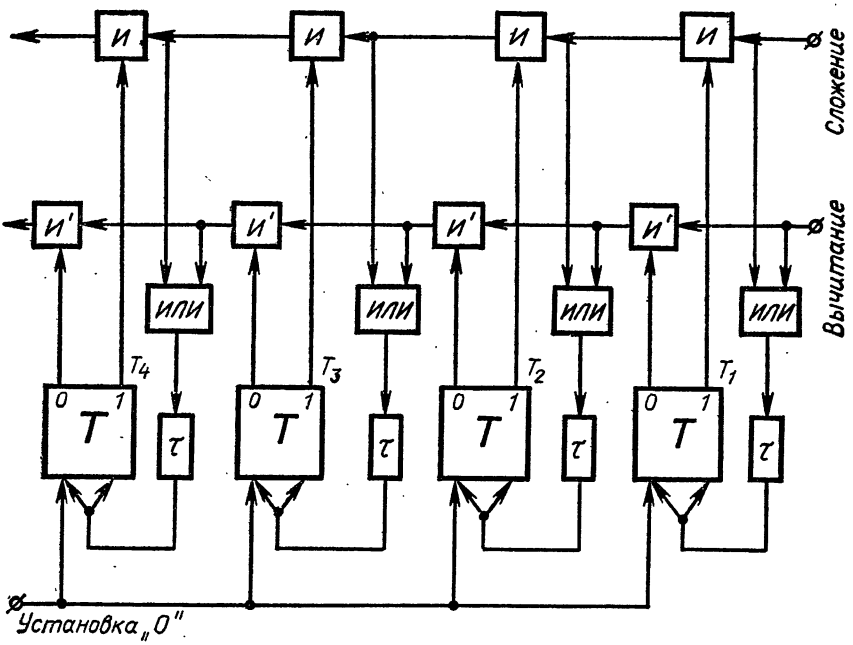


Рис. 53

рых мы и начнем. При сложении в одном разряде приходится иметь дело с тремя цифрами: цифры данного разряда первого и второго слагаемых и перенос в данный разряд из младшего. В результате сложения получаются две цифры — цифра данного разряда для суммы и перенос из данного разряда в старший.

Таким образом, одноразрядный двоичный сумматор есть устройство с тремя входами и двумя выходами, работа которого может быть описана следующей таблицей:

Входы			Выходы	
Первое слагаемое	Второе слагаемое	Перенос	Сумма	Перенос
0	0	0	0	0
0	0	1	1	0
0	1	0		
1	0	0		
1	1	0	0	1
1	0	1		
0	1	1		
1	1	1	1	1

Чтобы заменить приведенную таблицу формулами, введем обозначения. Пусть a, b означают цифры данного разряда первого и

второго слагаемых, c — перенос в данный разряд из младшего, S— цифру суммы в данном разряде и C — перенос из данного разряда в старший. Тогда, пользуясь правилами § 4, можем записать:

$$S = a \wedge \bar{b} \wedge \bar{c} \vee \bar{a} \wedge b \wedge \bar{c} \vee \bar{a} \wedge \bar{b} \wedge c \vee a \wedge b \wedge c, \qquad (1)$$

$$C = a \wedge b \wedge \bar{c} \vee a \wedge \bar{b} \wedge c \vee \bar{a} \wedge b \wedge c \vee a \wedge b \wedge c. \qquad (2)$$

Формулу (2) можно привести к более простому виду

$$C = a \wedge b \vee a \wedge c \vee b \wedge c. \qquad (3)$$

Справедливость этого утверждения легко проверить по таблице.

Схема, реализующая формулы 1 и 3, показана на рисунке 54. Приведенная схема формирует сигналы суммы и переноса, определяемые комбинацией входных сигналов слагаемых, одновременно поданных на входы схемы, и не обладает памятью: после снятия сигналов на входе выходные сигналы также снимаются. Такие сумматоры получили название *комбинационных сумматоров*. Они используются обычно в тех случаях, когда регистры арифметического устройства выполнены на триггерах, не имеющих счетных входов; результаты, полученные на выходе комбинационных схем, запоминаются в отдельном триггерном регистре.

Более широкое распространение получил *сумматор накапливающего типа*. Так называется схема, производящая суммирование поочередно поступающих на вход слагаемых с сохранением суммы после снятия сигнала очередного слагаемого. Функциональная схема и временная диаграмма работы одноразрядного накапливающего сумматора показаны на рисунке 55. Рассмотрим подробнее работу этой схемы.

В момент t_1 на счетный вход предварительно установленного в нулевое состояние триггера поступает цифра a первого слагаемого и запоминается. Спустя промежуток времени Δt (который должен превосходить длительность переходных процессов в триггере) на счетный вход поступает цифра b второго слагаемого. При этом в триггере происходит сложение цифр a и b по модулю два, что соответствует реализации логической функции

$$f_1 = a \wedge \bar{b} \vee \bar{a} \wedge b \qquad (4)$$

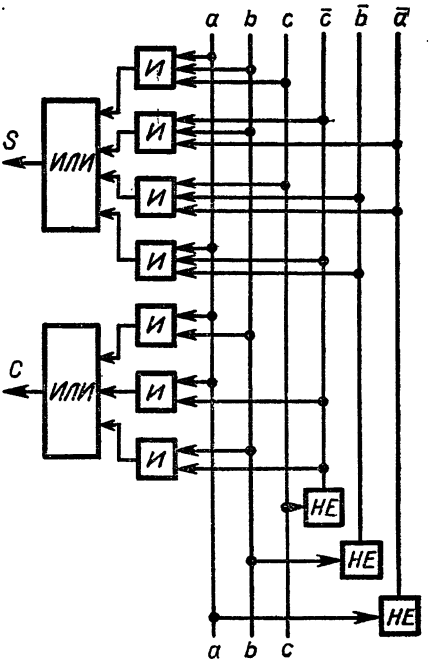


Рис. 54

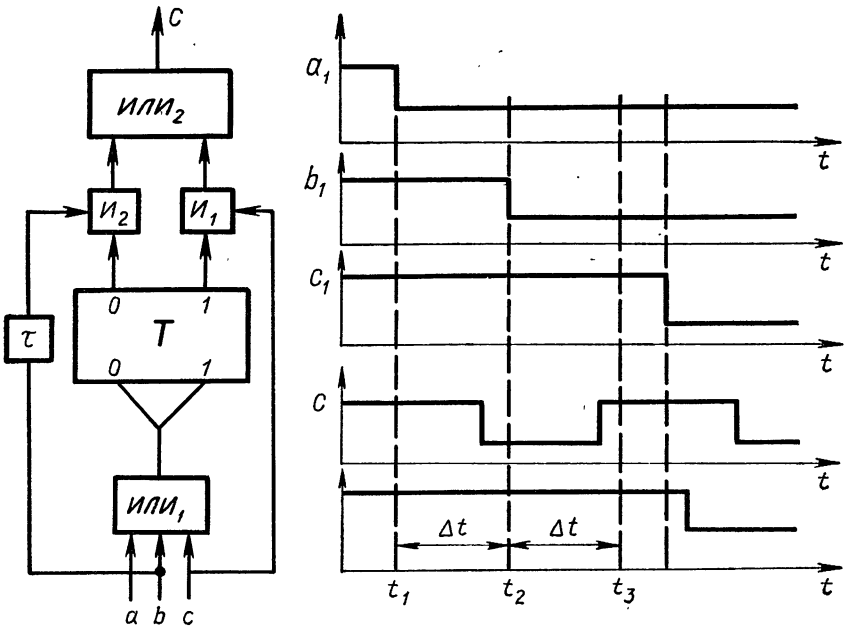


Рис. 55

(проверьте справедливость этого утверждения для всех возможных комбинаций a, b!). Спустя еще один промежуток Δt на тот же счетный вход подается цифра c переноса из младшего разряда, в результате чего триггер будет реализовать логическую функцию

$$f_2 = f_1 \wedge \bar{c} \vee \bar{f}_1 \wedge c.$$

Подставив в последнее выражение значение f_1 из (4) и преобразовав полученную формулу по правилам алгебры логики, о которых шла речь в § 4, придем к формуле

$$f_2 = a \wedge \bar{b} \wedge \bar{c} \vee \bar{a} \wedge b \wedge \bar{c} \vee \bar{a} \wedge \bar{b} \wedge c \vee \bar{a} \wedge b \wedge c, \qquad (5)$$

которая тождественна с формулой (1). Отсюда следует, что схема рисунка 55 действительно осуществляет сложение одноразрядных двоичных чисел.

Остается проверить правильность выработки сигнала переноса в старший разряд. Этот сигнал в нашей схеме имеет две составляющие. Первая из них вырабатывается элементом u_1, реализующим функцию

$$f_3 = f_1 \wedge c = a \wedge \bar{b} \wedge c \vee \bar{a} \wedge b \wedge c = a \wedge c \vee b \wedge c.$$

Эта составляющая возникает при взаимодействии цифр слагаемых с сигналом переноса c из младшего разряда. Вторая составляющая вырабатывается при взаимодействии цифр a и b первого и вто-

рого слагаемых. Она не может быть получена подачей на элемент u_1 слагаемых a и b, так как они поступают в различные моменты времени. Поэтому вторая составляющая реализуется элементом u_2, на один из входов которого подается сигнал с нулевого выхода триггера, а на другой — слагаемое b через элемент задержки τ. Схема u_2 реализует функцию

$$f_4 = F_1 \wedge b = a \wedge b^*.$$

Окончательно сигнал переноса на выходе схемы $или_2$ определяется выражением

$$C = f_3 \vee f_4 = a \wedge c \vee b \wedge c \vee a \wedge b, \qquad (6)$$

что совпадает с формулой (3).

Достоинством сумматора накапливающего типа является способность запоминать результат, благодаря чему его можно применять для нескольких последовательных сложений. Недостаток — необходимость д в у х т а к т о в (подачи второго слагаемого и подачи сигнала переноса) для получения суммы, что увеличивает время срабатывания.

Полный — многоразрядный — двоичный сумматор образуется из одноразрядных сумматоров, число которых должно быть равно числу разрядов слагаемых, путем соединения выходов с сигналами переноса в старшие разряды с входами с сигналами переноса из младших. На рисунке 56 показана схема сумматора накапливающего типа с последовательным переносом из разряда в разряд.

Содержимое триггеров сумматора представляет первое слагаемое. Второе слагаемое поступает поразрядно на входы схем $и$. С поступлением синхронизирующего (тактового) импульса цифры второго слагаемого поступают на счетные входы триггеров сумматора. В случае переброса триггера из состояния 1 в состояние 0 формируется с помощью дифференцирующей цепочки импульс переноса, который поступает через элемент задержки на счетный вход соседнего (старшего) разряда. Последний, в свою очередь, мог находиться в состоянии 1, так что поступивший импульс может вызвать необходимость переноса в следующий старший разряд. При определенных условиях возможен такой «пробег» единицы переноса через весь триггерный регистр, что, естественно, весьма отрицательно сказывается на времени выполнения операции. Аналогичные соображения уже высказывались в начале настоящего параграфа при рассмотрении счетчиков с последовательным переносом.

Ускорение работы сумматора за счет времени переноса может быть достигнуто несколькими различными путями. Один из них — схема сквозного переноса — был предложен Чарльзом Бэббеджем около 1840 года. Его идея состоит в том, чтобы сформировать сигнал переноса до образования цифры суммы в каждом из разрядов. Схема одного из вариантов такого сумматора показана на рисунке 57.

---

* Мы не приводим преобразований, которые производятся по правилам, изложенным в § 4.

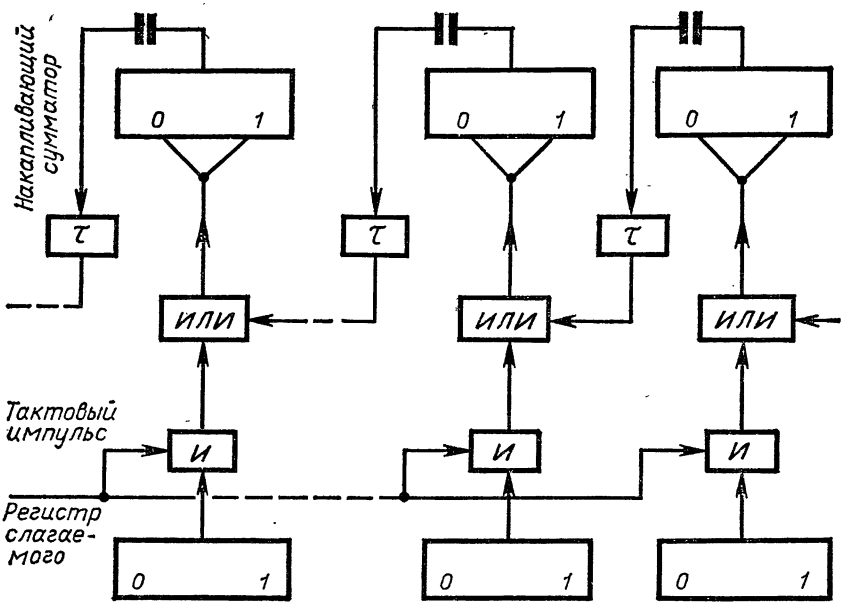

Накапливающий сумматор
0
1
τ
ИЛИ
Тактовый импульс
И
Регистр слагаемого

Рис. 56

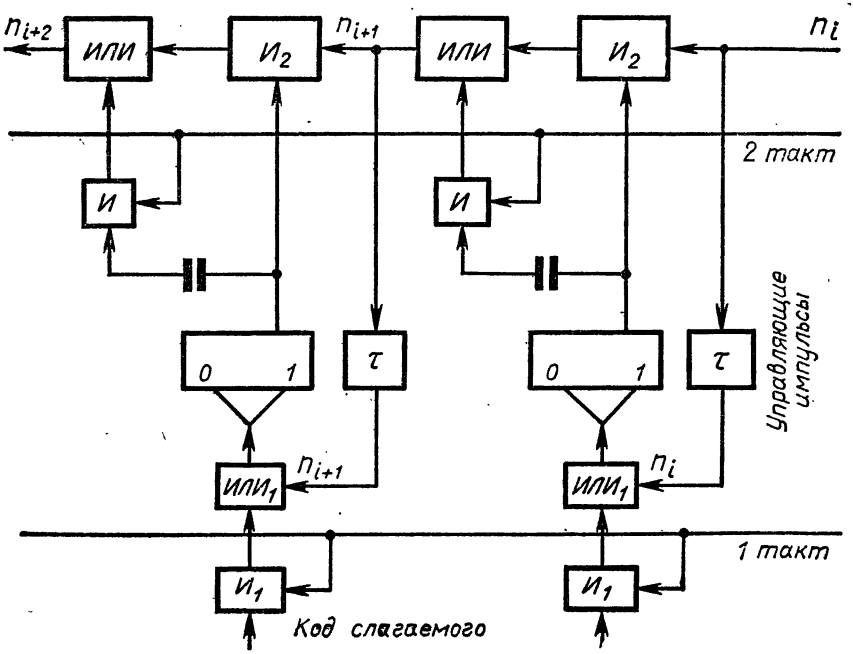

n_{i+2}
ИЛИ
$И_2$
n_{i+1}
n_i
2 такт
И
0
1
τ
Управляющие импульсы
$ИЛИ_1$
1 такт
$И_1$
Код слагаемого

Рис. 57

Входные шины сумматора подключены к единичным выходам триггеров регистра второго слагаемого. Сложение происходит в два такта. В первом такте подается *импульс поразрядного сложения.* Если в каком-либо разряде регистра второго слагаемого находится 1, то этот импульс через схемы $и_1$ и $или_1$ проходит на счетный вход триггера сумматора и меняет его состояние на противоположное, т. е. выполняет сложение по модулю 2. Во втором такте — спустя время, большее, чем время прохождения сигнала через логические элементы и время переходного процесса — подается импульс переноса. Формирование сигнала переноса осуществляется схемой $и_2$ аналогично тому, как это происходит в счетчике со сквозным переносом и было подробно описано в начале параграфа (см. схему на рис. 53 и описание ее работы на стр. 81).

Длительность переноса в таком сумматоре определяется временем прохода через логические схемы *и* и *или* и временем задержки τ, которые обычно много меньше, чем время срабатывания триггера.

Глава IV

ЦИФРОВЫЕ ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ. ОСНОВНЫЕ УСТРОЙСТВА

§ 1. АРИФМЕТИЧЕСКОЕ УСТРОЙСТВО

В предыдущей главе мы познакомились с блок-схемой цифровой вычислительной машины, а затем, углубившись в ее прямоугольники, быстро добрались до самых далеких внутренних элементов — операционных схем и их элементов. Сейчас нам предстоит возвратиться обратно и рассмотреть как раз сами эти прямоугольники — основные устройства цифровой вычислительной машины, и в первую очередь *арифметическое устройство*. Арифметическое устройство предназначено для выполнения основных арифметических и логических операций, а также различных сдвигов. Мы будем предполагать, как уже говорилось в § 7 предыдущей главы, что речь идет о машине, в которой числа представляются в форме с плавающей запятой.

Основой арифметического устройства является *сумматор*, так как выполнение арифметических операций сводится в конечном счете к сложению. В связи с этим быстродействие сумматора и способ выполнения сложения в заметной степени определяют время выполнения остальных арифметических операций.

Как правило, арифметическое устройство строится по трехрегистровой схеме: два регистра предназначаются для исходных чисел, над которыми будет совершаться требуемая операция *(операндов)*, и третий — *регистр сумматора* — используется для формирования результата операции. Каждый из регистров разделен на две части: одна из них предназначена для порядка числа, другая — для мантиссы. Трехрегистровая схема арифметического устройства не имеет отношения к адресности машины и применяется, скажем, в одноадресных машинах: просто перед началом операции в один из регистров для исходных чисел пересылается число из регистра, предназначенного для результата, а в другой — из ячейки памяти.

Арифметическое устройство связано *кодовыми шинами* с оперативной памятью и *управляющими шинами* с устройством управления. Из памяти поступают *операнды* (впрочем, один из них может поступить с регистра результата), а из устройства управления — код операции. Выполнение операции сводится к выполнению после-

довательности *микроопераций*, которые будем называть *тактами*, поскольку каждая отдельная микрооперация выполняется в течение такта машины (см. § 6 гл. III). К микрооперациям—тактам — относятся:

1. Сброс отдельных регистров для подготовки приема новой информации.
2. Прием на регистр новой информации.
3. Выдача информации с регистра.
4. Преобразование прямого кода числа в инверсный или обратно.
5. Суммирование кодов, которое в свою очередь расчленяется на два такта:

а) поразрядное суммирование и

б) окончательное суммирование с учетом переносов.

6. Сдвиг кода числа в регистре на один разряд влево или вправо.

Возможны и еще некоторые другие микрооперации, о которых мы упомянем в дальнейшем, когда это потребуется.

Местное управление операциями, включенное в состав арифметического устройства и связанное с центральным устройством управления, вырабатывает последовательность импульсов, нужных для выполнения каждой микрооперации, и подает их на соответствующие управляющие шины.

Содержимое сумматора мантисс, так же как и регистров мантисс исходных чисел, следует рассматривать как фиксированное число, запятая в котором фиксирована перед старшим значащим разрядом. Оба сумматора имеют дополнительные разряды переполнения, сумматор мантисс, кроме того, дополнительный младший разряд справа от первого. В сумматоре мантисс разряд переполнения фиксирует нарушение нормализации; в сумматоре порядков разряд переполнения фиксирует выход за пределы допустимого для данной машины диапазона чисел.

Прежде чем производить сложение мантисс, необходимо проверить порядки слагаемых, так как при различных порядках одинаковые разряды мантисс не соответствуют друг другу. Поэтому первым тактом (микрооперацией) сложения чисел является вычитание порядков, которое выполняется в сумматоре порядков: порядок p_2 второго слагаемого вычитается из порядка p_1 первого. Возможны следующие случаи.

1. $p_1 - p_2 = 0$. Равенство означает, что порядки слагаемых равны. Тогда одинаковые разряды мантисс слагаемых соответствуют друг другу и можно производить суммирование мантисс. Операция сложения выполняется далее следующим образом: в сумматор полностью переписывается первое слагаемое — порядок в сумматор порядка, а мантисса в сумматор мантисс. Затем к мантиссе первого слагаемого прибавляется мантисса второго. Если при этом образуется единица в разряде переполнения сумматора мантисс, то содержимое сумматора мантисс сдвигается на один разряд вправо, так чтобы разряд переполнения попал в старший разряд мантиссы, а ее младший разряд сместился в дополнительный, находящийся пра-

вее младшего. Одновременно порядок увеличивается на единицу. Таким образом мы достигаем нормализации суммы; такая нормализация называется *нормализацией вправо*.

Может случиться (например, если исходные числа были ненормализованы или при сложении чисел разных знаков, о чем мы еще будем говорить ниже), что один или даже несколько старших разрядов мантиссы окажутся равными нулю. В этом случае мантисса сдвигается на один разряд влево и одновременно порядок уменьшается на единицу. После этого снова проверяется старший разряд мантиссы, и если он опять нуль, то сдвиг и уменьшение порядка повторяются, и так до тех пор, пока старший разряд не сделается единицей, а число — нормализованным. Такую нормализацию называют *нормализацией влево*.

2. $p_1 - p_2 = k$, причем $0 < k \leqslant n$, где n — число разрядов мантиссы в машине. В такой ситуации поразрядное сложение мантисс невозможно; необходимо предварительно выравнять порядки путем «разнормализации» одного из слагаемых. Эта операция выполняется так. В сумматор помещается второе (меньшее) слагаемое. Затем к его порядку прибавляется k, так что получается порядок p_1, а мантисса сдвигается на k разрядов вправо, чем ее разряды приводятся в соответствие с разрядами мантиссы большего слагаемого. Теперь можно складывать мантиссы; в случае надобности производится нормализация суммы в нужную сторону.

3. $p_1 - p_2 < 0$, но $p_2 - p_1 = k \leqslant n$. Сложение выполняется так же, как и в предыдущем случае, но в сумматор записывается первое (снова меньшее!) слагаемое, которое и подвергается разнормализации.

4. $p_1 - p_2 = k > n$. Здесь можно в качестве суммы сразу брать первое слагаемое, так как при разнормализации второго его мантисса уйдет за пределы разрядной сетки.

5. $p_1 - p_2 < 0$, но $p_2 - p_1 = k > n$. Ситуация аналогична предыдущей, но в качестве суммы следует брать второе слагаемое (большее, как и в предыдущем случае).

Заключительным этапом сложения является *округление* результата, которое производится путем прибавления единицы в младший дополнительный разряд.

Если при разнормализации меньшего слагаемого и сдвиге или при нормализации вправо в младший дополнительный разряд попала единица, то округление увеличит младший разряд; если же там был нуль, то прибавленная единица так и останется в дополнительном разряде, который отсечется при записи результата в память или при переносе из регистра результата в регистр операндов для следующей операции.

Для большей ясности рассмотрим несколько примеров сложения чисел, ограничиваясь шестиразрядными мантиссами и трехразрядными истинными порядками. Слева и справа от регистра сложения мантисс в рамке заключаем разряд переполнения мантисс

и младший дополнительный разряд. Пусть сначала требуется сложить числа

$$\begin{array}{r} 101\,100\,101 \\ +\,101\,101\,001. \end{array}$$

Так как здесь $p_1 - p_2 = 0$, то можно сразу складывать мантиссы, что дает

$$101\,\boxed{1}\,001\,110\,\boxed{0},$$

а после нормализации вправо

$$110\,\boxed{}\,100\,111\,\boxed{}.$$

Округление не изменит результата, так как в младшем дополнительном разряде стоит 0. При округлении туда будет записана 1, которая исчезнет при использовании результата, так как используются — записываются в память или переносятся в регистр исходных чисел — лишь основные разряды мантиссы.

Рассмотрим теперь сложение чисел различных порядков. Пусть слагаемые равны

$$\begin{array}{l} 101\,100\,110, \\ 010\,101\,100. \end{array}$$

Так как здесь $p_1 - p_2 = 011$, то в регистр результата сначала просто перепишется второе слагаемое

$$010\,\boxed{}\,101\,100\,\boxed{},$$

которое после выравнивания порядков и сдвига приобретет вид

$$101\,\boxed{}\,000\,101\,\boxed{1}.$$

В результате сложения получим:

$$101\,\boxed{}\,101\,011\,\boxed{1},$$

а округление изменит последнюю триаду результата, так что окончательно будет:

$$101\,\boxed{}\,101\,100\,\boxed{}.$$

Пока мы имели дело со сложением положительных чисел или вообще чисел одного знака. На самом деле может потребоваться сложение чисел различных знаков, эквивалентное вычитанию. В предыдущей главе (см. § 7) уже замечалось, что вычитание можно свести к сложению с дополнительным кодом числа, причем в этом случае в операции участвует также знаковый разряд. Ограничимся для

простоты и ясности действиями с целыми шестиразрядными двоичными числами с учетом их знака (как обычно, 0 означает, что число положительно, 1 — отрицательно). Пусть требуется выполнить вычитание 65_8—23_8. Запись этих двух чисел в двоичной системе со знаком впереди будет 0 110 101 и 1 010 011 — в прямом коде. Дополнительный код числа 23_8 имеет вид 1 101 101; он получается, если в записи числа нули заменить единицами, а единицы нулями — обратный код — и затем прибавить единицу младшего разряда; разумеется, знаковый разряд в этом преобразовании не участвует. Выполним теперь сложение полученных кодов с учетом и знаковых разрядов:

$$\begin{array}{r} 0\,110\,101 \\ +\,1\,101\,101 \\ \hline 0\,100\,010 \end{array},$$

а единица переноса, образовавшаяся в знаковом разряде, теряется. Мы получили $+42_8$, т. е. требуемый результат. То же получится при вычитании 12_8—37_8. Действительно,

$$\begin{array}{r} 0\,001\,010 \\ +\,1\,100\,001 \\ \hline 1\,101\,011 \end{array}.$$

Полученный результат записан в дополнительном коде (знаковый разряд 1). Обратный код этого числа будет 1 101 010, а, значит, прямой 1 010 101, т. е.— 25_8, как и должно быть.

Легко перенести высказанные соображения и на действия с числами с плавающей запятой. Таким образом, для вычитания — или сложения чисел с разными знаками — мы получаем такой набор микроопераций. Сначала отрицательное число (или вычитаемое) следует перевести в дополнительный код. Для этого его переводят в обратный код, передавая число не с прямых, а с инверсных выходов триггеров регистра, а затем прибавляют единицу младшего разряда. После этого выполняется обычное сложение, в которое входит выравнивание порядков слагаемых и сложение мантисс, при этом участвуют также и знаковые разряды. Результат (после округления и нормализации) может содержать нуль в знаковом разряде, и тогда он является окончательным, а может содержать в знаковом разряде единицу. В этом случае он записан в дополнительном коде и его следует перевести в прямой код, для чего достаточно вычесть из него единицу младшего разряда, а затем передать с инверсных входов триггеров всех разрядов регистра, исключая знаковый.

Умножение в двоичной системе счисления сводится к последовательности сложений и сдвигов: так как множитель состоит лишь из нулей и единиц, то частичные произведения совпадают с множимым и могут только сдвигаться внутри разрядной сетки. Знак же и порядок произведения для чисел с плавающей запятой получаются очень просто. Знак произведения получается сложением знаковых

разрядов сомножителей без переноса — единицу переноса, если она образуется, следует опустить. Предварительный порядок произведения получается как сумма порядков сомножителей, так что остается получить произведение мантисс.

Возможны различные схемы получения произведения мантисс: начиная со старших разрядов или с младших, с неподвижным множимым или с неподвижными частичными произведениями. Все они вполне эквивалентны математически, но требуют для своего технического осуществления различного оборудования и поэтому неравноценны технически и экономически. Рассмотрим один из возможных методов — умножение, начиная с младших разрядов множителя при неподвижном множимом, который довольно часто используется в арифметических устройствах.

Этот способ равносилен обычному умножению многозначных чисел «столбиком», который легко проследить на примере перемножения шестизначных чисел

$$\begin{array}{r} 0{,}101\,011 \\ \times\ 0{,}100\,101 \\ \hline 101\,011 \\ 10\,101\,1 \\ 10\,101\,1 \\ \hline 0{,}011\,000\,110\,111 \end{array}.$$

Из этого примера видно, что для сохранения всех разрядов произведения сумматор частичных произведений должен иметь вдвое больше разрядов, чем число разрядов мантиссы* . При этом регистр множителя и сумматор частичных произведений должен иметь цепи сдвига вправо. Выполняемые действия определяются младшим разрядом множителя. Так как вначале этот разряд содержит единицу, то множимое переносится в сумматор частичных произведений в крайнее левое положение, так что содержимое сумматора приобретает вид

101 011 000 000.

Затем производится сдвиг вправо на один разряд в сумматоре частичных произведений и в регистре множителя. В результате такого сдвига последний разряд множителя оказывается равным нулю и прибавление не производится. После второго сдвига содержимое сумматора будет иметь вид

001 010 110 000

* На самом деле, для хранения младших разрядов произведения можно использовать старшие разряды регистра множителя, которые освобождаются по мере сдвига множителя вправо. Для этого при сдвиге младший разряд сумматора частичных произведений соединяют со старшим разрядом регистра множителя. Благодаря этому можно для сумматора иметь столько же разрядов, сколько и в мантиссе.

и, поскольку здесь младший разряд множителя — единица, то сюда снова прибавляется множимое в крайнем левом положении. Сумма будет равна 110 101 110 000. Аналогичные действия производятся до тех пор, пока не будут проверены все разряды множителя.

Как и в нашем примере, результат может оказаться ненормализированным — первая цифра мантиссы здесь равна нулю. Тогда производится нормализация влево, путем сдвига мантиссы влево на один разряд с одновременным вычитанием единицы из порядка до тех пор, пока первая цифра мантиссы не станет единицей. После нормализации результат округляется, как и при сложении, прибавлением единицы в младший дополнительный разряд, который потом, при записи произведения в память, отсекается вместе со всеми оставшимися младшими разрядами. В нашем случае шестиразрядных мантисс нормализация приведет к числу 0,110 001 $\boxed{\bar{1}}$01 11, в котором цифра, попавшая в младший дополнительный разряд, взята в рамку. Округление дает число 0,110 010 001 11, так что в память будет записываться результат 0,110 010.

Деление в цифровой вычислительной машине выполняется путем последовательного вычитания делителя из делимого. Прежде всего отметим, что знак частного получается так же, как и знак произведения, суммированием знаковых разрядов. Порядок частного легко получить вычитанием порядков делимого и делителя, после чего остается найти частное мантисс. Для этого из делимого, а затем из образующихся в процессе вычислений частичных остатков последовательно вычитается делитель, а затем частичный остаток сдвигается на один разряд влево. Если очередное вычитание оказывается возможным, то соответствующая цифра частного 1, если делитель больше предыдущего остатка, то в частное записывается нуль. В самом начале деления первая цифра мантиссы частного получается после сдвига делимого на разряд влево; если же вычитание оказалось возможным без сдвига, т. е. если мантисса делимого оказалась больше мантиссы делителя, то порядок частного следует увеличить на единицу.

§ 2. ПАМЯТЬ МАШИНЫ. ОБЩАЯ ХАРАКТЕРИСТИКА

Основными техническими характеристиками запоминающего устройства, как мы уже говорили в § 1 предыдущей главы, является *емкость* и *время обращения*. Обычно емкость оперативной памяти выбирается равной $N=2^m$, где m — количество разрядов адреса. Это дает возможность непосредственной адресации любой ячейки памяти в команде. Таким образом, единицей измерения считается ячейка памяти: емкость есть число машинных слов определенной разрядности, которые могут одновременно храниться в памяти. Для оперативной памяти, построенной на ферритовых сердечниках, иногда вводят укрупненную единицу емкости — *куб* ($1k=2^{10}=$ 1024 машинных слов). В некоторых случаях под емкостью запоминаю-

щего устройства понимают количество *запоминающих элементов* $M=Nn$, где n — число разрядов машинного слова в данной машине. Тогда емкость измеряется в *битах — двоичных разрядах.*

В некоторых машинах третьего поколения единицей измерения емкости считается не машинное слово и не отдельный разряд, а промежуточная единица: *группа* разрядов, чаще всего 8, изредка 12 или 4, называемая *байтом*. В такой машине можно работать со словами переменной длины, составленными из нескольких байтов, являющихся элементами информации и имеющих свою адресацию. Число байтов, объединяющихся в машинное слово, может быть различным в зависимости от требований задачи и определенным образом указывается в команде.

Быстродействие запоминающего устройства характеризуется временем обращения, которое складывается из *времени поиска* $t_п$, затрачиваемого на поиск нужного слова, и *времени записи* или *времени считывания* нужной информации.

Время поиска определяется, главным образом, *способом выборки* информации. По способу выборки запоминающие устройства делятся на устройства с *произвольной*, с *периодической* и с *последовательной выборкой*. В памяти с *произвольной выборкой* мы имеем доступ к любой ячейке памяти как при записи, так и при считывании. К такому типу памяти, позволяющему общаться с любой ячейкой по указанному адресу за одно и то же время, относится память на ферритовых сердечниках, широко используемая в качестве оперативной памяти, как о том уже говорилось в § 1 гл. III. В памяти с *периодической выборкой* все ячейки последовательно во времени проходят мимо считывающих или записывающих элементов *(головок)*. Типичным примером такого запоминающего устройства являются *магнитные барабаны* или *магнитные диски*. За один оборот барабана, например, могут быть последовательно выбраны машинные слова, расположенные по всей его поверхности. Таким образом, время поиска оказывается весьма различным в зависимости от момента поступления сигнала на чтение или запись: максимальное время равно периоду полного оборота барабана вокруг своей оси.

Примером запоминающего устройства с *последовательной выборкой* является магнитная лента. Чтобы получить доступ к машинному слову на ленте, необходимо перемотать ее от того места, где находится считывающая (или записывающая) головка до требуемого слова, для чего может иногда потребоваться довольно большое время. Поэтому, в частности, обмен информацией с магнитной лентой производится, как правило, целыми *зонами*. Магнитные ленты, а также барабаны и диски, т. е. запоминающие устройства с последовательной или периодической выборками, обычно используются в качестве устройств внешней памяти. Это происходит именно вследствие сравнительно большого среднего времени поиска по сравнению с памятью с произвольной выборкой.

Для общения с другими устройствами машины память имеет два внешних регистра: *регистр адреса*, куда из устройства управления передается адрес ячейки, и *регистр числа*. Сюда записывается машинное слово, которое следует запомнить в ячейке памяти, или,

наоборот, сюда выбирается из ячейки памяти машинное слово, которое затем должно передаваться в другие устройства машины.

Теперь мы можем обратиться к вопросу о *физической реализации* запоминающих устройств.

§ 3. ПРИНЦИПЫ ФИЗИЧЕСКОЙ РЕАЛИЗАЦИИ ОПЕРАТИВНОЙ МАГНИТНОЙ ПАМЯТИ

Источником магнитных явлений является движение электрических зарядов. Ток — движение зарядов по проводнику — создает в окружающем пространстве *магнитное поле*, которое оказывает силовое воздействие на проводники с током и намагниченные тела. Величина этого воздействия пропорциональна интенсивности магнитного поля, которая называется *напряженностью* поля и обычно обозначается буквой H. Магнитное поле в каждой точке пространства характеризуется не только напряженностью, определяющей величину силового воздействия, но и направлением этого воздействия, т. е. является величиной векторной. Направление поля можно определить, внеся в него свободно подвешенную магнитную стрелку: она будет располагаться по направлению магнитного поля в данной точке.

Намотаем на сердечник (рис. 58) из магнитного материала несколько витков изолированного провода и пропустим через него электрический ток. Последний создаст в сердечнике магнитное поле — *магнитный поток сердечника*. Величина магнитного потока Ф определяется напряженностью в н е ш н е г о магнитного поля, равной $H=Iw$, где I — ток в обмотке, а w — число витков, и магнитными свойствами материала самого сердечника. Величина магнитного потока, приходящаяся на единицу площади поперечного сечения (S), называется *магнитной индукцией*; она обозначается буквой B. Из определения следует, что $B=\Phi/S$. Направление магнитного потока определяется полярностью тока в обмотке. Выбор положительного направления позволяет приписать магнитному потоку, а следовательно, и магнитной индукции определенный з н а к.

Важнейшей характеристикой магнитных свойств материала является *магнитная проницаемость*, определяемая как отношение магнитной индукции B (в образце данной конфигурации) к напряженности H магнитного поля, создавшего поток, $\mu=B/H$. В зависимости от величины μ все материалы делятся на *диамагнитные* ($\mu<1$), *парамагнитные* ($\mu\approx1$) и *ферромагнитные* ($\mu\gg1$). Примерами диамагнитных материалов могут служить золото, серебро, медь; парамагнитных — платина, алюминий; ферромагнитных — железо, никель, кобальт. В дальнейшем мы будем иметь дело только с ферромагнитными материалами.

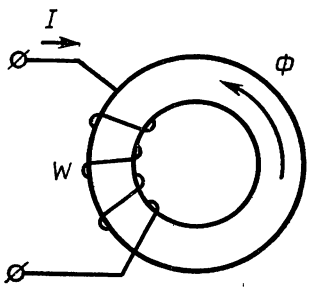


Рис. 58

Ферромагнитные материалы содержат большое число «элементарных магнитиков» — *доменов*, объем которых колеблется от 10^{-1} до 10^{-9} *см*^{3}. В размагниченном ферроматериале домены расположены беспорядочно, и создаваемые ими поля компенсируют друг друга, сводя к нулю магнитный поток. При подаче тока в обмотку сердечника создается внешнее магнитное поле, под действием которого домены поворачиваются и ориентируются по направлению поля: таково упрощенное представление процесса намагничивания ферромагнитного сердечника.

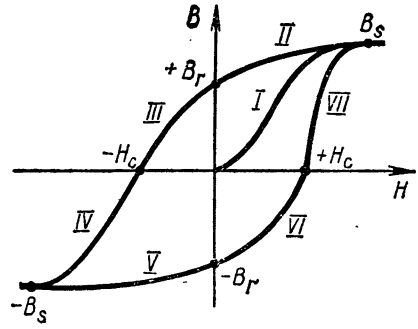


Рис. 59

Графически процесс намагничивания и размагничивания изображен на рисунке 59. Исходному (размагниченному) состоянию сердечника соответствует начало координат (0,0). Будем подавать в обмотку ток положительной полярности. По мере увеличения тока I, а значит, и напряженности H будет происходить и увеличение магнитной индукции сердечника B, так как все большее количество доменов будут ориентироваться в направлении внешнего магнитного поля (кривая I). При некоторой величине I_s (и $H_s = I_s w$, где w фиксировано) все домены окажутся направленными в одну сторону. Дальнейшее увеличение тока не приведет к увеличению магнитной индукции; ферромагнитный материал находится *в состоянии насыщения*. Току I_s (напряженности поля H_s) соответствует значение магнитной индукции B_s, называемое *индукцией насыщения*.

Рассмотрим теперь процессы, которые будут происходить в сердечнике при уменьшении тока. Размагничивание будет проходить в соответствии с кривой II, которая существенно отличается от предыдущей. При уменьшении тока часть доменов будет терять ориентацию, и значение B будет уменьшаться. Однако механические внутренние напряжения, дефекты и пустоты, инородные включения, всегда присутствующие в кристаллической структуре ферромагнита, будут препятствовать полной «разориентации» доменов. Поэтому даже при выключении тока ($H=0$) некоторое число доменов в сердечнике все еще будут сохранять некоторую упорядоченность, и в сердечнике сохранится состояние намагниченности, характеризуемое *остаточной* магнитной индукцией B_r.

Чтобы полностью размагнитить сердечник, необходимо создать внешнее магнитное поле противоположной полярности, для чего достаточно подать в обмотку ток противоположного направления (кривая III). Полная разориентация всех доменов произойдет при напряженности поля — H_c, называемой *коэрцитивной силой*. Если теперь продолжать увеличивать ток в том же направлении, то сердечник перемагнитится в противоположном направ-

л е н и и, т. е. в сердечнике снова возникает магнитный поток, но уже в направлении, противоположном предшествовавшему. Процесс намагничивания будет происходить в соответствии с кривой IV. Аналогично будет идти и процесс р а з м а г н и ч и в а н и я от точки ($-H_s$, $-B_s$) до точки (0, $-B_r$) (кривая V) и далее, до точки (H_c, 0) (кривая VI). Если продолжить увеличивать ток сверх значения, соответствующего коэрцитивной силе, то будем вновь наблюдать намагничивание, при котором магнитный поток опять дойдет до состояния насыщения (кривая VII).

Таким образом, полный цикл перемагничивания ферромагнитного сердечника изобразится замкнутой кривой, которую называют *петлей гистерезиса*. Она графически отражает отставание намагниченности от напряженности магнитного поля. Можно показать, что площадь, ограниченная петлей, пропорциональна энергии источника тока, необходимой для осуществления полного цикла перемагничивания; эта энергия рассеивается в основном в виде тепла.

Способность ферромагнитных материалов сохранять намагниченность при выключении тока лежит в основе их использования для построения запоминающих устройств. Как видно из графика рисунка 59, при $I=0$ магнитный поток может иметь в зависимости от предыдущего направления тока одно из двух различных значений, $+B_r$ или $-B_r$. Эти состояния можно использовать для запоминания двоичных цифр, поставив в соответствие, например, значению $+B_r$ код 1, а $-B_r$ —код 0. Следовательно, для записи двоичной цифры в ферромагнитный сердечник достаточно создать в нем с помощью внешнего поля намагниченность определенного знака.

Но информацию надо не только записывать и хранить, но и считывать; расскажем, как это делается. При всяком изменении магнитного потока Ф сердечника в его обмотке в соответствии с законом электромагнитной индукции возникает электродвижущая сила, величина которой пропорциональна скорости изменения магнитного потока и числу витков:

$$e = -kw\frac{d\Phi}{dt} = -\frac{kw}{S}\frac{dB}{dt}. \qquad (1)$$

Если замкнуть концы обмотки, то в ней потечет ток, причем его направление определяется законом Ленца: магнитное поле этого тока будет препятствовать изменению магнитного потока Ф, что и выражает знак минус в приведенной формуле (1).

Итак, для считывания информации нужны две обмотки. В одну из них — это может быть та же, которая применяется для записи — подается ток, изменяющий магнитный поток сердечника. Другая «воспринимает» это изменение, и по нагрузке, включенной между ее концами, проходят импульсы тока. Подадим в первую обмотку достаточно большой по амплитуде импульс отрицательной полярности. Тогда с выходной обмотки мы снимем два импульса взаимно противоположной полярности, соответствующие началу и концу

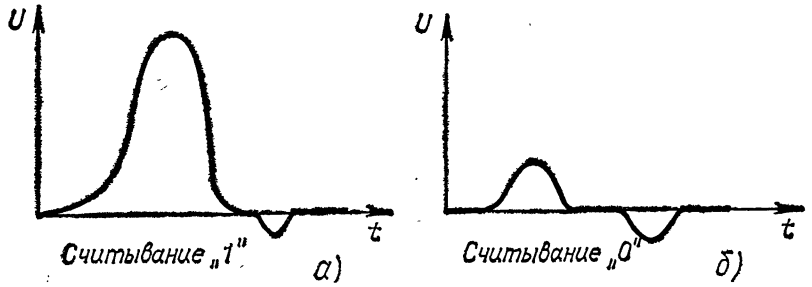


Рис. 60

считывающего импульса (переднему и заднему фронту).

Предположим, что в сердечнике была записана «1». Передний фронт считывающего импульса вызывает изменение магнитной индукции от B_r до $-B_s$, т. е. на величину $\Delta B_1=|B_r+B_s|$. Задний фронт вызовет изменение от $-B_s$ до $-B_r$, т. е. на величину $\Delta B_2=|B_s-B_r|$. Так как $\Delta B_1>\Delta B_2$, то амплитуда первого импульса будет больше амплитуды второго (на величину $2|B_r|$, рис. 60, *а*). Если же в сердечнике был записан «0», то передний фронт вызовет изменение магнитной индукции от $-B_r$ до $-B_s$, а задний — от $-B_s$ до $-B_r$, так что в обоих случаях это изменение будет равно $\Delta B_3=|B_s-B_r|=\Delta B_2$. Это означает, что оба импульса будут равны по амплитуде (рис. 60, *б*). Следовательно, по разнице в амплитудах снимаемых импульсов можно определить, что записано на сердечнике.

Эти обстоятельства определяют требования к сердечникам, вернее, к параметрам их петли гистерезиса, для применения их в запоминающих устройствах. Во-первых, должна быть достаточно велика остаточная индукция B_r, определяющая величину выходного импульса. Во-вторых, нужно, чтобы B_r было достаточно велико по сравнению с индукцией насыщения, так как при $B_r\approx B_s$ получается $\Delta B_3\approx 0$. Для этого форма петли гистерезиса должна быть как можно ближе к *прямоугольной*. Наконец, в-третьих, так как выделяемое при перемагничивании тепло, оказывающее вредное влияние на работу электронных схем, пропорционально площади петли, то эта петля должна быть по возможности узкой, поскольку её высота определяется B_r и ее уменьшать нельзя.

Описанными свойствами обладают *ферриты*— диэлектрики с ферромагнитными свойствами, изготовляемые из порошкообразной окиси железа путем автоматического спрессовывания и обжига. Петля гистерезиса ферритового сердечника приведена на рисунке 61. Как видно из рисунка, различие между B_s и B_r здесь совсем невелико. Это различие обычно характеризуется *коэффициентом прямоугольности*

$$k=\frac{B_r}{B_m}, \qquad (2)$$

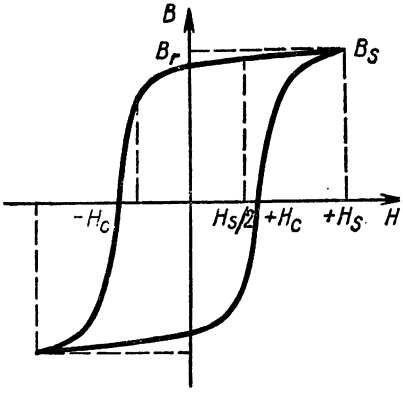


Рис. 61

где B_m соответствует напряженности внешнего поля, равной пятикратной коэрцитивной силе, $H_m=5H_c$. У ферритов коэффициент прямоугольности имеет значение порядка 0,9÷0,99. Даже при величине H, близкой $\pm H_s/2$, индукция B сохраняет значение, мало отличающееся от B_r. Только при величине H, близкой $-H_c$, происходит быстрое перемагничивание, и магнитная индукция очень быстро (при совсем небольших изменениях H) достигает значения $-B_m$, а после прекращения действия внешнего поля сохраняет постоянное значение $-B_r$.

Кроме коэффициента прямоугольности, важной характеристикой сердечника является также *время перемагничивания*, т. е. время перехода из одного устойчивого состояния в другое. Оно определяется приближенным равенством

$$\tau_п \approx S/(H_0 - H_c), \qquad (3)$$

где S — фиксированный *коэффициент перемагничивания*, определяемый свойствами материала, H_c — коэрцитивная сила и H_0 — напряженность воздействующего на сердечник поля. Из формулы (3) видно, что время перемагничивания будет малым, если выбрать материал с малым коэффициентом перемагничивания и небольшой коэрцитивной силой и воздействовать на него полем с большой напряженностью. Однако при увеличении напряженности H_0 воздействующего поля следует соблюдать осторожность: чем больше амплитуда тока и чем выше частота переключения, тем сильнее разогревается сердечник, а повышение температуры уменьшает коэрцитивную силу и остаточную индукцию ферритов и ухудшает коэффициент прямоугольности. Поэтому возможности повышения H_0 ограничены.

Время перемагничивания зависит также от т о л щ и н ы сердечника. В первых запоминающих устройствах применялись ферритовые сердечники с наружным диаметром $\approx$2,1 *мм* и внутренним $\approx$1,2 *мм* и временем переключения 1—3 *мксек* ($1\div3\cdot10^{-6}$ *сек*). В современных запоминающих устройствах используются сердечники с наружным диаметром $\approx$0,51 *мм*, а внутренним $\approx$0,31 *мм* и временем переключения порядка 0,2 *мксек*.

Обмотка таких сердечников, естественно, занятие непростое. Обычно вместо обмотки используются просто проводники, продетые через сердечник. Их бывает несколько, чаще всего 2 или 4; часто их называют *шинами*.

§ 4. ОПЕРАТИВНАЯ ПАМЯТЬ НА ФЕРРИТОВЫХ СЕРДЕЧНИКАХ

Существуют различные системы построения магнитных оперативных запоминающих устройств на ферритовых сердечниках, отличающиеся принципами формирования сигналов, способами расположения сердечников и схемами их соединения. Наибольшее распространение получили *матричные* и *линейные* запоминающие устройства.

В *матричной памяти* ферритовые сердечники собраны в плоские двумерные группы — *матрицы*. Каждая матрица объединяет сердечники, хранящие информацию одного разряда всех ячеек памяти. Общее число сердечников в матрице равно количеству ячеек памяти, а число матриц — количеству разрядов в одной ячейке. Таким образом, в машине, имеющей $2^{12}=4096$ ячеек памяти с 45-разрядным машинным словом, оперативная память состоит из 45 матриц по 4096 сердечников в каждой. Совокупность всех матриц памяти обычно называют *кубом памяти*.

Через каждый сердечник матричной памяти проходят четыре провода. Два из них, x и y, являются общими для горизонтальных и вертикальных рядов сердечников в каждой матрице и называются *координатными*. Два других провода проходят отдельно через все сердечники в матрице. Их называют шинами *считывания* и *запрета*.

При записи информации в некоторую ячейку сердечник данной матрицы, в который следует произвести запись, определяется с помощью координатных шин по содержимому регистра адреса. Например, для матрицы 4×4 с 16 сердечниками, изображенной на рисунке 62, для записи в сердечник № 10 следует воспользоваться координатными шинами x_3 и y_2. Прежде всего сердечник переводится в нулевое состояние. Для этого по каждой из координатных шин подается отрицательный импульс $-I_m/2$. Эти полутоки не оказывают никакого влияния на сердечники 9, 11, 12, а также 2, 6, 14, через которые они проходят, так как амплитуды $I_m/2$ недостаточно для перемагничивания. Однако через сердечник 10 пройдут оба полутока, так что этот сердечник испытает воздействие тока $-I_m$, а этого достаточно для приведения его в нулевое состояние. Теперь сердечник готов для записи новой информации.

Для записи «1» по тем же координатным шинам подаются импульсы полутоков положительной полярности с амплитудой $I_m/2$. В сердечнике, находящемся на их пересечении, и только в нем, эти импульсы создадут поле напряженностью H_m, которое и перемагнитит сердечник в единичное состояние. Для записи «0» можно было вообще не посылать импульса записи, но это неудобно для схемного осуществления. Обычно эти импульсы тоже посылаются, но одновременно с ними по шине запрета посылается импульс отрицательной полярности $-I_m/2$. Он компенсирует один из положительных полутоков записи, и результирующее поле окажется недостаточ-

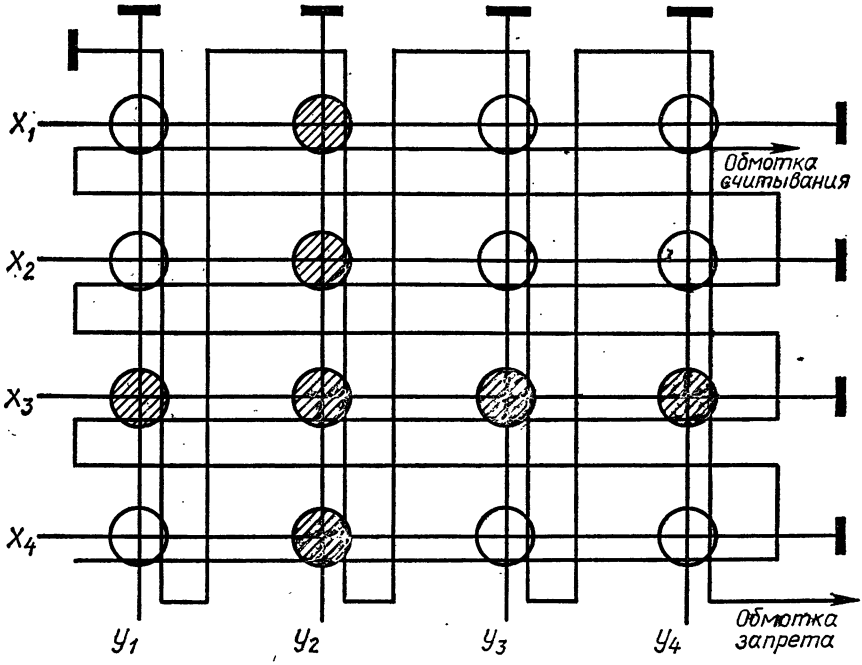


Рис. 62

ным для перемагничивания сердечника, так что он останется в нулевом состоянии.

Считывание информации производится описанным в § 3 способом: по координатным шинам x_i и y_j подаются отрицательные импульсы $-I_m/2$. Если сердечник, находящийся на их пересечении, был в состоянии «1», то он перемагнитится в состояние «0». В считывающей шине возникнут два импульса, один — достаточно большой, второй — близкий к нулю, что и означает «1»: первый импульс переведет триггер соответствующего разряда регистра числа, куда производится выборка, в единичное состояние. Если же сердечник находился в состоянии «0», то он останется в том же состоянии, а в считывающей шине возникнут одинаковые импульсы, близкие к нулю, которые не смогут перевернуть установленного в нулевое состояние триггера разряда регистра.

Заметим, что в процессе считывания выбранная информация разрушилась и ее нужно восстановить. Для этого после считывания по тем же координатным шинам подаются импульсы $+I_m/2$. Одновременно на шину запрета подается импульс $-I_m/2$ через схему совпадения, второй вход которого соединен с инверсным выходом триггера, на который считана информация. Если считана «1», то с инверсного выхода будет подан «0» и отрицательный импульс на шину запрета не пройдет. Поэтому на сердечнике, находящемся на пере-

сечении выбранных координатных шин, полутоки сложатся, и он перемагнитится снова в состояние «1». Если же был считан «0», то с инверсного выхода триггера на схему совпадения будет подана «1»; отрицательный импульс пройдет на шину запрета и скомпенсирует один из записывающих полутоков, так что сердечник останется в состоянии «0». Таким образом, правильное восстановление считанной информации будет обеспечено.

До сих пор мы говорили только об одной матрице, но ясно, что такая процедура записи или выборки относится ко всему кубу; все матрицы ферритового куба идентичны. Каждая из матриц имеет свои шины считывания и запрета и свои цепи записи и восстановления информации. Координатные же шины являются общими для всех матриц. Сердечники, являющиеся разрядами выбранной ячейки, лежат на пересечении одних и тех же координатных шин всех матриц и обращение ко всем сердечникам одной ячейки происходит одновременно.

Ферритовый куб составляет основную часть магнитного оперативного запоминающего устройства; кроме него, устройство содержит еще целый ряд электронных схем. О некоторых из них мы уже упоминали в § 2. К ним относится *регистр адреса*, служащий для приема адреса из устройства управления; *дешифратор адреса*, определяющий по заданному адресу номера координатных шин, на пересечении которых расположены требуемые сердечники, *регистр числа*, предназначенный для кратковременного хранения машинного слова, считанного из памяти или подлежащего записи в память. Кроме того, необходимы схемы для формирования импульсов чтения, записи и запрета и для усиления считанных сигналов.

Работа матричной памяти осложняется помехами, основная причина которых — полутоки, идущие по координатным шинам. Хотя их амплитуда недостаточна для того, чтобы вызвать существенные изменения магнитной индукции в сердечниках, через которые они проходят, все же вследствие отличия петли гистерезиса от идеальной небольшие изменения индукции происходят. Во время считывания в квадратной матрице $m \times m$ сердечников только один из них при считывании «1» полностью перемагнитится, но действию полутоков подвергнутся $2(m-1)$ из них. Если считывающая шина проходит через все сердечники в одном направлении, как на рисунке 62, то суммарная э. д. с., порожденная ими, может оказаться очень большой, что помешает распознаванию считанного сигнала.

Некоторой компенсации таких помех (заметной, но, к сожалению, неполной) можно достичь, прошивая сердечники считывающей шиной по диагонали, как это показано на рисунке 63. Соседние сердечники на одной координатной шине оказываются при этом прошитыми считывающей обмоткой в противоположных направлениях, и «мешающие» э. д. с. оказываются имеющими противоположные направления.

Описанную схему организации памяти часто называют *трехмерной* или *3D-схемой* (от английского *dimension* — размерность). Кроме трехмерных матричных схем в некоторых машинах применяются также *двумерные схемы* (*2D-схемы*), которые называют *линейными*.

Линейная память отличается от матричной, прежде всего, тем, что здесь функции выбора и функции хранения информации разде-

лены: выбор ячейки по заданному адресу производится «в стороне», независимо от сердечников, в которых записана информация. Считывающий импульс воздействует только на сердечники выбранной ячейки, а все остальные сердечники не подвергаются действию импульсов. Это позволяет использовать для выборки импульсы полной амплитуды, что обеспечивает сокращение времени обращения и получение большей амплитуды выходных сигналов.

Выбор ячейки памяти по заданному адресу осуществляется с помощью матрицы *координатных трансформаторов*, отличающейся от рассмотренной выше тем, что на пересечении координатных шин находятся не запоминающие сердечники, а сердечники трансформаторов значительно бо́льших размеров, чем запоминающие. Через сердечники координатных трансформаторов проходят, кроме координатных шин x и y, общая им всем *шина смещения*, а также *выходная шина* (или *выходная обмотка*) Z, которая проходит через запоминающие сердечники, служащие для запоминания машинного слова. Эти сердечники и образуют *ячейку памяти*; их называют также *числовой линейкой*. Схема линейного устройства памяти изображена на рисунке 64.

Протекающий через шину смещения постоянный ток смещения удерживает сердечники координатных трансформаторов в состоянии отрицательной намагниченности, равной (или почти равной) состоянию насыщения. Выбор ячейки осуществляется посылкой по координатным шинам, на пересечении которых находится координатный сердечник требуемой ячейки, двух полутоков, совпадение которых приведет к перемагничиванию одного координатного сер-

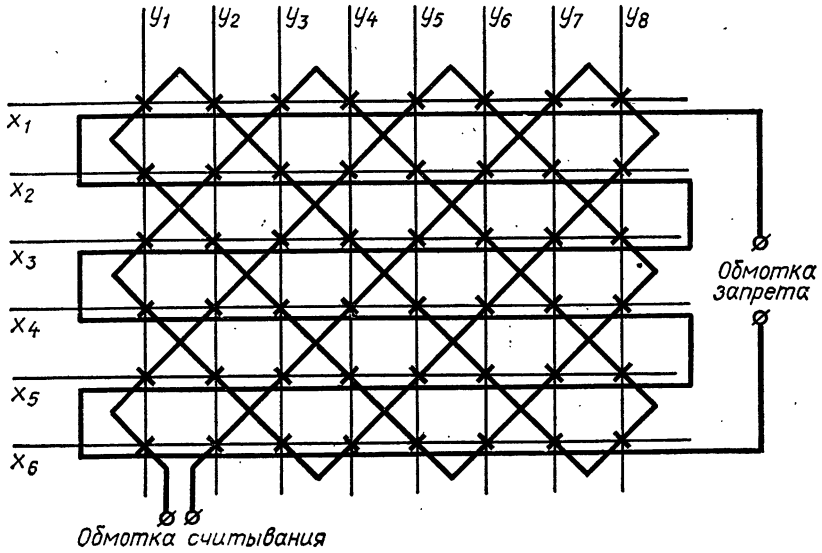


Рис. 63

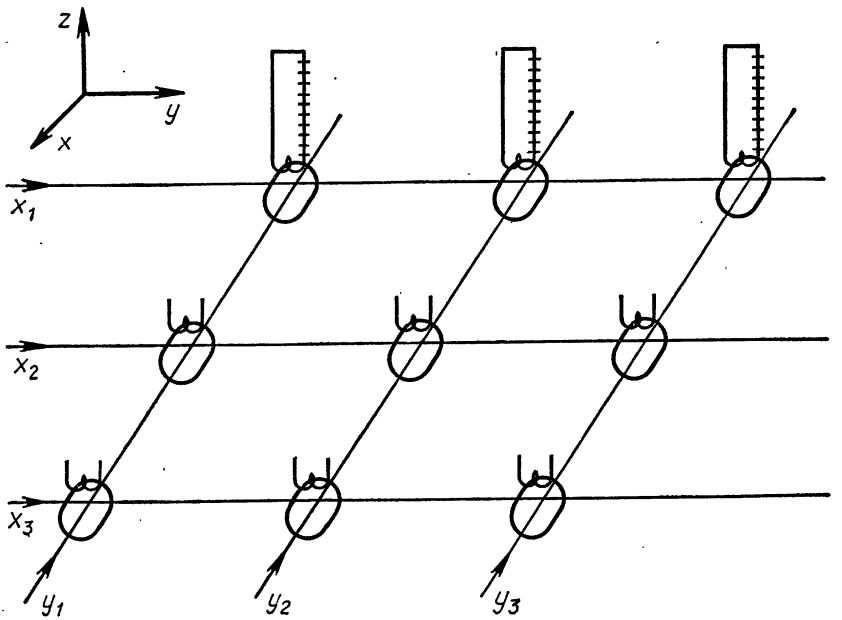


Рис. 64

дечника. При этом в выходной обмотке Z сердечника образуются два импульса разной полярности. Первый из них используется для считывания информации с числовой линейки, а второй — для восстановления считанной информации или записи новой. Естественно, что через сердечники числовой линейки должны проходить не показанные на схеме шины считывания, связанные с регистром числа на выходе устройства памяти. Так как считывающий импульс подается только в одну ячейку (в отличие от матричной памяти, где считывающий импульс проходит через все сердечники, хранящие информацию), то его амплитуда может быть выбрана достаточно большой.

§ 5. ВНЕШНЯЯ ПАМЯТЬ

Объем оперативной памяти цифровых вычислительных машин, обладающей произвольной выборкой, обычно ограничен величиной 16 $k \div 32$ k. Этого, как правило, достаточно для решения отдельных задач из области научно-технических расчетов, но может не хватить при *мультипрограммном режиме* (см. § 8 гл. VII). Аналогичная ситуация может сложиться при решении задач экономического характера или задач, связанных с планированием, для которых характерно очень большое количество входных данных при сравнительно небольшом объеме переработки информаций. Поэтому вычислительные машины обязательно снабжаются устройст-

вами памяти большого объема, хотя и невысокого быстродействия, с периодической или последовательной выборкой, которые объединяются общим названием *внешней памяти*. Сюда относятся, как мы уже говорили в § 2, магнитные ленты, магнитные барабаны и магнитные диски. Принципам их работы и посвящен настоящий параграф.

В качестве носителя информации здесь используется тонкий (10—30 микрометров *) слой лака, содержащего частицы ферромагнитного порошка, который состоит из кристаллов окиси железа Fe_2O_3. Этот *ферролак* наносится на ленту из немагнитного материала или поверхность металлического цилиндра или диска. Работа с таким запоминающим устройством состоит в намагничивании о т д е л ь н ы х у ч а с т к о в ферромагнитного покрытия или воспроизведении произведенной записи. Таким образом, принципиально работа с такой памятью не отличается от работы с хорошо известными *магнитофонами*.

Как и в магнитофонах, запись и считывание информации осуществляются с помощью записывающих и считывающих *магнитных головок*. Магнитная головка представляет собой кольцевой электромагнит, сердечник которого собран из тонких, изолированных друг от друга ферритовых пластин. Иногда вместо феррита применяется также железоникелевый сплав, называемый *пермаллоем*. Сердечник имеет воздушный зазор шириной несколько микрометров и находится на расстоянии 20—50 *мкм* от носителя информации.

При протекании тока через обмотку головки в ее сердечнике возникает магнитный поток. Он «сконцентрирован», главным образом, в сердечнике и замыкается через воздушный зазор головки и лишь небольшая его часть (так называемый *поток рассеивания*) — через магнитное покрытие носителя. Эта часть и является тем внешним магнитным полем, которое используется для намагничивания носителя. Направление потока, а значит, и направление намагниченности носителя определяется направлением тока в обмотке сердечника. После прекращения тока участок поверхности носителя, находившийся под головкой, остается намагниченным за счет остаточной индукции. Образуется как бы элементарный магнитик — *магнитный диполь* — одного из двух возможных направлений. Одному из направлений можно поставить в соответствие код «1», а другому — «0».

Если перемещать носитель информации относительно неподвижной головки, а через головку пропускать импульсы тока различной полярности, соответствующие кодам «1» и «0», то получим магнитную дорожку — последовательное расположение магнитных диполей, соответствующих записанному машинному слову. Обычная *плотность записи*, т. е. число диполей, которые можно разместить на единице длины дорожки, составляет от 2 до 15 единиц на 1 *мм* дли-

---

* М и к р о м е т р (*мкм*, прежнее название — *микрон*) равен тысячной доле миллиметра.

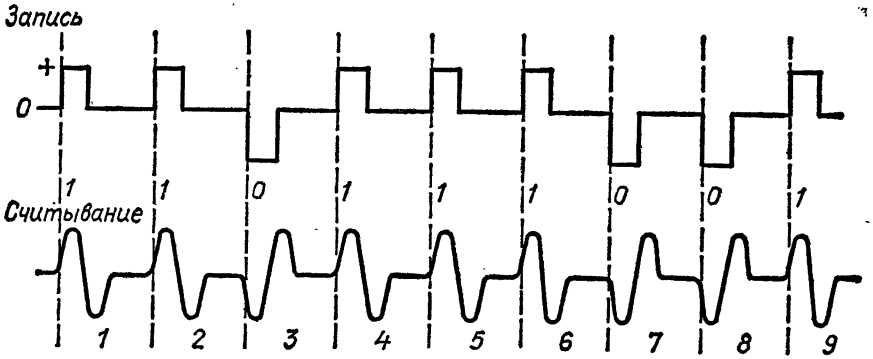


Рис. 65

ны. По ширине носителя можно разместить несколько магнитных дорожек, расстояние между осями которых 1—3 *мм*.

Чтение записанной информации можно осуществить перемещением с той же скоростью, что и при записи, носителя относительно магнитной головки, которая в данном случае будет уже с ч и т ы в а ю щ е й. Перемещение магнитного диполя вызывает переменный магнитный поток, обусловливающий появление э. д. с. в обмотке головки, причем полярность э. д. с. определяется полярностью диполя. Снимаемый с обмотки сигнал после некоторых преобразований используется в соответствующих цепях машины. При таком считывании информации не происходит ее разрушения и чтение можно производить повторно без предварительного восстановления.

Описанный способ записи и считывания называется *записью с возвращением к нулю*, так как поверхность носителя предварительно полностью размагничивается. Для записи каждой двоичной цифры используются импульсы различной полярности (рис. 65), а магнитная поверхность намагничивается в противоположных направлениях. Недостатком этого метода является необходимость предварительного размагничивания носителя, которое делается с помощью специальной дополнительной головки. Поэтому чаще применяется метод *записи с возвращением к смещению*. Здесь в обмотку магнитной головки подается постоянный ток смещения. Создаваемый им магнитный поток насыщает магнитный носитель в определенном направлении. Для записи «1» в обмотку подается импульс тока, насыщающий носитель в противоположном направлении. Коду «0» соответствует отсутствие записи.

Благодаря наличию смещения отпадает необходимость предварительного размагничивания носителя. Кроме того, вдвое увеличивается импульс чтения — амплитуда сигнала, наведенного в головке при считывании. Оба эти метода являются разновидностями *записи с промежутками*. В связи с промежутками плотность записи оказывается невысокой. Для уплотнения записи часто используется способ *записи без возвращения к нулю* или *запись без промежут-*

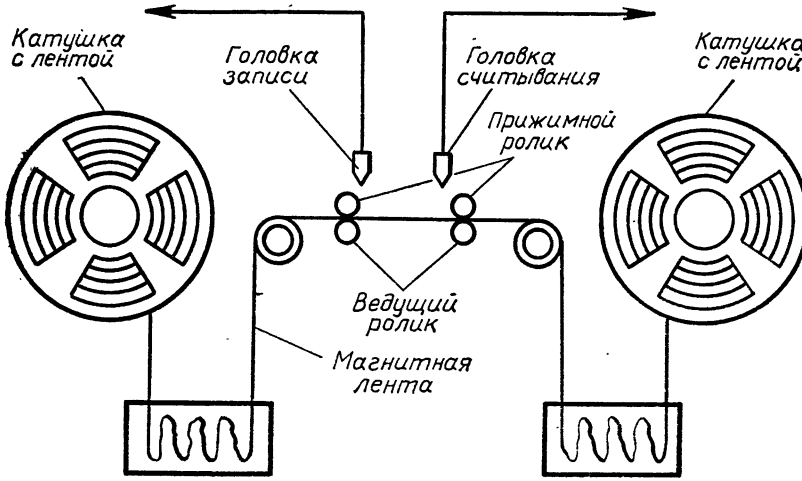


Рис. 66

ков. При этом способе изменение токов в обмотке записывающей головки производится только при переходе от записи «1» к записи «0». Следовательно, переход носителя от одного магнитного состояния к другому происходит реже, чем при записи с промежутками, благодаря чему запись получается плотнее.

Сказанное выше в равной степени относится к любым носителям информации, применяемым в устройствах внешней памяти,— лентам, барабанам, дискам. Теперь мы кратко остановимся на особенностях каждого из этих устройств.

Магнитная лента — длинная гибкая полоса из немагнитного материала (ацетилцеллюлозы, поливинилхлорида и т. д.) с магнитным покрытием — приводится в движение *лентопротяжным механизмом* (рис. 66). Она движется только во время считывания или записи. *Стартстопный механизм*, обеспечивающий быстрый пуск, протягивание мимо головок с постоянной скоростью, остановку и реверс (изменение направления движения на противоположное), состоит из непрерывно вращающихся в противоположные стороны *ведущих роликов*, *прижимных роликов* и *тормозного устройства*. При пуске ленты освобождается тормоз и нужный прижимной ролик прижимает ленту к своему ведущему валу. При остановке прижимной ролик отходит от ведущего и включается тормоз.

Головки располагаются по одной линии поперек направления движения ленты. Каждой головке соответствует дорожка записи на ленте. Некоторые дорожки используются для записи служебной информации — синхронизирующих импульсов, признаков начала и конца зоны или ее адреса, а также контрольных разрядов. Число головок определяется шириной ленты и способом размещения информации. Лента разбивается на *зоны*, между которыми оставляются свободные участки для разгона и торможения. Обмен информацией

между магнитной лентой и оперативной памятью машины обычно происходит целыми зонами. Зоны адресуются, т. е. им присваивается адрес, одним из двух следующих способов. Первый заключается в последовательном отсчете адресов вдоль ленты. В машине всегда фиксируется номер использованной зоны, так что при получении нового адреса происходит перемотка ленты в ту или иную сторону. При втором способе адресации в начале каждой зоны записывается ее адрес и тогда происходит считывание и сравнение адресов до совпадения с нужным.

Магнитный барабан представляет собой цилиндр диаметром от 100 до 800 *мм* из немагнитного материала (например, латуни) с ферромагнитным покрытием. Он вращается двигателем с постоянной скоростью от нескольких сотен до нескольких тысяч оборотов в минуту. Его поверхность разделена на дорожки, образующие кольцевые зоны на поверхности. Каждой дорожке соответствует магнитная головка, осуществляющая запись или считывание; блок магнитных головок располагается по образующей барабана (рис. 67).

Информация может размещаться на поверхности барабана различным образом. В одном случае каждая дорожка служит для записи одного разряда. Тогда все разряды одного числа записываются или считываются одновременно; такую запись называют параллельной. В другом случае — при последовательной записи — все разряды одного числа записываются на одной дорожке. Чаще всего применяется комбинированная запись: поверхность барабана разбивается на группы, по нескольку дорожек в каждой. На каждой из дорожек одной группы записывается одновременно одинаковое количество разрядов машинного слова (рис. 68).

Адресация барабана выполняется, как и в оперативной памяти. Для нее используются специальные адресные дорожки, на которых записаны соответствующие адреса.

Внешняя память на магнитных барабанах имеет несколько меньшую емкость, нежели память на магнитных лентах, но зато здесь

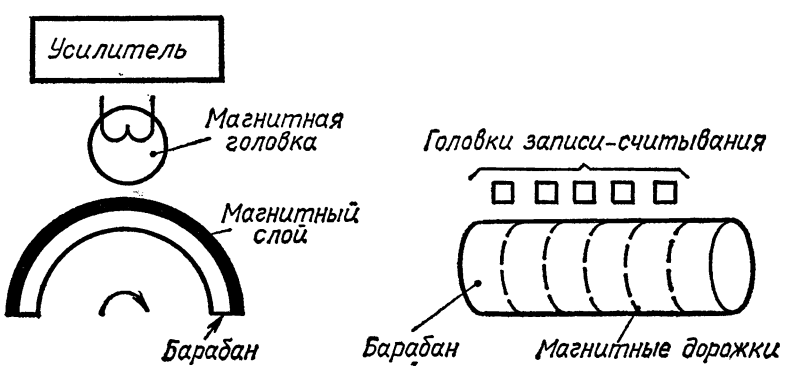


Рис. 67

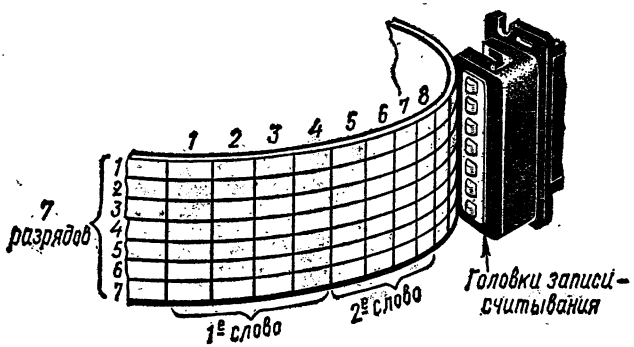


Рис. 68

значительно меньше среднее время выборки. Поэтому барабаны применяются в качестве устройств внешней памяти достаточно широко. Впрочем, в последнее время они начали вытесняться магнитными дисками, имеющими в том же физическом объеме гораздо большую рабочую поверхность и, благодаря этому, значительно большую емкость.

Носителем информации в *запоминающем устройстве на магнитных дисках* является набор дисков, изготовляющихся из легкого немагнитного материала с ферромагнитным покрытием. Информация записывается на поверхности дисков по дорожкам, представляющим собой концентрические окружности, причем обе поверхности являются рабочими (рис. 69). Диски устанавливаются на общий вал и приводятся во вращение электродвигателем. Головки записи-считывания (по одной на каждую рабочую поверхность) установлены на пружинных рычагах и перемещаются в зазорах

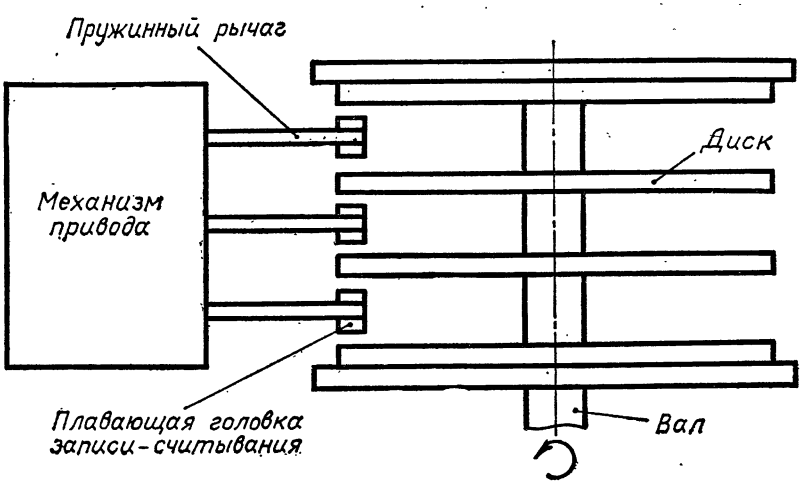


Рис. 69

между дисками. Рычаги укреплены на механизме привода, с большой точностью преобразующего цифровой код номера дорожки в соответствующее перемещение рычагов с головками вдоль радиуса диска, так, что головка попадает на нужную концентрическую дорожку. Ввиду различия в линейной скорости различных участков поверхности, плотность записи на различных дорожках оказывается различной, тем не менее объем информации, помещающейся на дисках, достаточно велик. Обычный диаметр диска 500—600 *мм*, расстояние между ними — порядка 10 *мм*.

§ 6. УСТРОЙСТВА ВВОДА И ВЫВОДА

Устройства ввода и вывода, как и рассмотренные в предыдущем параграфе устройства внешней памяти, относятся к так называемым *внешним* или *периферийным* устройствам цифровой вычислительной машины. При вводе информация с языка, привычного человеку (комбинации арабских цифр, латинских и русских букв), переводится на язык, понятный машине (комбинации двоичных кодов, последовательности электрических импульсов и др.). Обратное преобразование происходит при выводе.

Процедура ввода предполагает последовательное использование двух независимых устройств: *устройства подготовки данных*, переносящего входную информацию на первичный носитель, и *читающего устройства* — собственно устройства ввода, переносящего информацию с первичного носителя в оперативную память машины.

Основными первичными носителями информации для цифровых вычислительных машин являются *перфокарты* и *перфоленты*. Перфокарта (рис. 70) представляет собой прямоугольный лист тонкого эластичного картона стандартных размеров со срезанным одним углом, что позволяет ориентировать ее нужным образом во входных устрой-

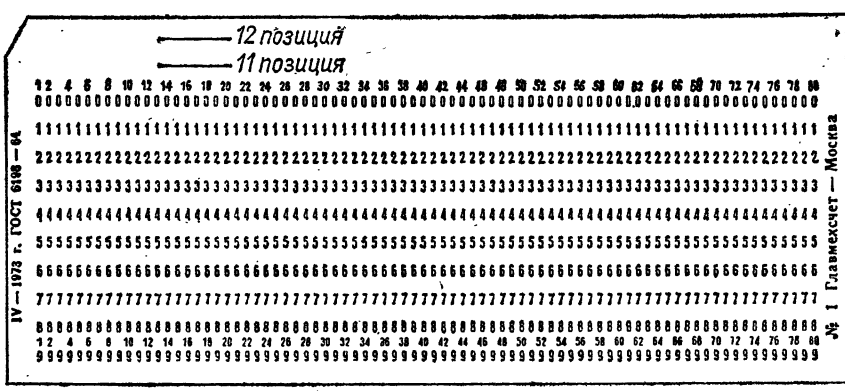


Рис. 70

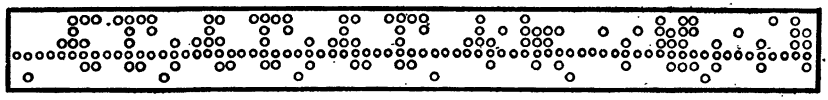


Рис. 71

ствах. Поле перфокарты разбито на 12 строк; десять из них помечены цифрами от 0 до 9 и называются *нулевой позицией*, *первой позицией* и т. д.; 11 и 12 позиции расположены сверху, над нулевой. Каждая строка содержит 80 колонок, которые занумерованы от 1 до 80; номера четных колонок напечатаны мелким шрифтом сверху и снизу перфокарты.

Наличие отверстия в каком-либо месте перфокарты означает «1», отсутствие отверстия — «0». Способ размещения информации на перфокарте различен не только для разных машин, но и для разных читающих устройств одной и той же машины, и мы не будем здесь на этом останавливаться. Этот вопрос будет подробно рассмотрен в § 3 следующей главы.

Кроме перфокарт, в качестве первичного носителя информации нередко используется перфолента — плотная бумажная или целлулоидная лента, в центре которой пробиты синхронизирующие отверстия. Чаще всего используется пятидорожечная бумажная телеграфная лента, употребляющаяся в буквопечатающих телеграфных аппаратах (телетайпах). Против каждого синхронизирующего отверстия на ней имеется место для 5 пробивок, различными комбинациями которых и изображаются привычные символы (рис. 71). Способы изображения символов пробивками на перфоленте также будут рассмотрены в § 3 гл. V.

Перенос данных на первичные носители производится при помощи *клавишного устройства* и *входного перфоратора*, действующих солидарно. На клавиатуре клавишного устройства нанесены те же символы, что и на бланке, написанном программистом: цифры, знаки + и — или буквы и другие символы, в зависимости от системы работы. Нажимая соответствующие клавиши, оператор обеспечивает выработку сигналов, направляемых по кабелю во входной перфоратор, который под влиянием этих сигналов и пробивает в нужных местах отверстия.

Читающее устройство строится на одном из двух возможных принципов: электромеханический ввод и фотоввод (рис. 72). При электромеханическом вводе (рис. 72 *а*, *б*) отверстие в носителе, перфоленте или перфокарте позволяет замкнуть контакт электрической цепи, в результате чего возникает электрический импульс, соответствующий коду «1». Отсутствие отверстия влечет отсутствие контакта, а значит, и отсутствие импульса, что соответствует коду «0». В фотовводе (рис. 72 *в*) импульс возникает в цепи фотоэлемента, воспринимающего свет от источника, попадающий через отверстие. Если отверстия нет, то луч света прерывается и импульс не возни-

кает. Скорость электромеханического ввода составляет до 700 перфокарт в минуту. Фотоввод работает быстрее.

В качестве *выходных устройств* цифровых вычислительных машин используются, главным образом, выходные перфораторы и *печатающие устройства*. Выходные перфораторы мало отличаются от входных. Скорость вывода составляет примерно 50—60 карт в минуту. Нужно, однако, отметить, что информация на перфокартах трудна для непосредственного восприятия человеком. Поэтому вывод на перфорацию используется обычно либо в тех случаях, когда полученные данные будут служить входными для другой задачи, т. е. должны будут снова вводиться в машину, либо тогда, когда полученный материал снова предполагается позже печатать на автономном печатающем устройстве, чтобы не загружать печатающее устройство машины. В остальных случаях выходные данные выводятся непосредственно на печать.

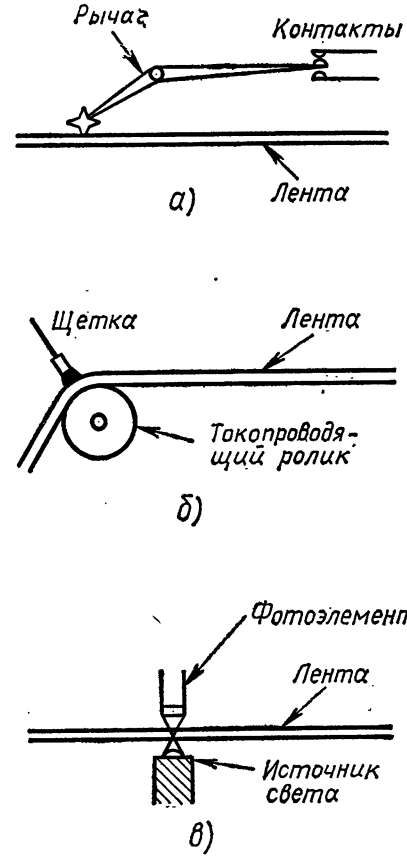


Рис. 72

С точки зрения «читателя» существует два типа печатающих устройств: *быстродействующая печать (узкая)* и *алфавитно-цифровая печать (широкая)*. В первом случае выходные данные печатаются на узкой бумажной ленте, по 16 символов в строке, причем в числе символов могут быть только цифры и знаки +, —. Узкая печать работает со скоростью 20 строк в секунду, но может выводить только числовой материал. Алфавитно-цифровое печатающее устройство печатает выходные данные по 128 символов в строке со скоростью около 400 строк в минуту, при этом используются свыше 100 различных символов, в том числе и знаки арифметических действий и буквы русского и латинского алфавитов. Алфавитно-цифровые печатающие устройства появились только на машинах второго поколения.

Конструктивно печатающие устройства делятся на ударные и безударные. Ударные печатающие устройства в свою очередь делятся на устройства *барабанные* и *ленточные*. На машинах первого

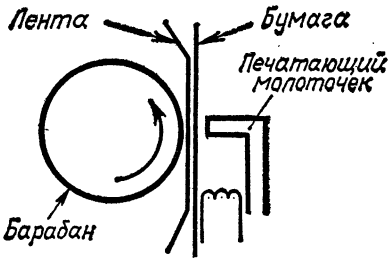


Рис. 73

и второго поколений использовались в основном ударные барабанные печатающие устройства, поэтому мы остановимся на них немного подробнее.

Основой печатающего устройства барабанного типа является алфавитно-цифровой (или — для узкой печати — цифровой) барабан, состоящий из набора печатающих колес. На каждом колесе в виде выпуклых литер изображены все печатаемые символы. Число колес равно числу символов в печатаемой строке: 16 для узкой и 128 для широкой печати. Колеса закрепляются на оси так, что одинаковые литеры располагаются по образующим барабана.

Механизм печати содержит набор молоточков, укрепленных над поверхностью барабана; их число равно числу колес. Каждый молоточек имеет свой управляющий электромагнит. В исходном положении все молоточки отведены от бумажной ленты, которая вместе с красящей лентой помещена между ними и печатающим барабаном. Если в обмотку электромагнита некоторого молоточка поступает управляющий сигнал, то электромагнит притягивает свой якорь, что вызывает удар молоточка по бумажной ленте, на которой отпечатывается символ, находившийся под молоточком в момент удара (рис. 73).

Общая схема печатающего устройства показана на рисунке 74. На одной оси с печатающим барабаном насажен синхронизирующий барабан с отверстиями, расположенными по двум дорожкам вдоль образующих. На одной из дорожек два отверстия, а на другой — столько, сколько различных символов содержит печатающее устройство (на применяющемся в советских машинах алфавитно-цифровом печатающем устройстве АЦПУ-128/2 число символов равно 117). Внутри этого барабана имеется лампа, свет которой воспринимается двумя фотодиодами. За полный оборот барабана один диод выдает два импульса, соответствующих сигналам «*Начало печати*» и «*Конец печати*»; число импульсов второго диода соответствует числу возможных символов. Эти импульсы последовательно подаются в *счетчик синхронизирующих импульсов*.

Информация, подлежащая печатанию, поступает в запоминающее устройство печати, имеющее 128 семиразрядных двоичных регистров. По сигналу *Начало печати* в счетчик синхронизирующих импульсов записывается нуль, а затем его содержимое сравнивается схемой совпадения последовательно со всеми 128 регистрами запоминающего устройства печати. В каком-либо из них может находиться семиразрядный нулевой код, соответствующий символу «0». В тех случаях когда произошло совпадение, схема совпадения выдает единичный импульс, поступающий в соответствующие электромагниты, и в нужных местах строки печатаются нули. В это же

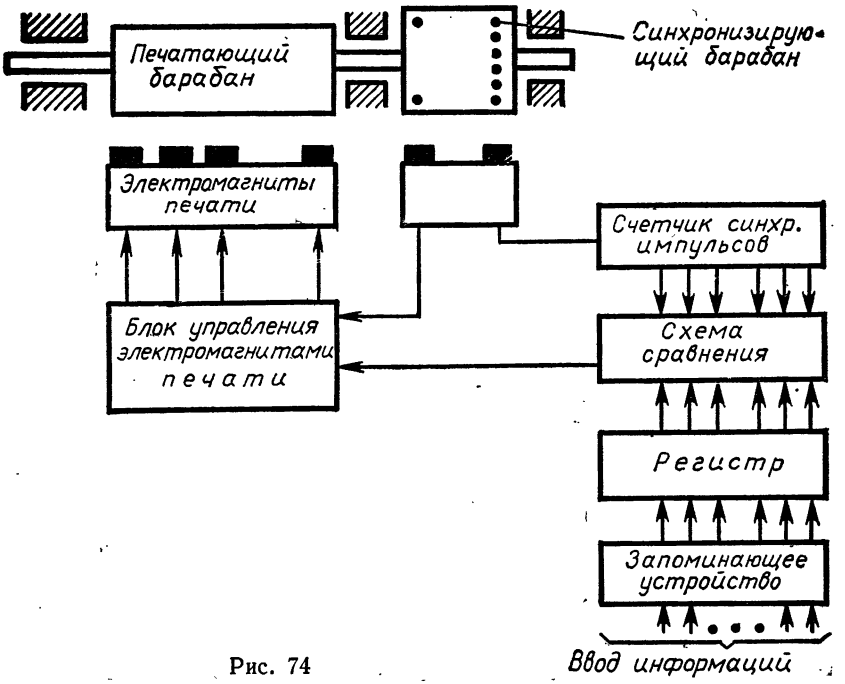


Рис. 74

Символьная строка барабана	Печать						
0							
1	1						
2	1						
3	1						
4	1			4			
5	1			4			
6	1			4			
7	1		7	4			
8	1		7	4			
9	1	9	7	4			
а	1	9	7	4			
б	1	9	7	4			
в	1	9	7	4			
г	1	9	7	4	г		
д	1	9	7	4	г		д
⋮							
0	1	9	7	4	г	0	д

Рис. 75

время с синхронизирующего барабана поступает первый синхронизирующий импульс, соответствующий повороту барабана, при котором к данной строке подошел следующий символ и снова производится сравнение счетчика синхронизирующих импульсов с регистрами запоминающего устройства печати. Аналогичный цикл длится до полного оборота барабана, после чего по сигналу *Конец печати* бумажная лента передвигается на одну строку, а в запоминающее устройство печати передается следующая информация. Процесс печати фразы «1974 год» показан выше на рисунке 75.

§ 7. УСТРОЙСТВО УПРАВЛЕНИЯ

Последнее устройство, с работой которого мы должны сейчас познакомиться,— *устройство управления* цифровой вычислительной машины. Его назначением является управление выполнением программы вычислений, что в свою очередь сводится к управлению исполнением отдельных команд и управлением их сменой.

Устройство управления содержит, прежде всего, *счетчик команд*, содержащий адрес команды, которая должна исполняться, а также *регистр команды*, куда эта команда вызывается из памяти. Для этого содержимое счетчика команд передается в регистр адреса оперативной памяти и передается сигнал на выборку содержимого данной ячейки, которое и пересылается затем из регистра числа оперативной памяти в регистр команды устройства управления. Регистр команды делится на *регистр кода операции* и *регистр адреса* (или адресов — для двух- или трехадресных машин).

Код операции подается с регистра кода операции на *дешифратор операций*, устроенный так, как это описано в § 8 гл. III. Дешифратор операций преобразует двоичное число, составляющее код операции, в выходной сигнал, подаваемый по одной выходной шине, соответствующей той именно операции, которую в данном случае следует выполнить. Управление выполнением команды состоит в том, что на определенные управляющие шины арифметического устройства, памяти или внешних устройств в определенные моменты времени и в определенной последовательности подаются управляющие импульсы. Каждой элементарной операции машины соответствует своя последовательность управляющих импульсов, реализующая свойственную данной операции последовательность *микроопераций* или *тактов*. Схемы, вырабатывающие нужную последовательность управляющих импульсов, могут быть построены на различных принципах.

Особенность *синхронного принципа управления* состоит в том, что все элементарные операции разбиваются на одинаковое количество тактов. Длительность цикла выбирается по самой продолжительной операции. Темп работы цифровой машины задается *генератором тактовых импульсов*, которые подаются на счетчик тактовых импульсов. Последний связан с дешифратором тактовых импульсов, число

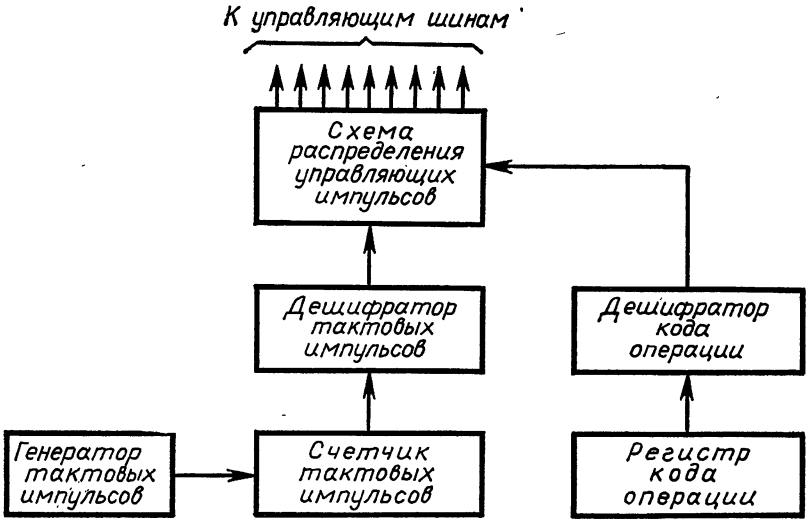


Рис. 76

выходов которого равно числу тактов в элементарной операции. В начале каждого такта дешифратор выдает импульс на соответствующем выходе. Общая структурная схема устройства синхронного управления операциями показана на рисунке 76.

Структура блока, который на рисунке 76 назван *Схема распределения управляющих импульсов*, определяется характером тактов, относящихся к выполняемой операции. При проектировании схемы управления заранее составляется таблица, в которой указывается, на какие управляющие шины в каждом такте должны быть поданы управляющие импульсы в каждой операции. При этом стараются согласовать последовательность выполнения различных операций так, чтобы импульсы на одну и ту же шину при различных операциях поступали по возможности в одном и том же такте.

При *асинхронном управлении операциями* для каждой операции используется столько тактов, сколько требуется, так что число тактов для различных операций может оказаться различным. Для каждой операции имеется отдельная схема управления, построение которой определяется характером операции. В простейшем случае схема управления имеет вид сдвигового регистра, управляемого тактовыми импульсами, с числом выходов, равным числу микроопераций, необходимых для выполнения данной команды (рис. 77). Выходы связаны с со-

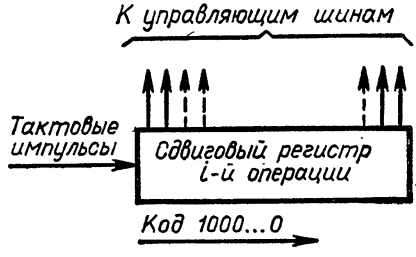


Рис. 77

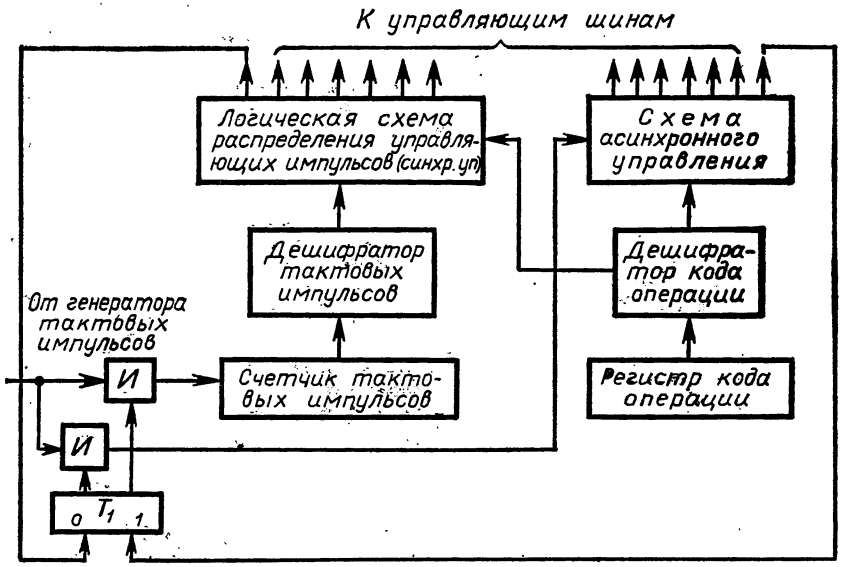


Рис. 78

ответствующими управляющими шинами. В регистр записывается единица, которая, последовательно сдвигаясь, поступает на все выходы по очереди, реализуя управление операцией. В более сложных случаях могут потребоваться какие-либо разветвления.

Очевидным преимуществом асинхронного управления операциями перед синхронным является заметное быстродействие вычислительной машины. При асинхронном управлении ни одна операция не содержит пустых тактов; каждая команда выполняется столько времени, сколько в действительности необходимо для ее выполнения. Сколь очевидно преимущество асинхронного управления, столь очевиден и его недостаток, который заключается в значительном повышении стоимости, связанным с увеличением требуемого для реализации оборудования, так как для управления каждой отдельной операцией требуется самостоятельная схема. Поэтому используют смешанную схему управления операциями, в которой применяются принципы как синхронного, так и асинхронного управления. Структурная схема такого решения представлена на рисунке 78.

Как и при синхронном, при смешанном управлении операциями выбирается стандартный по длительности цикл исполнения операций, рассчитанный, однако, не на самую продолжительную, а на «типичную» операцию, так, чтобы у большинства операций не было «пустых» тактов. При этом некоторые более длительные операции в этот стандартный цикл не укладываются. Управление ими частично возлагается на асинхронную схему управления. Чаще всего в роли

таких «не укладывающихся» операций оказываются арифметические плавающие операции, в особенности умножение и деление.

Совместная работа синхронной и асинхронной схем управления происходит следующим образом. Генератор тактовых импульсов с помощью триггера T_1 и логических ячеек *и* может подключаться либо к счетчику тактовых импульсов синхронной схемы управления, либо к асинхронной схеме управления. Один из управляющих импульсов синхронной схемы отключает генератор от счетчика тактовых импульсов и одновременно запускает некоторую асинхронную схему. Последняя начинает вырабатывать управляющие импульсы, а на счетчике запоминается номер последнего исполненного такта синхронной схемы. На некотором такте асинхронная схема вырабатывает импульс, включающий снова синхронную схему управления.

Рассмотрим теперь вопрос об управлении сменой команд. Как мы уже говорили в § 2 предыдущей главы, при естественном порядке выполнения команд в счетчик команд автоматически прибавляется единица. В случае необходимости отступления от естественного порядка содержимое счетчика команд изменяется специальной командой *безусловного* или *условного перехода*. Подробнее соответствующие команды будут рассмотрены в § 2 следующей главы. Мы ограничимся здесь лишь указанием на то, что команда безусловного перехода всегда заменяет содержимое счетчика команд на адрес, указанный в самой команде, а команда условного перехода делает это лишь при выполнении некоторого условия; при его невыполнении к счетчику команд прибавляется единица, как и при естественном порядке выполнения команд. Пример схемы, обеспечивающей такую работу, приведен на рисунке 79. Эта схема достаточно проста и в ней легко разобраться без дополнительных пояснений.

Важной составной частью устройства управления является *пульт управления и сигнализации*. Пульт любой из вычислительных машин состоит из двух панелей: горизонтальной и вертикальной. Вертикальная панель используется для сигнализации и содержит большое число лампочек различного назначения, о которых мы скажем несколько позже. Горизонтальная панель используется для управления и содержит различные кнопки, клавиши, тумблеры и переключатели. Расположение их на разных машинах весьма различно, даже, например, для таких однотипных машин, как БЭСМ-4 и М-220, имеющих почти тождественную систему команд. Поэтому мы не станем описывать расположение различных регистров и клавиш на панелях пульта, а только перечислим их на примере трехадресной машины БЭСМ-4.

Горизонтальная панель пульта содержит средства управления машиной. Опишем некоторые из них, наиболее простые и наиболее существенные.

1. Кнопка *Ввод*. При ее нажатии начинает работать читающее устройство ввода. Материал с перфокарт, стоящих на этом устрой-

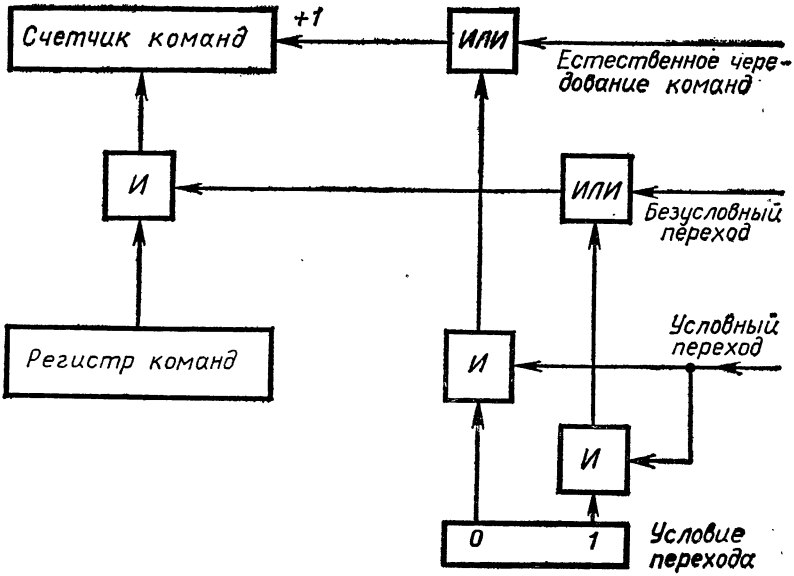


Рис. 79

стве, поступает в оперативную память машины в соответствии с набитым на карте адресным кодом. Если на карте нет адресного кода, то материал вводится начиная с ячейки 0001. Работа устройства ввода прекращается, если в вводимом материале встречается контрольная сумма (см. § 3 гл. V) или кончается колода перфокарт, стоящая на читающем устройстве.

2. Кнопка *Пуск*, нажатие которой включает работу машины, запуская генератор тактовых импульсов.

3. Кнопка *Стоп* служит для остановки работы машины.

4. Клавиши для *установки в нуль (сбрасывания)* основных регистров пульта управления.

5. *Регистр команд пульта* — ряд из 45 клавиш; на них можно набрать команду, которую нужно исполнить с пульта или передать в регистр команд устройства управления машины. При ручном управлении началом работы машины (без программы-диспетчера, о которой речь будет идти в гл. VII) работа по решению отдельной задачи обычно начинается с ввода программы и передачи управления с пульта началу этой программы.

6. *Переключатель режимов работы машины.* Он может занимать одно из трех положений: *автоматический режим*, *шаговый режим** и *одиночный режим*. При положении переключателя *автоматический режим* машина после нажатия кнопки *Пуск* выполняет программу автоматически, останавливаясь либо по команде

* На описываемой машине этот режим называется циклическим.

стоп, либо при аварийной остановке в предусмотренных случаях (например, при переполнении разрядной сетки, попытке деления на нуль и т. п.), либо при нажатии кнопки *Стоп*. При *шаговом режиме* нажатие кнопки *Пуск* вызывает лишь исполнение одной очередной команды, после чего машина останавливается; чтобы выполнить еще одну команду, нужно нажать *Пуск* еще раз. Наконец, при одиночном режиме нажатие *Пуска* приводит к выполнению одной микрооперации, т. е. лишь очередного такта. Для выполнения всей команды в одиночном режиме необходимо нажать *Пуск* столько раз, сколько тактов в этой команде. Этот режим используется инженерами для проверки правильности работы машины.

7. *Клавишные запоминающие регистры пульта управления.* Каждый из них представляет ряд из 45 клавиш, на котором можно набрать 45-разрядное машинное слово. Содержимое этих четырех регистров можно менять вручную, но не из машины. С помощью специальной команды оно может переноситься в ячейки оперативной памяти или при другом режиме выбираться из этих регистров, как из обычных ячеек оперативной памяти.

8. *Клавиши отладочных стопов*, нажатие которых дает возможность остановить работу машины в нужном месте. Возле каждой такой клавиши имеется ряд из 12 клавиш, в котором можно набрать 12-разрядное двоичное слово. Имеется три таких возможности.

Стоп по счетчику команд. При наборе некоторого адреса ячейки оперативной памяти и нажатии этой клавиши машина останавливается перед исполнением команды, находящейся в этой ячейке, т. е. в тот момент, когда содержимое счетчика команд совпадает с набранным адресом.

Стоп по записи. При наборе здесь адреса ячейки памяти машина останавливается после исполнения команды, заносящей в эту ячейку какое-либо машинное слово.

Стоп по РА. Машина останавливается в тот момент, когда содержимое регистра адреса (так называется на этой машине индексрегистр) совпадает с набранным здесь двенадцатиразрядным словом.

Эти клавиши используются при проверке правильности работы программы или машины. Кроме перечисленных клавиш и регистров, имеется также ряд других клавиш, переключателей и тумблеров, связанных с коммутацией устройств внешней памяти и управлением некоторыми другими устройствами, на чем мы не останавливаемся.

На вертикальную панель пульта управления выведено содержимое большого числа регистров машины. Прежде всего, сюда выведены регистры арифметического устройства — два *регистра исходных данных*, куда вызываются числа из оперативной памяти, *сумматор* и *регистр результата*. Все они изображаются горизонтальными рядами по 45 неоновых лампочек, причем горящая лампочка имеет значение двоичной единицы, негорящая — нуля. При

работе машины в одиночном режиме можно полностью проследить процесс выполнения команды по тактам и микрооперациям. При шаговом режиме можно таким же образом проверить результат выполнения каждой команды. Если машина работает в автоматическом режиме, то уследить за содержимым какого-либо регистра, разумеется, невозможно. Однако в этом случае есть возможность проследить за содержимым какой-либо ячейки памяти, если оно редко изменяется. Для этого достаточно набрать адрес этой ячейки на одной из клавиатур (стоп по РА) и нажать имеющуюся там клавишу *Вызов кода*. Содержимое соответствующей ячейки будет вызвано в регистр результата.

Кроме описанных регистров, на вертикальной панели пульта управления находятся также *регистр команд* и *счетчик команд*, а также *регистр адреса (индекс-регистр)*, *управляющий сигнал* ω, сигнал *авост*, указатели операций, счетчики тактовых импульсов синхронной и асинхронной схем управления и тому подобная сигнализация. Вся она используется в основном при проверке правильности работы машины, так как состояние описанных (и неупомянутых) регистров полностью характеризует состояние и работу вычислительной машины.

Глава V

ЭЛЕМЕНТЫ ПРОГРАММИРОВАНИЯ

§ 1. СИСТЕМА КОМАНД ВЫЧИСЛИТЕЛЬНОЙ МАШИНЫ. АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ОПЕРАЦИИ

Системой команд цифровой вычислительной машины называется совокупность всех элементарных операций, имеющих присвоенные им коды и доступных для выполнения при помощи команд.

Элементарные операции машины, независимо от ее адресности, можно разбить на несколько групп в зависимости от назначения операций и характера их исполнения. Все эти группы обязательно присутствуют в системе команд любой машины. Рассмотрим их.

Арифметические операции. Часто их называют *плавающими* действиями, потому что они выполняются над числами, записанными в форме с плавающей запятой. При записи в ячейке памяти числа с плавающей запятой (см. § 7 гл. III) некоторое число разрядов отводится для записи порядка числа, остальные — для записи мантиссы. Отсюда ясно, что различные разряды при этом «работают» существенно по-разному — разряды мантиссы не так, как разряды порядка, а и те и другие иначе, чем знаковые. Выполнение основных арифметических операций было разобрано в предыдущей главе.

Для трехадресной машины в команде, соответствующей арифметическим операциям, первый и второй адреса указывают ячейки, хранящие числа, над которыми совершается операция, в частности при вычитании или делении первый адрес соответственно указывает уменьшаемое или делимое. Третий адрес предназначен для записи результата. Например, команда

05 3472 1765 0203

означает, что следует перемножить (05 — код умножения) числа из ячеек 3472 и 1765 и результат поместить в ячейку 0203. Точно так же, команда

04 0101 0007 7767

означает, что число из ячейки 0101 надо разделить на число из ячейки 0007 и частное записать в ячейку 7767.

Несколько иначе обстоит дело для арифметических операций на машинах меньшей адресности. Арифметические операции двухадресной машины Минск-22 имеют по четыре модификации, причем каждая модификация имеет свой код. Например, для операции умножения имеются четыре кода: +34, +35, +36, +37 (на этой машине знак включается в код операции и код — 34 означает иную операцию, нежели + 34), которые выполняются так. Команда

+34 2567 4430

означает перемножение содержимого ячеек 2567 и 4430 с записью результата по второму адресу (в ячейку 4430). Код +35 означает умножение без записи в ячейку памяти. Результат здесь остается только в сумматоре. Команда 36 2567 4430 означает, что содержимое сумматора умножается на содержимое ячейки 2567 и результат записывается в ячейку 4430 по второму адресу. Наконец, команда с кодом +37 вообще не использует второго адреса и означает умножение содержимого сумматора на содержимое ячейки по первому адресу. Результат действия остается в сумматоре.

Таким образом, у двухадресной машины сумматор нередко выступает как один из компонентов арифметической операции. Для одноадресной машины это уже совершенно обязательно. Например, для машины БЭСМ-6 команда вида

017 2564

означает: содержимое сумматора умножить на содержимое ячейки 02564 (арифметические операции относятся к командам короткоадресной структуры, см. § 7 гл. III). Точно так же команда 005 1672 означает вычитание из содержимого сумматора содержимого ячейки 01672. На машине БЭСМ-6 имеется и обратное вычитание: команда 006 7430 означает вычитание из содержимого ячейки 07430 содержимого сумматора. Результат арифметической операции в одноадресной машине во всех случаях остается в сумматоре.

Кроме перечисленных основных арифметических операций, на всех машинах имеется операция *вычитания абсолютных величин*. Так, на трехадресной машине типа М-20, команда

03 4572 6343 0770

выполняется так: при передаче из памяти в арифметическое устройство у чисел, выбранных из ячеек 4572 и 6343, гасятся знаковые разряды, затем из первого числа вычитается второе и результат записывается в ячейку 0770. Аналогично по комане 007 3546 в машине БЭСМ-6 гасится знаковый разряд в сумматоре и в числе, вызванном из ячейки 03546 (число, остающееся в ячейке, не изменяется), затем из содержимого сумматора вычитается полученное число, т. е. абсолютная величина вызванного числа, и результат остается в сумматоре. Кроме этого, на машинах типа М-20 в число элемен-

тарных операций входит также операция извлечения квадратного корня.

Группа *фиксированных действий* относится уже к специфически машинным операциям, в основном используемым для работы с командами или для решения других программистских задач (см. § 5,6 настоящей главы). Название группы берет начало от первых машин, в которых машинное слово могло восприниматься как двоичное число с фиксированной перед старшим или, наоборот, после младшего значащего разряда запятой. Эти операции на разных машинах устроены по-разному, но мы рассмотрим их лишь для трехадресных машин типа М-20.

Ячейка памяти типа М-20 имеет 45 разрядов и делится на две части: операционную и адресную (см. рис. 44 на стр. 70). Основными фиксированными действиями являются сложение и вычитание адресных частей и сложение и вычитание операционных частей (иногда первые два называют сложением и вычитанием мантисс). Операция сложения адресных частей имеет код 13. Команда

13 2600 2612 2631

выполняется так: адресные части машинных слов, записанных в ячейках 2600 и 2612, рассматриваются как целые 36-разрядные двоичные числа с фиксированной (после первого разряда) запятой и складываются по обычным правилам. Единица переполнения, которая может возникнуть, теряется. Полученная 36-разрядная сумма записывается в адресную часть ячейки 2631. Что касается операционной части результата, то сюда без изменений переносится операционная часть ячейки 2600, т. е. ячейки из первого адреса команды.

Сложение операционных частей с кодом 53 выполняется аналогично: складываются операционные части (разряды 45—37), а адресная часть переписывается без изменения из ячейки по первому адресу.

Вычитание адресных (33) или операционных (73) частей вполне понятно; если вычитаемое окажется при этом больше уменьшаемого, то вычитание производится так, как будто есть возможность «занять» единицу старшего разряда.

Машина БЭСМ-6 не имеет фиксированных операций указанного типа. Аналогом фиксированного сложения является операция сложения содержимого индекс-регистров. Она вполне соответствует операции сложения адресных частей без переноса в операционную часть и отличается от нее лишь тем, что может быть выполнена только над содержимым индекс-регистров, а не над содержимым произвольных ячеек памяти.

Следующей группой операций являются *логические операции*, которые делятся на собственно логические и сдвиги. Характерным отличием операций этой группы от двух предыдущих является то, что при плавающих действиях содержимое ячейки памяти высту-

пает как единое целое, при фиксированных действиях — распадается на изолированные части, а при логических рассматривается как набор не связанных между собой нулей и единиц. Именно это обстоятельство и позволяет нам объединить в одну группу сдвиги и логические операции.

Логические операции рассматривались нами достаточно подробно в гл. III. Обычно в число логических операций машины входят логическое умножение (конъюнкция), логическое сложение (дизъюнкция) и отрицание равнозначности. Иногда для этих операций употребляют соответственно наименования: *выделение* (или высечение), *формирование* и *сверка*, связанные с употреблением их в стандартных подпрограммах, о чем будет идти речь в § 6 настоящей главы.

Логическое умножение состоит в поразрядном логическом перемножении соответствующих разрядов машинных слов, не отличающемся от обычного перемножения. При логическом сложении разрядов мы встречаем лишь одно правило, отличающееся от обычного сложения: $1 \vee 1 = 1$. Отрицание равнозначности (сверка) состоит в поразрядном сложении по модулю 2, или, что то же самое, в проверке цифр разряда на несовпадение: совпадение цифр дает в результате 0, несовпадение — 1, т. е. $0 \varkappa 0 = 1 \varkappa 1 = 0$, $0 \varkappa 1 = 1 \varkappa 0 = 1$. В каждом разряде эта операция производится независимо от остальных. Эти операции на всех машинах выполняются одинаково, аналогично арифметическим.

Другая часть группы логических операций — *сдвиги*. *Сдвиг* машинного слова является однокомпонентной операцией и состоит в перемещении слова по разрядной сетке на нужное количество разрядов влево или вправо, причем освобождающиеся разряды заменяются нулями. Например, 12-разрядное машинное слово 110 101 111 011 при сдвиге на четыре разряда влево будет иметь вид 011 110 110 000, а при сдвиге на три разряда вправо 000 110 101 111.

В одноадресных машинах производится сдвиг содержимого сумматора, а адресная часть команды служит для указания направления и величины сдвига. Результат остается в сумматоре. Например, машина БЭСМ-6 имеет две команды сдвига, которые называются *сдвиг по адресу* и *сдвиг по порядку числа* (или сдвиг *по коду*). В команде *сдвиг по адресу* (код 036) адресная часть команды содержит не адрес ячейки памяти, как это должно было бы быть при последовательном выполнении адресного принципа работы машины, а некоторое условное число. Младшие семь разрядов этого числа показывают направление и число разрядов сдвига. Именно здесь записывается число $100_8 + n$, где $|n|$ — число разрядов сдвига, $n > 0$ означает сдвиг вправо, а $n < 0$ — сдвиг влево. Таким образом, команда 036 00115 означает сдвиг содержимого сумматора на $15_8 = 13_{10}$ разрядов вправо, а команда 036 00072 — сдвиг на 6 разрядов влево ($100_8 - 72_8 = 6$). Аналогично выполняется и команда сдвига по порядку числа (код 026): адресная часть команды здесь содержит адрес ячейки, порядковая часть содержимого которой исполняет роль указателя направления и величины сдвига так же, как и адрес в команде 036.

Такой же принцип работы команд сдвига применяется в двухадресных и трехадресных машинах: один из адресов используется для указания величины и направления сдвига тем или иным способом, а другой — для указания адреса сдвигаемого слова. В трехадресной машине третий адрес есть адрес результата. В двухадресной — результат остается в сумматоре или записывается по второму адресу.

§ 2. СИСТЕМА КОМАНД ВЫЧИСЛИТЕЛЬНОЙ МАШИНЫ. ОПЕРАЦИИ УПРАВЛЕНИЯ И ОБМЕНА

К операциям управления в первую очередь относится команда *стоп*, останавливающая вычисления и прекращающая работу машины. На трехадресных машинах типа М-20 она имеет код 77, на машине БЭСМ-6 — код 33. Если в этой команде указать адреса ячеек памяти, то их содержимое в момент остановки машины вызывается в регистры пульта управления.

Как уже говорилось (см. § 2 гл. III), выражение «управление находится в ячейке *Я*» означает, что адрес ячейки *Я* записан в *счетчике команд*. По адресу в счетчике команд устройство управления вызывает содержимое данной ячейки, которое и расшифровывается как очередная команда, подлежащая выполнению. Все плавающие, фиксированные и логические операции, которые рассматривались в предыдущем параграфе, являются *операциями с естественным управлением*. Это означает, что при выполнении такой команды к счетчику команд автоматически прибавляется единица. Тем самым управление передается следующей ячейке. Если требуется изменить порядок выполнения команд, т. е. передать управление какой-либо другой ячейке, то для этого необходимо изменить содержимое счетчика команд. Именно это и делают операции управления (более точно следует говорить операции *передачи управления*).

Передача управления делится на безусловную и условную. Безусловная передача происходит всегда, условная — при выполнении некоторых условий. Начнем с рассмотрения безусловной передачи управления на одноадресной машине. На машине БЭСМ-6 эта команда имеет код 30. Выполнение команды 30 36457 состоит в том, что содержимое счетчика команд стирается и туда записывается адрес 36457. В результате этого следующая команда будет вызвана из ячейки 36457, т. е. туда п е р е д а н о у п р а в л е н и е. Напомним, что в каждой ячейке машины БЭСМ-6 записываются две команды, которые вызываются одновременно. Таким образом, управление будет передаваться л е в о й из них.

Безусловная передача управления на двухадресных машинах использует только один адрес из имеющихся двух в команде. Например, на машине Минск-22 команда с кодом — 30 передает управление по первому (левому) адресу. Это означает, что в результате выполнения команды

—30 1243 5472

в счетчик команд будет занесено число 1243. Второй адрес служит для записи результата предыдущей операции, хранившегося до сих пор в сумматоре. В приведенной команде этот результат запишется в ячейку 5472.

Аналогично устроена команда безусловной передачи управления на трехадресных машинах, но здесь благодаря большому числу адресов появляются еще некоторые дополнительные возможности. Для задания передачи управления достаточно, как мы уже видели, одного адреса. Если считать, что передача управления происходит, скажем, по второму адресу, как это сделано в машинах типа М-20, то команду безусловной передачи управления ячейке 1243 можно записать так:

56 0000 1243 0000.

В результате выполнения этой команды сотрется содержимое счетчика команд и туда запишется средний адрес команды, т. е. в данном случае 1243. Но в команде осталось д в а свободных адреса, которые можно использовать. В рассматриваемых машинах они используются для того, чтобы выполнить пересылку машинного слова из одной ячейки памяти в другую, если это потребуется. Команда 56 безусловной передачи управления может записываться в приведенном выше виде и будет выполняться так, как сказано. Но, кроме этого, ее можно записать, например, в виде

56 3412 1243 2176

и тогда при ее выполнении произойдет не только передача управления ячейке 1243, но также и пересылка содержимого ячейки 3412 (левый адрес) в ячейку 2176 (правый адрес).

Кроме описанной команды безусловной передачи управления, на всех машинах имеется еще одна аналогичная команда, которую называют командой *передачи управления с возвратом*. Она бывает совершенно необходима при обращении к подпрограммам, о чем мы подробнее расскажем в § 6 настоящей главы. Проще всего эта команда устроена в трехадресной машине. Именно в машинах типа М-20 команда

16 3412 1243 2176

выполняется так. В счетчик команд записывается адрес 1243, т. е. происходит безусловная передача управления по второму адресу команды. Одновременно с этим в ячейку 2176 (правый адрес) записывается машинное слово, которое имеет вид

2176: 16 0000 3412 0000,

т. е. команда безусловной передачи управления ячейке 3412 (левый адрес первоначальной команды!). Таким образом, команды с кодами 16 и 56 различаются использованием крайних адресов: в команде 56 в ячейку по правому адресу записывается с о д е р ж и м о е ячейки по левому, а в команде 16 — команда передачи управления

по левому адресу (команда *возврата*). Последняя команда уже является трехадресной, вся информация в ней существенна.

Выполнить все требуемые в команде передачи управления с возвратом действия на двухадресной или тем более одноадресной машине нельзя, так как вся нужная информация не поместится в менее чем трехадресной команде. Поэтому следует уменьшить объем информации. Из практики программирования известно, что чаще всего возврат после передачи управления подпрограмме происходит на команду, следующую за командой обращения. Так как адрес команды обращения в момент передачи управления известен, то его нет нужды указывать. Тем самым достигается требуемое уменьшение информации.

У двухадресных машин команда передачи управления с возвратом устроена так. На машине Минск-22 это команда с кодом—31. Если написать

$$Я\colon\ -31\ 1243\ 2401,$$

где $Я$ — адрес ячейки, в которой эта команда записана, то в счетчик команд запишется адрес 1243, а в ячейку 2401 — команда

$$2401\colon\ -30\ \ Я+1\ \ 0000,$$

т. е. команда безусловной передачи управления ячейке $Я+1$, следующей за командой передачи управления с возвратом.

Для одноадресной команды дело обстоит еще сложнее, так как там негде запоминать адрес возврата, хотя бы его и не требовалось указывать. Поэтому здесь нельзя обойтись без индекс-регистров, о которых мы до сих пор не упоминали. В § 7 гл. III было сказано, что во всех машинах второго поколения имеется один или несколько индекс-регистров и номер индекс-регистра, с которым работает данная команда, в ней указывается (кроме трехадресных машин, где такой регистр один). В чем состоит работа команды с индекс-регистром, мы будем говорить в § 5, а пока молчаливо предполагали, что индекс-регистр в командах не используется.

Перейдем теперь к рассмотрению *условных передач управления*. Команды условной передачи управления используют *управляющий сигнал*, который вырабатывается в машине одновременно с результатом выполнения арифметической или логической операции. Управляющим сигналом называют содержимое специального одноразрядного регистра. Таких регистров вообще может быть несколько, однако при выполнении команды условной передачи управления всегда учитывается значение только одного из них, в зависимости от характера предыдущей операции. Поэтому мы можем представлять себе, что всегда имеем дело с одним управляющим сигналом.

Трехадресные машины типа М-20 имеют один регистр управляющего сигнала, который обозначают буквой ω. Так как этот регистр одноразрядный, то возможны значения управляющего сигнала $\omega = 0$ и $\omega = 1$. Машины имеют две команды условной передачи управления, с кодами 36 и 76, которые передают управление в зави-

симости от значения управляющего сигнала ω, выработанного операцией, предшествующей команде условной передачи. Как обычно, адрес передачи управления указывается в этих машинах во втором (среднем) адресе команды.

Выполнение команды

76 0000 3564 0000

состоит в следующем: если управляющий сигнал ω к моменту исполнения данной команды имеет значение ω=0, то счетчик команд стирается и туда записывается 3564, т.е. управление передается по второму адресу команды. Если же ω=1, то эта команда пропускается, а к счетчику команд прибавляется 1, как и при выполнении любой команды с естественным управлением. Оставшиеся свободными два адреса в команде можно использовать, так же как и в команде с кодом 56, для пересылки машинного слова из одной ячейки в другую, если это требуется, причем пересылка происходит безусловно. Команда

76 3412 3564 7707

пересылает содержимое ячейки 3412 в ячейку 7707 и осуществляет условную передачу управления ячейке 3564 в зависимости от значения управляющего сигнала ω, как о том сказано выше. Аналогично работает и команда 36, передавая управление при другом значении управляющего сигнала. Именно в результате выполнения команды

36 0007 7710 1024

содержимое ячейки 0007 перешлется в ячейку 1024. Кроме того, если к моменту исполнения данной команды управляющий сигнал ω=0, то к счетчику команд прибавляется 1, а если ω=1, то счетчик команд стирается и туда записывается адрес 7710. Иначе говоря, при ω=0 управление передается на следующую ячейку, а при ω=1— по среднему адресу.

Остается познакомиться с правилами выработки управляющего сигнала. Арифметические (плавающие) операции делятся на две группы: группа сложения, куда относятся сложение, вычитание и вычитание абсолютных величин, и группа умножения, содержащая умножение, деление и извлечение квадратного корня. Для первой группы управляющий сигнал ω определяется знаком числа, а для второй — знаком порядка. Именно для первой группы ω=1, если результат отрицателен, и ω=0—в противном случае; для второй группы ω=1, если порядок результата положителен, т. е. если результат по абсолютной величине не меньше 1; в противном случае ω=0. Для фиксированных операций — сложения и вычитания адресных или операционных частей — ω=1 при наличии переполнения («теряющаяся» единица посылается в регистр ω), в противном случае ω=0. Наконец, для группы логических операций

$\omega=1$ тогда и только тогда, когда результатом действия является нулевое машинное слово.

Двухадресные вычислительные машины имеют три различных управляющих сигнала, которые определяются по описанным только что правилам. Именно сигнал ω вырабатывается для операций группы сложения, сигнал γ — для операций группы умножения и сигнал ν — для логических операций. Их значения определяются по тем же правилам, что и для трехадресных машин, кроме сигнала γ. Дело в том, что здесь записывается истинный, а не условный порядок числа со своим знаком, поэтому $\gamma=1$ при отрицательном порядке и $\gamma=0$, если порядок числа неотрицателен. Иногда используется также *сигнал переполнения* φ, который вырабатывается при всех арифметических операциях; $\varphi=1$, если произошло переполнение.

Здесь имеются команды для передачи управления по каждому из управляющих сигналов в отдельности. На машине Минск-22 для каждого управляющего сигнала достаточно одной команды: в ней указываются два адреса передачи управления в зависимости от выполнения условий (в трехадресной машине одним из таких адресов в с е г д а является адрес $Я+1$). Например, команда

—32 2564 7431

передает управление ячейке 2564 при $\omega=0$ и ячейке 7431 при $\omega=1$. Команда с кодом — 34 передает управление по левому адресу при $\nu=0$ и по правому при $\nu=1$. Сигнал γ на этой машине не используется, но есть команда передачи управления по сигналу φ, с кодом — 33, которая передает управление по левому адресу при $\varphi=0$ и по правому при $\varphi=1$.

На одноадресной машине БЭСМ-6 могут вырабатываться те же три управляющих сигнала ω, γ, ν (сигнал γ вырабатывается так же, как на трехадресных машинах типа М-20), но имеется только две команды условной передачи управления, проверяющие один из этих сигналов, в зависимости от группы, к которой принадлежит предшествующая этой команде операция. Так, команда 26 54314 передает управление ячейке 54314 в том случае, если управляющий сигнал $\Delta=0$, причем Δ равно ω, γ или ν в зависимости от того, к какой группе — сложения, умножения или логической — принадлежит предыдущая команда. Аналогично команда с кодом 27 передает управление при $\Delta=1$.

Следующая большая группа команд относится к работе с индексными регистрами. Мы не будем останавливаться на этих командах, так как вообще оставляем работу с индекс-регистрами в стороне, ибо она требует более детального знакомства с программированием, чем это будет сделано в нашей книге. Укажем только, что эти команды дают возможность записать в любой индекс-регистр заданное машинное слово или содержимое любой ячейки памяти (или части ячейки) или требуемым образом изменить содержимое индекс-регистра. Имеются также команды, осуществляющие условную передачу управления в зависимости от содержимого индекс-регистра, которые мы не рассматривали в предыдущей группе. Они осуществляют одновременно проверку содержимого индекс-регистра и

его изменение. Обычно такие команды называют *командами окончания цикла*. Несмотря на их важность для программирования, мы не можем уделить им достаточное внимание, так как с самим понятием цикла мы познакомим читателя лишь в § 5 настоящей главы.

Наконец, последняя группа команд, на которой мы остановимся, относится к обмену информацией. Это так называемые *операции обмена*, куда относятся как операции обмена внутри машины, между ее различными регистрами или устройствами, доступные для программиста, т. е. осуществляемые п р о г р а м м н ы м путем, так и операции обмена между оперативной памятью машины и различными периферийными устройствами, в частности внешней памятью.

Сюда относятся, прежде всего, операции переноса машинного слова из памяти в сумматор и из сумматора в память, особенно существенные для одноадресных машин. Так, на одноадресной машине БЭСМ-6 команда 000 1254 переносит содержимое сумматора в ячейку памяти 01254, а команда 010 3040, напротив, переносит содержимое ячейки 03040 в сумматор.

Двухадресные машины также имеют аналогичные команды переписи из одной ячейки в другую и в сумматор или из сумматора. Так, на машине Минск-22 команда

— 10 2534 1114

передает машинное слово из ячейки 2534 в ячейку 1114 и одновременно в сумматор. Кроме того, команда с кодом — 13 переписывает в ячейку по правому адресу и в сумматор содержимое пультового регистра, номер которого указан в первом адресе.

На трехадресных машинах типа М-20 также имеются аналогичные команды: команда 00 переносит содержимое ячейки так же, как и в командах передачи управления (из левого адреса в правый), а команда 20 переносит содержимое пультового регистра, номер которого указан в левом адресе, в ячейку, указанную в правом адресе. Однако не эти команды определяют работу рассматриваемой группы, а команды обмена информацией между оперативной памятью и внешними устройствами, куда входят также ввод и вывод. Этими последними мы пока заниматься не будем, оставляя их до следующего параграфа, а скажем несколько слов об операциях обмена. Заметим, что при обмене между оперативной и внешней памятью перенос из оперативной памяти во внешнюю называется *записью*, а перенос из внешней памяти в оперативную — *чтением*.

Трехадресные машины типа М-20 имеют по существу одну команду, предназначенную для целей обмена. Однако, ввиду того что для такой команды требуется много различной информации, под нее отводится две ячейки памяти. Тем самым мы получаем для записи требуемой информации шесть адресов. Первая из этих команд имеет код 50, вторая код 70, которая всегда может идти только после команды с кодом 50 (первая команда в нескольких случаях может выполняться самостоятельно, но обычно исполняется вместе со следующей с кодом 70). Левый адрес команды 50 занят условным

числом, с помощью которого определяется, какую именно операцию следует выполнять: работу с магнитным барабаном или с магнитной лентой, чтение или запись, разметку ленты и т. п.

Обмен между оперативной памятью и внешними запоминающими устройствами идет не отдельными машинными словами, а массивами, границы которых следует указывать. Его можно контролировать при помощи *контрольного суммирования*: все машинные слова, входящие в обмениваемый массив, суммируются при помощи операции циклического сложения. Полученная *контрольная сумма* записывается на барабан или на ленту после перенесенного туда массива, а при обратном считывании может быть проверена. Подобный контроль производится автоматически и может быть заблокирован занесением единицы в один из разрядов условного числа.

Второй адрес команды 50 указывает начало массива или зоны, отведенных для обмена на внешнем запоминающем устройстве. Границы обмениваемого массива в оперативной памяти задаются началом и концом. Адрес последней ячейки обмениваемого массива в оперативной памяти записывается в третьем адресе команды 50, а адрес первой — в первом адресе команды 70, идущей непременно в следующей строке после команды 50. Второй и третий адреса команды 70 используются так: третий адрес указывает адрес ячейки оперативной памяти, предназначенной для записи туда контрольной суммы, накопленной при записи или при считывании; во втором адресе указывается ячейка, которой будет передано управление при несовпадении контрольных сумм, если контроль не блокирован. Например, команда

```
50  0015  4000  3500
70  3000  1060  3501
```

означает: записать на барабан № 1, в ячейки, начиная с 4000, массив, занимающий в оперативной памяти ячейки от 3000 до 3500. Контрольную сумму записать в ячейку 3501. Адрес 1060 в настоящей команде не используется.

Машины Минск-22 и БЭСМ-6 имеют несколько различных команд с модификациями для различных операций.

§ 3. ВВОД И ВЫВОД. ПРОСТЕЙШИЕ ПРОГРАММЫ

В предыдущих параграфах мы рассмотрели основные операции, входящие в системы команд большинства советских электронных вычислительных машин второго поколения. Этого достаточно, чтобы составить простейшие программы, написанные *на языке вычислительной машины*. Рассмотрим, в качестве примера, вычисление по простой алгебраической формуле

$$y=\frac{a+bx}{ax+b}$$

в предположении, что величины a, b, x находятся в ячейках памяти, например, соответственно с адресами 0700, 0701, 0770. В § 2 гл. III мы уже разбирали алгоритм этого вычисления, т. е. представление его в виде последовательности элементарных операций; остается записать каждую операцию на языке вычислительной машины в виде команды машины. Естественно, что для различных машин это придется делать по-разному.

Начнем с трехадресных машин типа М-20 (БЭСМ-4, М-220, М-222). При разборе арифметических операций на этих машинах мы указывали, что команды умножения и деления имеют коды соответственно 05 и 04. Добавим к этому, что сложение имеет код 01, результат вычисления y поместим в ячейку 0771, а для записи промежуточных результатов отведем *рабочие ячейки* начиная с 0601. Тогда программу можно будет записать так:

```
05 0701 0770 0601
01 0700 0601 0602
05 0700 0770 0603
01 0603 0701 0604
04 0602 0604 0771
77
```

Программы для двухадресных машин будут иметь приблизительно такую же длину и потребуют меньшего числа ячеек, так как благодаря различным модификациям арифметических операций в ряде случаев можно использовать сумматор арифметического устройства для запоминания промежуточного результата, используемого в следующей команде. Так, в кодах машины Минск-22, сохраняя те же адреса ячеек для записи величин a, b, x, y и промежуточных результатов, можно написать:

```
+35 0701 0770
+16 0700 0601
+35 0700 0770
+16 0701 0602
+44 0602 0601
—10 0601 0771
—00
```

Для пояснения работы программы заметим, что команда плавающего умножения +35 берет сомножители из указанных ячеек памяти и результат оставляет в сумматоре; команда +16 складывает содержимое сумматора с содержимым ячейки по левому адресу и записывает результат в ячейку по правому адресу. В команде деления делимое записывается во втором адресе, а делитель — в первом, причем команда +44 записывает результат по второму адресу команды, стирая промежуточный результат в ячейке 0601 (числитель). Наконец, команда —10, как мы указывали в § 2, переносит окончательный результат из ячейки 0601 в ячейку 0771, которую мы отвели для величины y. Команда —00 — команда «*стоп*».

Теперь напишем эту же программу в кодах одноадресной машины БЭСМ-6. Заметим только, что выгоднее вычислить сначала знаменатель, а потом числитель, так как при операции деления делимое предполагается находящимся в сумматоре. Команды мы располагаем по две в одной строке, так что строка соответствует ячейке памяти

```
010 0700 017 0770
004 0701 000 0601
010 0701 017 0770
004 0700 016 0601
000 0771 37
```

Здесь 04 — код сложения, а 016 — код деления. Об остальных операциях мы говорили в § 1,2.

Чтобы эта (или любая другая) программа работала, необходимо ввести ее в память вместе с числовыми значениями величин a, b, x, причем команды программы должны быть записаны в ячейки памяти подряд, внести в счетчик команд адрес первой из командных ячеек и пустить машину в режиме автоматического выполнения команд. Для этого программу и числовые данные нужно поместить на *внешнем носителе*, что делается при помощи различных *устройств подготовки данных*, и при помощи *устройства ввода* (или *читающего устройства*) ввести в машину.

Как мы уже говорили, внешними носителями информации являются *перфокарты* или *перфоленты*, а устройствами подготовки данных — *перфораторы* (иногда говорят — *входные перфораторы*), пробивающие на картах или лентах нужные отверстия. Информация всегда записывается в двоичной системе, и пробитое в каком-либо месте отверстие означает 1, а отсутствие отверстия — 0.

Рассмотрим некоторые возможные способы представления информации на перфокартах и перфолентах. Начнем с перфокарт, о которых несколько слов уже было сказано в предыдущей главе.

При вводе перфокарт для трехадресных машин типа М-20 возможны два способа размещения информации: *построчное* и *поколонное*. При построчном размещении информации (такие карты часто называют *двоичными*) читающее устройство машины воспринимает пробивки не во всех колонках, а только в 47 из них, которые показаны в верхней (двенадцатой) строке перфокарты на рисунке 80. Колонки 18 и 80 играют при этом особую роль и называются *маркерными*, а пробивки в них — *маркерами*: пробивка в 80-й колонке называется *адресным маркером*, а пробивка в 18-й колонке — *исполнительным маркером*. Остальные 45 разрядов называются основными и соответствуют 45 разрядам ячейки памяти. Эти разряды делятся на части, аналогичные частям машинного слова, операционную и адресную; например, колонки 26—31 соответствуют 42—37 разрядам ячейки памяти, содержащим код операции. Адресная часть в свою очередь делится на 3 адреса по 12 разрядов — колонки 34—47, 49—62, 64—77 (с пропусками).

Если в строке пробит адресный маркер (пробивка в 80-й колонке), то часть слова, относящаяся к первому адресу, воспринимается читающим устройством как адрес ячейки, а остальные части пробивок в этой строке (если они есть) игнорируются. Впрочем, обычно их и не бывает. В нулевой строке перфокарты на рисунке 80 набит адрес 1523 с адресным маркером. Если в строке пробит исполнительный маркер (18-я колонка), то основные разряды этой строки воспринимаются, как 45-разрядное машинное слово, которое следует ввести в ячейку с указанным адресом. После адресного слова может быть набито несколько слов с исполнительными маркерами. Они будут вводиться в ячейки памяти подряд, начиная с указанного адреса до тех пор (включая и следующие перфокарты), пока не встретится новый адресный код. Пробивки в обеих маркерных колонках одновременно есть признак контрольной суммы; встречая ее, устройство ввода прекращает свою работу. На рисунке 80 в строках 2—7 набита рассмотренная нами выше программа, в строке 9 — контрольная сумма.

Ввод числового материала осуществляется аналогично. Числа в двоичной системе набиваются так же, как и команды. Десятичные числа представляются в двоично-десятичной системе: каждая десятичная цифра записывается в двоичной системе в виде *тетрады* (четверки) двоичных разрядов. Такое преобразование совершается автоматически перфоратором.

Пуск читающего устройства ввода можно осуществить одним из двух различных способов: кнопкой «ввод» на пульте управления машины и программно, при помощи команд. Трехадресные машины типа М-20 имеют две команды ввода с кодами 10 и 30. В первом адресе этой команды указывается адрес ячейки, начиная с которой следует вводить прочитанную информацию, если только на самой карте нет адресного кода; при его наличии адресный код, пробитый на карте, имеет приоритет, адрес в команде ввода будет игнорироваться. Встретив контрольную сумму, машина останавливает читаю-

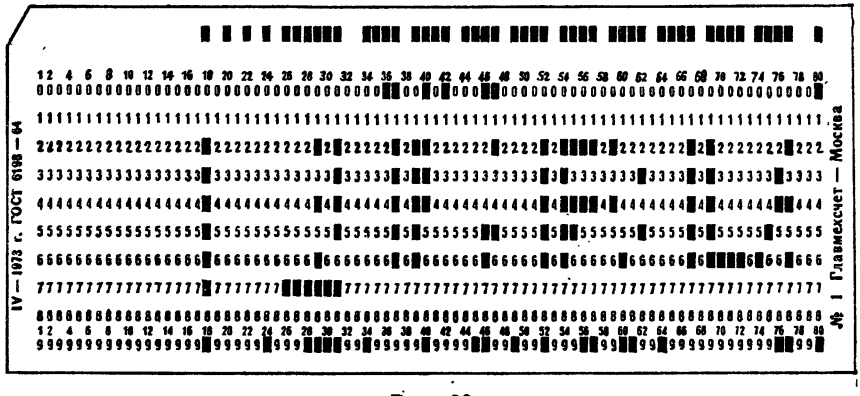


Рис. 80

щее устройство и накопившуюся контрольную сумму вводимого материала, которая подсчитывается в процессе ввода, записывает в ячейку, указанную в третьем адресе команды ввода. Одновременно эта сумма сравнивается с той, которая набита на перфокарте. Дальше наступают различия в работе двух команд ввода. Команда с кодом 30 поступает так: при совпадении контрольных сумм управление передается на следующую ячейку, при несовпадении — по второму адресу команды ввода. Команда с кодом 10 при совпадении контрольных сумм передает управление на следующую ячейку, как и предыдущая, при несовпадении останавливается. Если снова нажать пуск, то управление передается по второму адресу команды.

Мы говорили пока о д в о и ч н ы х картах для построчного ввода цифровой информации. Так же в случае надобности происходит ввод алфавитно-цифровой информации. Различие состоит лишь во входном перфораторе, который в этом случае переводит пробиваемый на них символ не в триаду, как восьмеричную цифру при набивке команд или тетраду, как десятичную цифру при работе в десятичной системе, а в комбинацию из с е м и двоичных разрядов. При такой набивке на карте не пробиваются колонки 20, 22, 24, соответствующие разрядам 45—43 ячейки, а в оставшиеся 42 разряда помещаются 6 символов.

Наряду с описанным применяется и еще один способ ввода в трехадресную машину алфавитно-цифровой информации: при помощи так называемых *десятичных карт*. Это те же самые перфокарты, только они вводятся в читающее устройство у з к о й стороной (боком) и в них информация размещается поколонно. Для изображения каждого символа отводится к о л о н к а. Цифра изображается одной пробивкой в соответствующей строке: знаки + и — также одной пробивкой соответственно в 12 или 11 строках. Буквы и прочие знаки изображаются уже двумя пробивками в колонке в различных комбинациях, которых мы приводить не будем. На рисунке 81 в первых девяти колонках показано набитое число —3874,119.

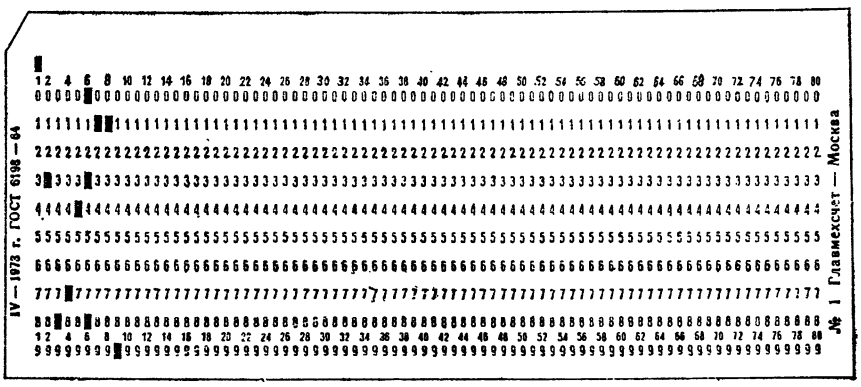


Рис. 81

Информация на перфокартах, предназначенных для двухадресной машины Минск-22, наносится только поколонно, но двумя различными способами: с помощью *кода перфокарт* и двоичного кода. Код перфокарт вполне аналогичен описанному выше для трехадресных машин: для изображения символа отводится колонка. Цифра изображается одной пробивкой в соответствующей строке; буквы и прочие знаки — различными комбинациями пробивок (отличными от используемых в трехадресных машинах). *Двоичный код* изображает алфавитно-цифровые символы шестеркой двоичных цифр. Каждая колонка имеет 12 строк, которые делятся на две группы по 6; в них-то и помещаются соответствующие шестерки двоичных цифр—пробивка означает 1, а ее отсутствие 0.

Соответственно этим двум способам возможны два режима ввода с перфокарт: *ввод форматных карт*, если информация записана в коде перфокарт, и *ввод копии карт*, если информация записана в двоичном коде.

Команда ввода на машине Минск-22 имеет код — 54. Во втором адресе команды указывается адрес ячейки оперативной памяти, начиная с которой следует вводить информацию. Первый адрес содержит условное число, 13-й разряд которого указывает режим ввода (0 — ввод по форматной карте, 1 — ввод копий карт), а в разрядах 16—24 записано число, соответствующее количеству вводимых карт.

Для двухадресных машин (особенно для Минск-22) в качестве первичного внешнего носителя информации более распространена *перфолента*. На перфоленте информация может быть набита как *в коде машины*, так и в *международном телеграфном коде* МК-2 и соответственно возможны различные режимы ввода.

Адрес ячейки, с которой надо начинать ввод информации, может быть набит впереди с признаком адреса, таким же, как и при вводе в коде машины, либо указан в левом адресе команды ввода. Адресный код, указанный на внешнем носителе, обычно имеет приоритет перед кодом, указанным в команде.

Перейдем теперь к вопросу о выводе информации из машины. Как мы уже говорили в предыдущей главе, используется вывод на перфорацию (на перфокарты или перфоленту), на быстропечатающее цифровое устройство («узкая» печать) и алфавитно-цифровое печатающее устройство («широкая» печать). Почти во всех случаях вывод требует большой дополнительной работы, связанной с переводом чисел из двоичной системы в десятичную, размещения их в нужных местах и в нужном порядке и т. п. Вся эта деятельность обычно возлагается на стандартные библиотечные подпрограммы, о чем будет идти речь в конце настоящей главы. Здесь мы остановимся только на командах вывода, являющихся завершающими командами таких программ.

Трехадресные машины не имеют отдельных команд вывода, а используют специальные условные числа в командах обмена 50—70, о которых шла речь в предыдущем параграфе. Например,

обращение к десятичной печати требует условного числа 0100 (без контроля — 2100), к перфорации — 0200, к алфавитно-цифровой печати — 0140.

У двухадресной машины Минск-22 имеются специальные команды вывода. Например, машина Минск-22 имеет команды вывода с кодами — 60, —61, —62, —63. Из них команда — 63 применяется при выводе на перфокарты, команда —61— при выводе на телетайп или перфоратор № 2, команда — 60 — при выводе на узкую печать или перфоратор № 1 и, наконец, команда — 62 — при выводе на алфавитно-цифровую печать. В трех последних случаях первый адрес занят различными условными числами, а второй указывает адрес печатаемой ячейки.

Совсем иначе обстоит дело с одноадресной машиной БЭСМ-6. Здесь и для ввода, и для вывода применяются экстракоды, представляющие собой лишь подобие команд, а на самом деле означающие обращение к специальным подпрограммам. Эти подпрограммы тесным образом связаны с операционными системами, в которые они входят, и определяются не устройством машины, а математическим обеспечением; поэтому здесь мы не станем рассматривать эти вопросы. Кое-что будет сказано позже, в гл. VIII.

§ 4. ЯЗЫК СОДЕРЖАТЕЛЬНЫХ ОБОЗНАЧЕНИЙ

Программы на языке машины, которые мы рассматривали в предыдущем параграфе, очень трудно писать и еще труднее читать. Программа будет выглядеть много проще и привычнее, если обозначить адреса ячеек памяти обычными буквами, теми же, которые употребляются в программируемой формуле, а вместо кода операции писать знак соответствующего действия да еще в нужном месте. Про такую программу говорят, что она написана в «содержательных обозначениях».

Напишем, например, программу для вычисления величины

$$y=\frac{a+bx}{ax+b}.$$

В содержательных обозначениях запись этой программы будет почти точно совпадать с привычной математической записью алгоритма вычислений по заданной формуле:

1) $b\times x = R_1$	4) $R_3+b=R_4$
2) $a+R_1=R_2$	5) $R_2:R_4=y$
3) $a\times x = R_3$	6) *стоп*

Буквы a, b, x, y означают здесь адреса ячеек, в которых помещены соответствующие величины, R_1—R_4 означают адреса *рабочих*

ячеек, предназначенных для записи промежуточных результатов, словом «*стоп*» заменена команда остановки машины. Каждая строка этой записи представляет собой условную запись команды; так как в каждой из них фигурируют три компоненты, то ясно, что речь идет о команде трехадресной машины.

Легко превратить каждую такую строку в команду, написанную в кодах машины. Для этого достаточно знать коды операций и фактические адреса ячеек, отведенных для записи упоминаемых величин. Например, если число b помещено в ячейку памяти с адресом 0701, число x — в ячейку 0770, а для промежуточного результата R_1 отведена ячейка 0601, то команда 1) соответствует машинному слову вида

$$05 \quad 0701 \quad 0770 \quad 0601$$

Такой переход от содержательных обозначений к языку машины носит название *кодирования* (иногда *кодировки*). Чтобы облегчить кодирование и превратить его в чисто механический процесс, выполняемый по простым формальным правилам, полезно ввести некоторые ограничения в содержательную запись команд.

С точки зрения математической записи алгоритма нет никакой разницы между записями $a+b=c$ и $c=a+b$. При содержательной записи команды они не эквивалентны, и мы будем всегда пользоваться только одной из них в зависимости от особенностей расположения адресов в команде машины: мы хотим, чтобы последовательность обозначений в содержательной записи команды точно соответствовала последовательности адресов в машинной записи. Так как в арифметических командах адрес результата является третьим, то будем считать запись $a+b=c$ верной, запись же $c=a+b$ недопустимой, а потому неправильной.

Иногда смысл содержательной записи команды может отличаться от смысла соответствующего математического равенства. Например, в приведенной нами программе произведение $b \times x$, вычисленное командой 1) и записанное ею в ячейку R_1, требуется только как слагаемое во второй команде и после использования может быть стерто. Поэтому числитель дроби, который вычисляется командой 2), можно записать в ту же ячейку R_1, не занимая новой. Это означает, что команду 2) можно записать в виде

$$a+R_1=R_1,$$

т. е. сложить числа из ячеек с адресами a и R_1 и сумму записать в ячейку R_1, что отнюдь не означает «равенства» $a=0$. Здесь R_1 слева и справа от знака равенства означает о д и н и т о т ж е а д р е с, но р а з н ы е ч и с л а, записанные в этой ячейке до и после выполнения операции.

Для команд, соответствующих арифметическим операциям, содержательные обозначения напрашиваются сами собой, и мы огра-

ничимся тем, что приведем их список.

$$a + b = c$$
$$a - b = c$$
$$|a| - |b| = c$$
$$a : b = c$$
$$a \times b = c$$
$$\sqrt{a} = c$$

Однако при написании программы приходится пользоваться и многими другими командами, поэтому в языке содержательных обозначений необходимо иметь условные обозначения для всех используемых команд машины. Из них столь же естественные и бесспорные, как для арифметических, можно предложить лишь для логических действий, поскольку для них имеются общепринятые символы:

$a \vee b = c$ (логическое сложение),
$a \wedge b = c$ (логическое умножение, или пересечение),
$a \nsim b = c$ (сверка, отрицание равнозначности).

Остальные обозначения могут уже оказаться менее удачными. Ниже приведем достаточно широко используемые обозначения для фиксированных операций и команд передачи управления, работа которых была описана в § 1—2. Для сложения и вычитания адресных частей будем пользоваться обозначениями

$$a+,\ b=c,$$
$$a-,\ b=c.$$

Наличие запятой вместе со знаком арифметического действия подчеркивает, что речь идет об операции с фиксированной запятой. Аналогичные обозначения используются для сложения и вычитания операционных частей, только запятая ставится перед знаком действия:

$$a,+\ b=c,$$
$$a,-\ b=c.$$

Команды передачи управления, как безусловные, так и условные, совмещены с пересылками. В зависимости от того, в каких местах команды записываются те или иные адреса, команду передачи управления нужно будет обозначать по-разному. Предположим, например, что адрес команды, которой следует передать управление, т. е. номер ячейки, который будет записан после выполнения данной команды в счетчике команд устройства управления, указывается на месте третьего адреса команды, а пересылка происходит из ячейки, указанной в первом адресе, в ячейку, указанную во втором адресе. Тогда команду можно записать так:

$$K\ \ a \rightarrow b,\ c,$$

где на месте буквы K следует поставить символ, показывающий характер передачи управления, скажем $Б$, если передача безусловная, и $У0$ или $У1$ в случае условной передачи управления при заданном значении управляющего сигнала. Вместо $У0$ и $У1$ нередко пишут $У_0$ и $У_1$ или даже просто (0) и (1), а символ $Б$ нередко опускают.

Адреса в команде передачи управления могут располагаться поиному. Например, в машинах БЭСМ-4, М-220 и М-222 ячейка, которой передается управление, всегда указывается в среднем адресе команды (передача управления по второму адресу), а ячейка, в которую происходит запись,— в правом (третьем) адресе. Поэтому для этих машин команду безусловной передачи управления следует записать так:

$$Б \quad \overline{a \quad b} \quad \overrightarrow{c}.$$

В результате выполнения этой команды машинное слово из ячейки с адресом a будет переписано в ячейку с адресом c, а в счетчик команд будет записан адрес b.

Эти примеры показывают, что содержательные обозначения самым непосредственным образом связаны с конкретной машиной. Язык содержательных обозначений относится, таким образом, к *машинно-ориентированным языкам* программирования: при переходе к новой машине его «слова» необходимо менять. Тем более это необходимо при переходе к машинам другой адресности — двухадресным или одноадресным.

Рассмотрим, как выглядят команды в одноадресных или двухадресных машинах. Как уже говорилось, из трех компонент арифметической операции здесь одна или даже две компоненты находятся в сумматоре. Поэтому для одноадресной машины операция сложения будет выглядеть так:

$$S+a=S,$$

где S означает сумматор машины. Опуская всегда подразумевающиеся обозначения S слева и справа, мы получаем обозначение для операции сложения в одноадресной машине

$$+a.$$

Кроме уже перечисленных операций, обозначения для которых легко получить аналогично описанной, для одноадресной машины большое значение имеют операции посылки из ячейки в сумматор и обратно, записи из сумматора в ячейку памяти. Эти операции мы будем обозначать

$$\Leftarrow a$$
$$\Rightarrow a$$

и называть соответственно *чтением* (из ячейки) и *записью* (в ячейку).

Пользуясь описанными обозначениями, мы можем теперь написать программу вычисления той же величины $y=(a+bx)/(ax+b)$

в содержательных обозначениях для одноадресной машины следующим образом:

1) $\Leftarrow a$	6) $\times x$
2) $\times x$	7) $+a$
3) $+b$	8) $:R_1$
4) $\Rightarrow R_1$	9) $\Rightarrow y$
5) $\Leftarrow b$	10) *стоп*

Несколько иначе обстоит дело для двухадресных машин. Здесь одна из компонент операции, причем различная в зависимости от модификации, находится в сумматоре. Поэтому наряду с естественной операцией $a+b=S$ здесь возможна и операция $S+a=b$, и, кроме того, возможна и такая операция сложения, при которой сумма записывается в память по второму адресу. Ее обозначают $a+b=b$. Так как в последнем обозначении знак S, означающий сумматор, отсутствует, то это обозначение нельзя превратить в двухкомпонентное, опустив S, что можно сделать с предыдущими модификациями. Поэтому удобнее для двухадресной машины сохранить трехкомпонентную запись, имея в виду, что либо одна из компонент есть сумматор (S), либо две последние совпадают (запись по второму адресу).

Если воспользоваться этой системой обозначений, то программу вычисления по уже дважды рассмотренной формуле можно записать так:

1) $a\times x=S$	4) $S+a=R_2$
2) $S+b=R_1$	5) $S:R_1=y$
3) $b\times x=S$	6) *стоп*

В качестве пояснения напомним только, что результат любой операции, независимо от того, записывается он в ячейку или нет, всегда остается и в сумматоре, чем мы и воспользовались в команде 5). Этим объясняется вычисление знаменателя, прежде чем вычислен числитель.

§ 5. ПРОГРАММИРОВАНИЕ В СОДЕРЖАТЕЛЬНЫХ ОБОЗНАЧЕНИЯХ

Пользуясь языком содержательных обозначений познакомимся с основами программирования. При этом ограничимся рассмотрением только трехадресных машин, так как именно для них язык содержательных обозначений ярче всего демонстрирует свои достоинства и преимущества и кроме того здесь наиболее четко выявляются основные особенности различных типов программ. Переход от трехадресных машин к одноадресным или двухадресным, как мы видели в предыдущем параграфе, особых затруднений не вызывает.

Наиболее простым и вместе с тем одним из основных типов программ является счет по формуле, или, как часто говорят, *расписка*

формулы по командам. Написание программы счета по формуле сводится к выписыванию последовательности арифметических операций; пример такой программы мы уже видели в предыдущем параграфе. По этому поводу следует сделать только два небольших замечания. Первое состоит в том, что последовательность расположения операций весьма существенна и может иногда сильно ускорить или, наоборот, замедлить вычисления. Поэтому часто бывает полезно предварительно подвергнуть заданную формулу каким-либо аналитическим преобразованиям. Второе замечание относится к тому, что в состав формулы могут входить те или иные элементарные функции (например, тригонометрические), о вычислении значений которых мы будем говорить несколько позже.

Рассмотрим еще один пример программы вычисления по формуле. Пусть требуется вычислить значение алгебраической дроби

$$z=\frac{5x^2y-3}{x^3-2xy+4y^2}.$$

При этом предполагается, что числа x, y и все требуемые числовые коэффициенты уже введены в память машины в соответствующие ячейки и нашей задачей является только получение в ячейке памяти значения z. Вопрос о том, как узнать полученное значение, пока нами не рассматривается.

Требуемые вычисления можно осуществить с помощью программы

1) $x \times x = R_1$
2) $R_1 \times y = R_2$
3) „5“ $\times R_2 = R_2$
4) $R_2 -$ „3“ $= R_2$
5) $R_1 \times x = R_1$
6) $y + y = R_3$
7) $R_3 - x = R_4$
8) $R_3 \times R_4 = R_3$
9) $R_1 + R_3 = R_1$
10) $R_2 : R_1 = z$
11) *стоп*

Для удобства вычислений мы записали знаменатель в виде $x^3+2y(2y-x)$. Это позволило уменьшить число операций и обойтись без ячеек, содержащих коэффициенты 2 и 4. Кроме того, мы использовали еще одно новое обозначение, впрочем, достаточно понятное: „5“ и „3“ означают *адреса ячеек*, в которых лежат соответственно числа 5 и 3. Такое обозначение очень удобно и будет часто использоваться в дальнейшем.

Вычисляемая функция может быть задана различными формулами на разных участках своей области определения. Тогда возникает необходимость пользоваться *разветвляющимися программами*, в которых, кроме арифметических операций, используются команды передачи управления.

Пусть, например, требуется вычислить функцию, определенную условиями:

$$y=\begin{cases} 1, & \text{если } x<0, \\ (x-1)^2, & \text{если } 0\leqslant x\leqslant 1, \\ x-1, & \text{если } x>1. \end{cases}$$

Вычисления можно выполнить с помощью такой программы.

1)	$x + 0 = x$	4)	$У0 \quad 6)$
2)	$У1 \quad$ „1“ $\quad 6) \quad \overrightarrow{y}$	5)	$y \times y = y$
3)	$x -$ „1“ $= y$	6)	*стоп*

Сделаем несколько пояснений относительно работы программы. Команда 1) нужна для проверки знака x. Если $x<0$, то вырабатывается управляющий сигнал $\omega=1$. В этом случае команда 2) перешлет в ячейку y число 1 и передаст управление команде 6), которая остановит машину. Если $x\geqslant 0$, то $\omega=0$ и команда 2) передаст управление на 3), хотя пересылка все равно произойдет и в ячейку y будет записана ненужная единица. Это не имеет значения, так как уже следующая команда 3) запишет в ячейку y величину $x-1$. Одновременно выработается и управляющий сигнал: $\omega=1$ при $x<1$ и $\omega=0$ при $x\geqslant 1$. Во втором случае в ячейке y уже находится нужное значение и команда 4) передаст управление на *стоп*. В первом же случае величину $x-1$ надо еще возвести в квадрат, что и будет сделано командой 5), которой будет передано управление при $\omega=1$.

Разветвляющиеся программы используются и в задачах невычислительного характера, например при упорядочении по величине. Пусть, например, в ячейках x_1, x_2 записаны два числа и их требуется перенести в ячейки y_1, y_2 так, чтобы в y_1 было записано большее из них, а в y_2 — меньшее, т. е.

$$y_1 = \max\{x_1,\ x_2\}, \quad y_2 = \min\{x_1,\ x_2\}.$$

Это можно сделать с помощью такой программы.

1)	$x_1 - x_2 = 0$	4)	$Б \quad \overrightarrow{x_1 \quad 6) \quad y_2}$
2)	$У0 \quad \overrightarrow{x_1 \quad 5) \quad y_1}$	5)	$x_2 = y_2$
3)	$x_2 = y_1$	6)	*стоп*

Как и в предыдущем случае, команда 1) служит для проверки условия $x_1\geqslant x_2$. Разность x_1-x_2 не нужна и ее можно никуда не записывать, поэтому в правом адресе можно поставить нуль. Работа программы будет вполне понятной, если построить ее *блок-схему* (рис. 82). Здесь прямоугольники означают части (блоки) программы, выполняющие рабочие операции (в данном случае — пересылку), а ромб — проверку условий. Из рисунка понятен также и смысл названия *разветвляющаяся программа*: программа состоит из

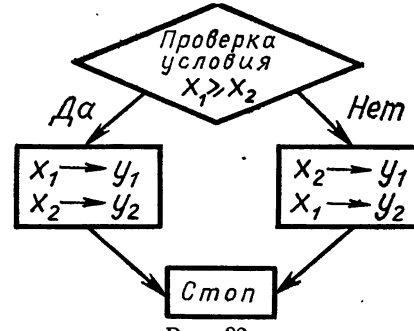


Рис. 82

двух в е т в е й; фактически исполняется та или другая из них, в зависимости от конкретных условий.

И программы счета по формуле, и разветвляющиеся программы характерны тем, что отдельные их команды выполняются только один раз (или не выполняются вообще, если в разветвляющейся программе процесс идет по другой ветви). Такие программы весьма невыгодны, поскольку время написания команды неизмеримо превышает время ее выполнения. Нужны программы, в которых одна и та же команда или группа команд выполняется многократно. Это достигается за счет введения *циклов* или *циклических программ*. Циклы являются составными частями всяких практически используемых программ, а потому организация цикла есть наиболее часто встречающаяся задача программирования.

Циклом называется группа команд, которая выполняется более чем один раз. Это определение было впервые дано Адой Лавлейс в 1843 году; оно сохраняет силу и до сегодняшнего дня. Рассмотрим простейшие примеры циклов.

Напишем программу для вычисления суммы

$$S = 1/1^2 + 1/2^2 + 1/3^2 + \ldots + 1/100^2.$$

Легко заметить, что все слагаемые вычисляются одинаково и зависят только от номера. Поэтому достаточно написать несколько команд, вычисляющих очередное слагаемое и прибавляющих его в S, а затем обращаться к этой группе, меняя номер слагаемого. Для этого можно воспользоваться такой программой.

1)	$0 = S$	6)	$S + R = S$
2)	Б $\overrightarrow{\text{„1“ 4) } n}$	7)	$n - \text{„100“} = 0$
3)	$n + \text{„1“} = n$	8)	У1 3)
4)	$n \times n = R$	9)	*стоп*
5)	$\text{„1“} : R = R$		

В приведенной программе команды 1)—2) выполняются только один раз и поэтому по точному смыслу определения к циклу не относятся. Их можно объединить в группу, которую мы будем называть *подготовкой цикла* *. Назначение этих команд — привести в первоначальное состояние рабочие ячейки программы S и n. Команда 2), кроме того, передает управление команде 4), обходя не требующееся пока изменение n.

Группа команд 4)—6) носит название *рабочей части* цикла. При заданном n эти команды вычисляют очередное (n-е) слагаемое и прибавляют его к ранее полученной сумме. Команда 3) предназначена для изменения n, т. е. для перехода к следующему слагаемому.

---

* По причинам, на которых мы не будем останавливаться, в учебниках программирования эту группу команд часто называют *восстановлением* или *возобновлением*.

Она будет выполняться, начиная со следующего шага. Соответствующая часть цикла (в более сложных случаях таких команд может быть несколько) называется *изменением*. Наконец, команды 7)—8) относятся к *проверке окончания*. До тех пор, пока $n<100$, при выполнении команды 7) вырабатывается управляющий сигнал $\omega=1$ и команда 8) передает управление команде 3), начиная вычисление следующего слагаемого. Как только станет $n=100$, разность в команде 7) станет равной нулю и выработается сигнал $\omega=0$, в результате чего команда 8) передаст управление на 9) и машина остановится.

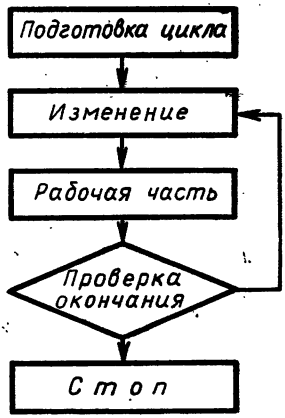


Рис. 83

Названные нами части приведенной программы выделяются в любом цикле. Обычно они и располагаются так же, как в нашем примере. Это означает, что обычный цикл строится, как правило, в соответствии с блок-схемой, приведенной на рисунке 83. Правда, в отдельных случаях бывают удобны и выгодны те или иные отступления от этой схемы. Иногда удается строить цикл так, чтобы некоторые команды выполняли сразу несколько функций, например изменения и проверки окончания одновременно. Но при любом расположении частей цикла названные элементы в нем непременно должны присутствовать.

Переменная n называется здесь *счетчиком цикла*, число 100, используемое для проверки окончания и означающее здесь количество слагаемых,— *эталоном цикла*. Поскольку текущее значение счетчика участвует в вычислениях, то счетчик называют *естественным*. В тех случаях, когда счетчик цикла требуется лишь для счета количества сделанных шагов и проверки окончания, а в вычислениях не участвует, его называют *искусственным счетчиком*.

Легко привести пример циклической программы с искусственным счетчиком. Пусть требуется вычислить значение $y = x^n$, где произвольное число n записано, как и x, в соответствующей ячейке памяти. Программу вычисления y можно написать так:

1) „1“ $=y$	5) $k-n=0$
2) $Б$ „1“ 4) $\overrightarrow{k}$	6) $У1$ 3)
3) $k+$„1“ $=k$	7) *стоп*
4) $y \times x = y$	

Счетчик цикла k здесь не участвует в вычислениях рабочей части и потому является искусственным. В обоих рассмотренных примерах циклов счетчики были плавающими числами, поэтому их называют *плавающими*. В ряде случаев, особенно для искусственных

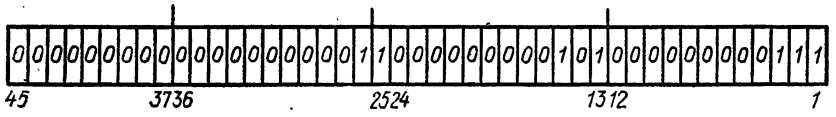


Рис. 84

счетчиков, удобно иметь их фиксированными, в виде числа единиц какого-либо адреса.

Будем обозначать ячейку памяти, содержащую одну единицу в 25, 13 или 1 разрядах, соответственно через (1, 0, 0), (0, 1, 0) или (0, 0, 1). Иначе говоря, будем представлять адресную часть ячейки состоящей из трех фиксированных двенадцатиразрядных двоичных целых чисел, содержащихся в разрядах 36—25, 24—13 и 12—1, которые и записываются в скобках через запятые. Символ (3,5,7) будет тогда обозначать ячейку, содержащую число 3 в левом адресе (единицы в 26 и 25 разрядах), число 5—во втором адресе (единицы в 15 и 13 разрядах) и число 7—в третьем адресе (единицы в 3, 2 и 1 разрядах). Предполагается, что в операционной части ячейки (разряды 45—37) записаны все нули (см. рис. 84).

Если считать, что показатель степени записан в фиксированной форме в виде содержимого ячейки (0, 0, n), то программу вычисления $y=x^n$ можно переписать с фиксированным искусственным счетчиком в виде

1)		„1“	$=y$	5)	$k-,(0,\ 0,\ n)=0$
2)	*Б*	(0, 0, 1)	4) $\longrightarrow k$	6)	*У*1 3)
3)		k	$+,(0,\ 0,\ 1)=k$	7)	*стоп*
4)		y	$\times\ x=y$		

Циклы, организованные по счетчику, подобно рассмотренным примерам, в которых проверка окончания производится путем сравнения счетчика с эталоном, носят название *арифметических циклов*. Другой тип циклов — *итерационные циклы* — отличаются от арифметических способом проверки окончания. В итерационных циклах прекращение вычислений происходит не после заданного числа повторений, а по достижении нужной точности.

Напишем программу для вычисления e^x с помощью ряда

$$e^x=1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots+\frac{x^n}{n!}+\ldots.$$

Обозначим общий член ряда через $u_n=x^n/n!$. Легко заметить, что

$$u_n=x\cdot x^{n-1}/(n\cdot(n-1)!)=u_{n-1}\cdot x/n.$$

Таким образом, следующее слагаемое получается из предыдущего умножением на x/n. Начальная часть соответствующей циклической программы пишется, как обычный арифметический цикл с естественным плавающим счетчиком:

1)		„1“	$=y$	5)	$u\times x=u$
2)		„1“	$=u$	6)	$u\ :\ n=u$
3)	*Б*	„1“ 5) $\longrightarrow n$		7)	$y+u=y$
4)		$n+$„1“	$=n$		

Дальше должна идти проверка окончания, которую можно организовать различными способами. Можно задаваться числом слагаемых N. Тогда мы получим обычный арифметический цикл (команды 8′)—10′)). Этот способ неудобен, так как полученная погрешность будет весьма различной для разных x. Другой способ — задать величину ε и продолжать вычисления до тех пор, пока очередное слагаемое не сделается меньше заданного ε — приводит нас к итерационному циклу (команды 8″—10″):

8′)	$n-N=0$	8″)	$\lvert u\rvert-\lvert\varepsilon\rvert=0$
9′)	*У1* 4)	9″)	*УО* 4)
10′)	*стоп*	10″)	*стоп*

Ошибочно полагать, что итерационный цикл с проверкой окончания 8″)—10″) обеспечит погрешность для e^x не свыше ε: из того, что все отброшенные члены меньше ε, никак не следует, что меньше ε будет их сумма. Если мы хотим получить e^x с точностью ε, то нужно подобрать такое ε_1, зависящее от ε (его нетрудно подсчитать с помощью методов теории рядов), чтобы выполнение неравенства $|u|<\varepsilon_1$ обеспечило бы, что сумма всех отброшенных членов не превосходила ε.

Чтобы получить значение e^x с машинной точностью, следует заменить ε нулем и записать команды в виде

8‴)	$\lvert 0\rvert-\lvert u\rvert=0$		
9‴)	*У1* 4)		

Так как слагаемые неограниченно убывают, то в конечном счете u выйдет из диапазона допустимых в машине чисел и сделается машинным нулем. В этом случае команда 8‴) выработает управляющий сигнал $\omega=0$ и команда 9‴) передаст управление на *стоп*.

Впрочем, той же цели можно достичь и более простым путем. Будем сначала записывать вновь полученную сумму в другую ячейку, сравнивать новую сумму со старой и прекращать вычисления, когда они совпадут. Это произойдет, когда u сделается весьма малым по сравнению с y, но много р а н ь ш е, чем u станет машинным нулем. Изменив команду 7), получим окончание такого итерационного цикла в виде:

7)	$y+u=R$	9)	*УО* $\overrightarrow{R}$ 4) y
8)	$y \nsim R=0$	10)	*стоп*

Арифметические и итерационные циклы объединяются общим названием *циклы без переменных команд*, в отличие от *циклов с переадресацией*, в которых команды программы в процессе ее выполнения изменяются самой машиной нужным образом. Познакомимся с примером цикла с переадресацией.

Допустим, что в нескольких ячейках памяти записаны числа $a_1, a_2, \ldots, a_n$, и нам нужно вычислить их произведение

$$P=a_1\times a_2\times \ldots \times a_n.$$

Первые две команды этой программы можно написать сразу:

1) „1“ $=P$
2) $P\times a_1=P$,

а перед написанием следующей придется задуматься. Следующей должна быть команда $P\times a_2=P$; понятно, что мы имеем дело с циклом, весьма напоминающим цикл при вычислении $y=x^n$, но там команда рабочей части в с е г д а имела вид $y\times x=y$, а здесь команды

изменяются, так как следующий сомножитель берется уже из другой ячейки.

Таким образом, необходимо позаботиться о том, чтобы в процессе выполнения программы машина сама изменяла выполняемые ею команды. Это можно сделать с помощью уже рассматривавшихся фиксированных действий. Обозначим буквой T команду

$$T:\ P\times a_1=P$$

и выполним команду

$$T+,(0,\ 1,\ 0)=T.$$

При ее выполнении содержимое ячейки T вызывается не в регистр команд устройства управления, что обычно происходит с командами, а в арифметическое устройство, как обычное число. Напомним, что (0, 1, 0) означает ячейку, содержащую нули во всех разрядах, кроме 13-го, в котором записана единица. В результате фиксированного сложения мы получим новое машинное слово, операционная часть которого та же, что и в ячейке T, а адресная отличается от адресной части T единицей второго адреса. Прочитанное как команда, оно означает уже

$$P\times a_2=P$$

(если только адрес a_2 на единицу больше адреса a_1, что было предположено заранее). Такое изменение команды самой машиной и называется *переадресацией*.

Нужную программу мы можем теперь написать в виде цикла с искусственным счетчиком k, который мы сделаем фиксированным, и эталоном (0,0, n), где n означает число сомножителей.

1)	„1“ $=P$	T	6) $(P\times a_1=P)$
2)	$(0,\ 0,\ 1)=k$		7) $k-,(0,\ 0,\ n)=0$
3)	Б T° 6) T		8) У1 4)
4)	$T+,(0,\ 1,\ 0)=T$		9) стоп
5)	$k+,(0,\ 0,\ 1)=k$	T°	10) $P\times a_1=P$

Команда 6), обозначенная буквой T, в процессе выполнения программы изменяется. Чтобы иметь возможность пустить программу снова, надо восстановить эту команду, приведя ее в первоначальное состояние. Удобнее всего сделать это в самом начале: записать начальное состояние команды T в ячейке T° (команда 10)) и в начале работы программы заслать требуемое содержимое на рабочее место в ячейку 6). Это и делается командой 3). Первоначально ячейку T можно оставлять пустой или заносить туда любое содержимое; все равно оно сотрется при выполнении команды 3). Поэтому, записывая в команде 6) ее первоначальный вид, мы взяли его в скобки.

Сравнивая рассмотренный пример с предыдущими, мы можем убедиться в том, что эта программа имеет очень много общего с ариф-

метическим циклом. Различие их лишь в том, что в арифметическом цикле нет изменяющихся команд; все команды арифметического цикла исполняются так, как они написаны с самого начала. В цикле же с переадресацией адреса некоторых (по крайней мере одной, но, возможно, и многих) команд изменяются в процессе работы программы. Благодаря этому оказывается возможным работать с последовательностью чисел (естественно, к о н е ч н о й), или, как принято говорить в программировании, с *массивом* чисел или ячеек, в которых они записаны.

Переадресация команд является частным случаем более общей работы с командами программы — *формированием команд*, с которым мы познакомимся в следующем параграфе. В некоторых машинах первого и во всех машинах второго и следующих поколений переадресация облегчается специальными *индексными регистрами* (или *регистрами адреса*), которые позволяют производить переадресацию непосредственно в устройство управления, не изменяя вида команды в памяти. Это облегчает программирование, хотя и усложняет обучение ему.

Существо работы с индексными регистрами состоит в следующем. *Исполнительные адреса*, т. е. адреса, с которыми фактически исполняется команда, образуются в устройстве управления непосредственно перед их исполнением путем *модификации по регистру*. Эта модификация состоит в прибавлении содержимого индекс-регистра (или его части) к адресу, написанному в команде, хранящейся в памяти. В отдельных частях команды записывается при этом номер индекс-регистра, который надо использовать, если в данной машине таких регистров несколько, и признаки, показывающие, н а д о л и модифицировать адреса данной команды, и если да, то какой адрес по какой части индекс-регистра модифицируется.

Так как содержимое индекс-регистра легко изменяется (специальными командами, входящими в число элементарных операций машины), то индекс-регистры

Таблица 1

	.0	.1	.2	.3	.4	.5	.6	.7
0.	$\Rightarrow$	$+$	$-$	$\|-\|$	:	$\times$	(+)	$\oplus$
1.	ВС	(1) РА <	РА <	+,	$\longrightarrow$,	$\curlywedge$	БВ	ДЗ
2.	КЗУ	+ (БО)	− (БО)	|− (БО)|	:(БО)	× (БО)	[+]	$\ominus$
3.	В	(1) РА $\geqslant$	РА $\geqslant$	−,	[$\longrightarrow$,]	$\curlywedge$ стоп	У1	ДВ
4.	[РА] <	+ (БН)	− (БН)	|− (БН)|	V^-	× (БН)	(−)	мл. разр.
5.	М (а)	(0) РА <	РА	,+	$\longrightarrow$	$\wedge$	Б	ИРП
6.	[РА] $\geqslant$	+ (БОН)	− (БОН)	| − (БОН)|	V^- (БО)	× (БОН)	[−]	$\rightleftarrows$
7.	М (б)	(0) РА $\geqslant$	[РА]	,−	[$\longrightarrow$]	$\vee$	У0	стоп

удобно использовать в качестве фиксированных счетчиков в циклах и для переадресации, изменяя не адрес в команде, записанный в ячейке памяти, который здесь остается неизменным, а содержимое индекс-регистра, по которому модифицируется адрес, благодаря чему исполнительный адрес изменяется нужным образом.

В заключение параграфа несколько слов о кодировании программы. Фактически может выполняться только программа, написанная на машинном языке. Перевод программы, написанной в содержательных обозначениях, на язык машины — *кодирование*, как указывалось в § 4, является простой механической процедурой: требуется в каждой команде заменить символ выполняемой операции ее кодом, а содержательные обозначения участвующих в команде величин их фактическими адресами.

Коды операций можно выписать на отдельную табличку и заглядывать в нее в случае надобности (см. табл. 1); для данной машины коды операций заданы раз и навсегда и от задачи к задаче не изменяются. Другое дело адреса: наименование и смысл переменных и их размещение в памяти от задачи к задаче могут существенно изменяться и поэтому для каждой задачи нужно составлять свою таблицу. Ее составление облегчается специальными бланками, на которых печатаются клетки с номерами ячеек памяти; в них и записывают обозначения величин, для которых отводится данная ячейка. Этот процесс называют *распределением памяти*. Бланки с адресами называют *памятками* или, чаще, *шпаргалками*. Для больших задач распределение памяти может оказаться уже не слишком простой задачей. В табл. 2 приведена часть шпаргалки с распределением памяти для последнего примера — цикла с переадресацией.

Таблица 2

3000 1)	3040 P	3100 a_1
3001 2)	3041 k	3101 a_2
3002 3)	3042	3102 ...
3003 4)	3043	3103
3004 5)	3044	3104
3005 6) T	3045	3105
3006 7)	3046	3106
3007 8)	3047	3107
3010 9)	3050	3110
3011 10) T°	3051	3111
3012 (0, 0, 1)	3052	3112
3013 (0, 1, 0)	3053	3113
3014 (0, 0, n)	3054	3114
3015 „1“	3055	3115
3016	3056	3116
3017	3057	3117

Содержательная и кодированная части программы пишутся на одном бланке специальной формы. Программа цикла с переадресацией, закодированная в соответствии со шпаргалкой табл. 2, приведена в табл. 3. По ней можно судить, для какой

Таблица 3

Составил Задача Блок Стр. Лист

				3000	A	1 Шифр
	„1“ = P	3000	75	0000	3015	3040
	(0, 0, 1) = k	1	75	0000	3012	3041
	Б T° $\overrightarrow{\text{T}}$ T	2	56	3011	3005	3005
Δ	T+, (0, 1, 0) = T	3	13	3005	3013	3005
	k +, (0, 0, 1) = k	4	13	3041	3012	3041
T	($P \times a_1 = P$)	5			н. п.	
	k —, (0, 0, n) = 0	6	33	3041	3014	0000
	У1 Δ	7	36	0000	3003	0000
	стоп	3010	77			
T°	$P \times a_1 = P$	1	05	3040	3100	3040
	(0, 0, 1)	2	00	0000	0000	0001

				3013	A	2 Шифр
	(0, 1, 0)	3013	00	0000	0001	0000
	(0, 0, n)	4	00	0000	0000	
	„1“	5	101	4000	0000	0000

Кодировала: **Перфорировала:**

Проверила кодировку: **Проверила карты:**

цели предназначена каждая графа бланка. Отметим только, что нумерация команд, поскольку в ней нет надобности, здесь опущена. Для команд, которым передается управление, оставлены специальные обозначения, помещенные в графе, предназначенной для номеров или условных обозначений. При написании программ эти обозначения часто заменяются проведением стрелок до команды, которой передается управление (если она на той же странице). Мы не делаем этого из-за трудностей набора.

§ 6. СТАНДАРТНЫЕ ПРОГРАММЫ. ФОРМИРОВАНИЕ КОМАНД

Достаточно написать несколько программ для несложных, но реальных (т. е. не придуманных для учебных целей) задач, чтобы понять, что большое число частных программ приходится писать и и использовать многократно. Стоит упомянуть хотя бы программы вычисления элементарных функций. Та же показательная функция e^x может входить в несколько различных формул; не переписывать же один и тот же арифметический или итерационный цикл несколько раз! Тем более, что вычислением элементарных функций дело не ограничивается. Существует большое число простых математических и программистских задач, которые входят составными элементами во многие реальные задачи — решение систем линейных уравнений, приближенное вычисление интегралов, работа с комплексными числами, перенос массива чисел из одних ячеек памяти в другие и т. п.

Естественно желание иметь для каждой из таких элементарных задач свою программу, написанную один раз и наиболее тщательным образом, и использовать ее затем столько раз, сколько потребуется. Такие программы называют *стандартными программами* или стандартными *подпрограммами*, учитывая, что они служат частями программ для решения тех или иных задач. Создание *библиотек стандартных подпрограмм* явилось первым шагом в автоматизации программирования, о чем мы еще будем говорить ниже. Библиотека стандартных подпрограмм является неотъемлемой частью математического обеспечения вычислительной машины или системы.

При работе со стандартными программами возникает несколько общих программистских проблем, с решением которых мы и познакомимся. Прежде всего, обращение к такой программе при помощи обычной команды передачи управления невозможно или по крайней мере неудобно. Действительно, после окончания работы стандартной программы управление должно быть передано снова в программу, которая к ней обращалась. При написании стандартной программы нельзя знать, где в памяти будет расположена обращающаяся к ней программа, тем более, что таких обращений может быть несколько и возвращаться придется в разные места. Поэтому перед тем, как написать команду передачи управления стандартной программе, необходимо обеспечить возврат, т. е. поместить в

конец стандартной программы команду передачи управления в нужное место основной программы.

Эта сложная операция облегчается использованием команды, специально приспособленной для таких целей, команды *безусловной передачи управления с возвратом*. Мы будем обозначать ее

$$БВ \overleftrightarrow{\quad a \quad b \quad c \quad}$$

При ее выполнении пройсходит передача управления ячейке b, а в ячейку c заносится не содержимое ячейки a, как в обычной команде передачи управления, а команда передачи управления ячейке a, т. е. $БВ\ 0\ \overrightarrow{a\ 0}$. С помощью этой команды мы можем одновременно передать управление стандартной программе и обеспечить, чтобы после окончания ее работы управление было передано в нужное место, записав в конец стандартной программы команду возврата.

Такое обращение к стандартной программе предполагает известным и ее начало и ее конец. Это неудобно, так как заставляет, в частности, фиксировать длину стандартной программы и препятствует ее модификации. Поэтому обычно поступают иначе: выделяют одну стандартную ячейку, которую используют в качестве «всеобщего конца» — мы будем обозначать эту ячейку буквой Ω — и при передаче управления стандартной программе, засылают команду возврата в эту ячейку Ω. Стандартная программа в этом случае должна заканчиваться передачей управления ячейке Ω, в которой уже будет заготовлена передача управления в нужное место основной программы.

Итак, обращение к стандартной программе должно иметь вид

$$БВ \quad \overleftrightarrow{\quad S \quad СП \quad \Omega \quad}$$

где $СП$ — адрес заголовка (начала) первой команды стандартной программы. Обычно возврат происходит в ячейку, следующую за командой обращения, т. е. обращение из ячейки $Я$ имеет вид

$$Я{:}БВ \quad \overleftrightarrow{\quad Я+1 \quad СП \quad \Omega \quad}$$

При таком стандартном обращении разрешается опускать подразумевающиеся левый и правый адреса и писать просто $БВ \quad СП$.

Следующей проблемой, с которой мы встречаемся при организации библиотеки стандартных программ, является проблема обмена информацией. Первой задачей, приводящей к идее написания стандартных программ, является задача вычисления элементарных функций. Как указывал еще Бэббедж в 1840 году, оказалось гораздо более выгодным не хранить в памяти машины таблицы элементарных функций, а вычислять нужное значение заново всякий раз, как это потребуется. Программа вычисления элементарной функции, например $\ln x$, должна быть стандартной. Вот здесь и возникает вопрос: как стандартная программа узнает, логарифм како-

го числа ей нужно вычислить? И как программа, обращающаяся за вычислением логарифма, узнает, чему же равен этот логарифм, вычисленный стандартной программой?

Обмен информацией между программами является одной из важнейших задач программирования, и мы не будем затрагивать его в полном объеме. Коснемся лишь двух основных возможных решений, относящихся к обмену информацией со стандартными библиотечными программами.

Наиболее просто идея обмена осуществляется в стандартных программах с входными и выходными ячейками. Применительно к программам для вычисления элементарных функций это означает следующее. Фиксируются две определенные ячейки, находящиеся в той части памяти, которая отведена для библиотеки стандартных программ и не используется никакими другими программами. Мы будем обозначать их через α и γ*. Стандартная программа для вычисления элементарной функции $\gamma=f(\alpha)$ пишется в предположении, что аргумент находится в ячейке α, а ответ — значение функции — помещается в ячейке γ. Программное осуществление этой идеи затруднений не вызывает.

Обращению к программе вычисления элементарной функции должна предшествовать в таком случае предварительная засылка аргумента в ячейку α. Местонахождение результата также известно и после получения его можно переслать в любую нужную ячейку или использовать любым нужным образом. Но и предварительную засылку и пересылку ответа после его получения должен предусмотреть программист в своей программе.

Такой способ обмена информации прост и удобен в тех случаях, когда обмениваемая информация состоит из одного, двух или, во всяком случае, очень небольшого количества чисел, и становится неприемлемым, если мы имеем дело с большим массивом или в тех случаях, когда обращение к стандартной программе происходит многократно. Необходимость частых засылок и пересылок отнимает очень много места и времени. В таких случаях целесообразно идти другим путем.

В качестве примера можно указать на стандартную программу вывода — перфорации или печати. В команде вывода необходимо указать адрес ячейки, содержимое которой выводится. Обычно при этом мы пользуемся д е с я т и ч н о й печатью, т. е. содержимое ячейки воспринимается как плавающее десятичное число; следовательно, прежде чем ставить команду печати, надо перевести полученный результат из двоичной системы, в которой производились вычисления, в десятичную. Естественно объединить все эти действия в одной, раз и навсегда написанной, стандартной программе печати.

---

* Иногда аргумента или ответа может быть два; например, удобно писать программу, одновременно вычисляющую $\cos\alpha$ и $\sin\alpha$. Тогда пользуются обозначениями α_0, α_1 и γ_0, γ_1, причем $\alpha_0{=}\alpha$ и $\gamma_0{=}\gamma$.

Было бы, однако, крайне неудобным пользоваться программой, печатающей всегда одни и те же ячейки. Стандартная программа печати должна строиться таким образом, чтобы программист имел возможность указать любые ячейки, содержимое которых эта стандартная программа могла обработать и выдать на печать. Такого рода программы носят название *стандартных программ с информацией*. При обращении к такой программе указываются определенным образом ячейки, содержащие обмениваемую информацию.

§ 7. СТРУКТУРА ПРОГРАММЫ

Рассмотренные нами до сих пор примеры программ были более чем элементарны и носили легко бросающийся в глаза искусственный, учебный характер. Программы реальных задач гораздо сложнее и требуют значительно больше выдумки и изобретательности. Мы лишены возможности приводить здесь такие примеры и сделаем лишь несколько замечаний о структуре общих программ.

Любая задача обычно распадается на некоторое число частных подзадач, которые имеют свои определенные цели; эти цели должны быть осуществлены частями программы, которые мы называем *блоками*. Важную задачу программирования составляет правильное расположение и правильная последовательность действия блоков. Это достигается при помощи так называемой *блок-программы* или *собирающей программы* (иногда говорят просто *собирающей*), которая содержит команды передачи управления блокам программы в нужной последовательности и проверки различных условий, если некоторые блоки должны работать лишь при их соблюдении или если осуществление каких-либо частичных целей предполагает разветвляющийся вычислительный процесс. В свою очередь программы некоторых блоков также могут представлять собой *собирающие*, передающие управление своим подблокам, осуществляющим свои частные подзадачи.

Написание собирающих программ есть важный этап программирования, обеспечивающий правильное расположение требуемых вычислений и правильную программную реализацию алгоритма решения задачи. Для облегчения этой работы алгоритм обычно заранее расчленяют на части, составляя *блок-схему* алгоритма; по нему и пишется блок-программа. Собственно говоря, блок-программа задачи

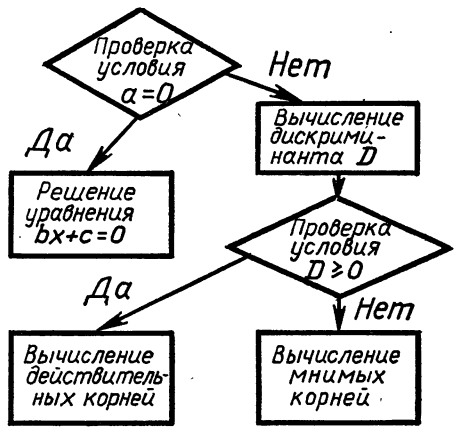


Рис. 85

и есть блок-схема алгоритма ее решения, осуществленная не с помощью прямоугольников, соединенных линиями на бумаге, а с помощью подпрограмм и команд передачи управления.

Приведем пример. Пусть требуется написать программу для решения квадратного уравнения $ax^2+bx+c=0$ для любых значений коэффициентов. Блок-схему алгоритма легко представить в виде, показанном на рисунке 85. Переведя ее на язык команд, мы получим блок-программу, которая в содержательных обозначениях для трехадресной машины будет выглядеть так:

1) $a \times 0 = 0$
2) *У*0 4)
3) ⟵ 8) *Линейное уравнение* Ω
4) *Дискриминант*
5) *У*1 7)
6) ⟵ 8) *Действительные корни* Ω
7) *Мнимые корни*
8) *стоп*

В самом деле, команда 1) проверяет выполнение условия $a=0$. Если это так, то в результате выполнения этой команды выработается управляющий сигнал $\omega=1$. Тогда команда 2) передаст управление команде 3). Последняя представляет собой передачу управления блоку, цель которого — решение линейного уравнения, поскольку при $a=0$ квадратное уравнение вырождается в линейное уравнение $bx+c=0$. Заметим, кстати, что этот блок также следует начать с проверки условия $b\neq 0$.

Уход на блок решения линейного уравнения осуществляется при помощи команды передачи управления с возвратом. Наименование блока означает, что в среднем адресе команды 2) при кодировке следует поставить адрес начала программы соответствующего блока. Команда возврата, как всегда при уходе на блок или стандартную подпрограмму, засылается в стандартную ячейку Ω. Так как в случае $a=0$ работа программы должна закончиться решением линейного уравнения, то в ячейку Ω мы засылаем команду возврата сразу на конец нашей программы — команду 8).

Если $a\neq 0$, то команда 1) выработает управляющий сигнал $\omega=0$ и команда 2) передаст управление команде 4). Здесь начинается программа собственно решения квадратного уравнения. Естественно, что прежде всего необходимо вычислить дискриминант уравнения, в чем и заключается задача соответствующего блока. При передаче управления блоку *Дискриминант* в Ω записывается команда возврата на следующую ячейку 5). Как уже говорилось в предыдущем параграфе, в этом случае Ω и 5) можно не писать: они подразумеваются и при кодировке будут поставлены автоматически.

Команда 5) несколько условна. Ясно, что здесь следует проверять знак дискриминанта. Если последняя команда в блоке вычисления дискриминанта имела вид $R_1 - R_2 = D$, где $R_1 = b^2$ и $R_2 = 4\,ac$, то команда 5) делает то, что нужно, поскольку при $D < 0$ имеем $\omega = 1$, а команда безусловной передачи управления, которая будет выполняться между блоком и командой 5), не изменяет управляющего сигнала. Вообще говоря, блок «Дискриминант» мог бы заканчиваться как-либо иначе; тогда перед командой 5) в блок-программе следовало бы поместить команду $D + 0 = D$.

Остальные команды блок-программы пояснений не требуют. Отметим только, что при уходе на блок «Действительные корни» необходимо возвращаться не на следующую команду, а прямо на конец программы 8). Поэтому в команде 6) крайние адреса указаны, тогда как в команде 7) они снова опущены, как и в 4),— после работы блока *Мнимые корни* возвращаться следует в ячейку $Я + 1$.

§ 8. ЯЗЫК «АВТОКОД 1:1»

Рассмотренный нами в нескольких предыдущих параграфах язык содержательных обозначений позволяет достаточно просто писать и читать написанную программу. При написании такой программы алгоритм решения задачи расписывается в виде последовательности элементарных операций (команд) машины. Однако программа, написанная в содержательных обозначениях, не может выполняться машиной непосредственно; необходимо сначала распределить память и закодировать программу (вручную), после чего ее можно уже набивать на перфокартах, вводить в машину и выполнять.

Как уже замечалось в § 5, после распределения памяти кодирование представляет собою простую процедуру, выполняемую по формальным правилам. Распределение памяти в некоторых случаях (для задач, требующих больших массивов или с логически сложными и большими программами) может вызывать серьезные затруднения, но для большинства обычных задач средней сложности может выполняться почти механически. Поэтому естественно поставить вопрос о возможности выполнения кодирования самой машиной.

Для этого, прежде всего, нужно иметь возможность ввести в память машины программу, написанную в содержательных обозначениях. Представим себе, что мы имеем перфоратор, клавиатура которого содержит, кроме цифр, еще и буквы алфавита, на котором пишется программа в содержательных обозначениях и все требуемые знаки (например, знаки арифметических действий). Чтобы различным знакам соответствовали различные пробивки, надо каждому знаку ставить в соответствие достаточно большую группу разрядов. Обычно их берут шесть или семь.

Алфавитно-цифровой перфоратор переводит каждый знак своей клавиатуры в семиразрядное двоичное число; с его помощью можно набить на картах, а затем ввести в машину программу, написанную

в содержательных обозначениях. Например, если принять соответствие

A	0100000
B	0100010
C	0110001
$+$	0001010
,	0001101
$=$	0010101

то команда

$$A+,\ B=C$$

изобразится следующим 45-разрядным машинным словом

000 0100000 0001010 0001101 0100010 0010101 0110001.

Ясно, что непосредственно выполнить такую команду нельзя, так как структура этого машинного слова отличается от той, которую должна иметь эта команда в машинном коде. Однако можно произвести преобразование структуры команды с помощью специальной преобразующей программы. Эта преобразующая программа должна обследовать каждую введенную команду, выделить в ней символ операции и заменить его кодом, поставленным в нужных разрядах ячейки. Кроме того, буквенные обозначения адресов должны быть заменены истинными адресами ячеек памяти. Это может быть сделано по введенной в память машины шпаргалке, если память распределена программистом вручную или же самой преобразующей программой.

Способ кодирования, при котором кодирование осуществляется специальной программой, называют *автоматическим кодированием*. Программа, осуществляющая кодирование, т. е. перевод программы с языка содержательных обозначений на язык машинных кодов, называется *автокодировщиком* или *транслятором (переводчиком)*; процесс кодирования называется в этом случае *трансляцией (переводом)*.

Транслятор должен быть пригоден для трансляции любой программы. Поэтому необходимо в значительно большей, чем мы делали это ранее, степени стандартизировать и унифицировать обозначения и правила записи команд. Произведя эту работу, мы придем к некоторому новому языку программирования, являющемуся формализацией языка содержательных обозначений. Его называют *языком автоматического кодирования* или *Автокодом*. Так как каждой строке — отдельной команде — автокодной программы соответствует точно одна команда в кодах машины, то принято говорить об «Автокоде 1 : 1» (один к одному).

Как язык содержательных обозначений, так и язык «Автокод 1 : 1», получающийся его формализацией, тесно связаны со спецификой конкретной машины, для которой они предназначены. При переходе к другой машине может измениться система команд, поря-

док следования адресов и т. п. детали, что влечет естественные изменения в способах записи команд. Поэтому эти языки относятся к *машинно-ориентированным языкам*. Программа, написанная на машинно-ориентированном языке, не может выполняться на машине другого типа; она должна быть предварительно переписана на язык, понятный соответствующей машине, или, точнее говоря, соответствующему транслятору.

Мы не будем останавливаться подробно на правилах записи автокодных программ, укажем только несколько наиболее существенных соображений. Для транслятора важно, что все автокодные команды делятся на две группы по структуре их записи: команды со средним расположением кода операции и команды с левым расположением кода операции.

В командах со средним расположением кода операции символ операции, определяющий ее код, находится внутри команды, между обозначениями первого и второго адресов. Такая структура используется для записи арифметических, фиксированных, логических операций, сдвигов и т. п. действий над числами или машинными словами. Команда со средним расположением кода операции имеет вид

адрес, знак операции, адрес, знак равенства, адрес.

Команды с левым расположением кода записываются в виде

обозначение операции, адрес, адрес, адрес

и применяются для операций управления, работы с внешней памятью или с индексным регистром. В такой команде обозначения адресов должны разделяться запятыми, тогда как в команде со средним расположением кода операции адреса разделяются символом операции и знаком равенства.

Употребление символов операций в автокоде требует уточнения, для того чтобы сделать каждый символ однозначным. Так как запятая употребляется как разделитель между адресами, то при записи десятичного числа нельзя отделять целую часть от дробной запятой, как это обычно делается: транслятор воспримет запись 1,25 как два числа. В качестве разделителя здесь используется точка и число 1,25 записывают в виде 1.25. Но тогда нельзя использовать точку в качестве знака умножения, так что для этой цели используется только косой крест $\times$. Аналогично деление изображается лишь наклонной чертой: $a/b=c$.

Адреса в автокодной программе можно писать по-разному. Допускается на месте адреса писать фактический адрес, если он известен, т. е. соответствующее восьмеричное число. Например, можно писать $1205\times7413=a$. При трансляции такая автокодная команда преобразуется в команду 05 1205 7413 адрес a. В качестве адреса можно ставить некоторую десятичную константу, заключенную в круглые скобки, например (1.37). Транслятор переведет эту константу в двоичное число, отведет для нее некоторую ячейку

памяти и поставит адрес этой ячейки на нужное место при кодировке содержащей ее команды.

Чаще всего для условного изображения адресов, т. е. величин, участвующих в процессе решения задачи, в автокодной программе используются *идентификаторы*. В качестве идентификаторов разрешается использовать любую последовательность букв и цифр, начинающихся с буквы. В частности, идентификаторами могут служить те буквы, которыми обозначаются переменные в соответствующих формулах. Единственным ограничением на выбор идентификаторов в автокодной программе является запрещение использовать в роли идентификаторов комбинации, служащие условными обозначениями для операций, например *БВ*, *УО* и т. п.

Из сказанного ясно, что по крайней мере для трехадресных машин, о которых до сих пор у нас шла речь, автокодная программа мало отличается от программы, написанной на языке содержательных обозначений. Поэтому мы ограничимся уже сказанным и не станем приводить примеров автокодных программ. Впрочем, следует иметь в виду, что возможны и фактически существуют различные системы автокодов, даже и для трехадресных машин. Так, в некоторых автокодах не разрешается запись команды со средним расположением кода операции. В таких случаях, например, команду умножения $a \times b = c$ полагается писать в виде

$$\times a, \quad b, \quad c.$$

Иногда знак умножения полагается заменять кодом операции или условным буквенным обозначением, вроде «Умн» или *АУ* (арифметическое умножение). Все это, как легко понять, никак не меняет природы языка «Автокод 1 : 1».

Для машин меньшей адресности язык «Автокод 1 : 1» имеет аналогичный смысл и аналогичную структуру. Правда, для одноадресных машин язык содержательных обозначений менее удобен, а для двухадресных — труднее поддается формализации. Например, для одноадресных машин большое значение имеют операции посылки в сумматор из памяти или в память из сумматора. Применяемые в содержательных обозначениях символы посылок $\Rightarrow$ или $\Leftarrow$ на алфавитно-цифровых перфораторах обычно отсутствуют. Поэтому чаще всего в автокодах для одноадресных машин условные значения кодов операций составляются из букв.

Глава VI

АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ

§ 1. ПРОБЛЕМНО-ОРИЕНТИРОВАННЫЕ АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ

Машинно-ориентированные языки программирования позволяют использовать все особенности и возможности каждой конкретной машины. Однако они неудобны тем, что при переходе с одной машины на другую, программу, написанную на машинно-ориентированном языке, приходится полностью переписывать заново. Естественно желание создать такой язык программирования, который не требовал бы приспособления к языку конкретной машины.

Такой язык не может находиться в отношении 1 : 1 к машинному, поскольку системы команд различных машин различны, и то, что на одной машине может быть выполнено одной командой, на других может потребовать нескольких. Но это обстоятельство как раз не является недостатком, так как позволит, в частности, избавить программиста от такой скучной, хотя и легкой, работы, как расписывание формулы по элементарным операциям.

В настоящее время существует несколько сот различных языков программирования. Уже сам этот факт показывает, что единый, удобный для всех областей применения цифровых вычислительных машин, универсальный язык программирования все еще не создан. Работа в области создания языков программирования (их называют *языками автоматического программирования*, так как они позволяют автоматизировать выполнение некоторых задач программирования, например расписку формулы по командам, распределение памяти и кодирование, или *алгоритмическими языками*, поскольку предназначены для записи алгоритмов в той или иной форме) идет в основном в направлении создания языков, достаточно хорошо приспособленных для решения задач из конкретной области применения, т. е. задач определенного типа. Такие языки называют, в противоположность машинно-ориентированным языкам, рассматривавшимся ранее, *проблемно-ориентированными языками*.

Первой областью применения цифровых вычислительных машин явились научно-технические и инженерные расчеты (она же оста-

ется наиболее важной и по настоящее время). Поэтому первые проблемно-ориентированные языки программирования создавались для задач вычислительной математики; они же продолжают оставаться наиболее употребительными и сейчас.

Одним из самых первых проблемно-ориентированных языков программирования был язык ФОРТРАН, разработанный группой программистов под руководством Д. Бэкуса. Первая публикация о фортране относится к 1954 году. Название этого языка происходит от сокращения слов FORmula TRANslation — *преобразование (перевод, трансляция) формул*, что прямо указывает на его назначение. Фортран получил очень широкое распространение, сначала в Америке, а за последние годы и в Европе. За истекшие 20 лет возможности фортрана расширились; сейчас существует уже четвертый вариант этого языка, причем каждый следующий содержит все возможности предыдущего.

Другим распространенным языком программирования является международный алгоритмический язык АЛГОЛ, разработанный международной комиссией при участии П. Наура и Э. Дикстры. Название его происходит от сокращения слов ALGOrithmic Language *(алгоритмический язык)*. Наибольшую популярность приобрел вариант алгол-60, официальное сообщение о котором было опубликовано в 1960 году. Сейчас имеется уже новый вариант алгола, алгол-68, который, правда, не получил еще достаточно широкого распространения, хотя и имеет более широкие возможности.

Алгол-60 во многом схож с фортраном. Однако в ряде случаев решения, принятые при создании алгола, являются более общими и предоставляют больше возможностей и больше удобств для программистов-пользователей. По тем же причинам трансляция с алгола оказывается более сложной, чем с фортрана, и получающаяся программа на машинном языке после трансляции с алгола больше проигрывает в объеме и быстродействии по сравнению с программой на языке машины, нежели транслированная с фортрана.

Большая общность и более широкие возможности алгола по сравнению с фортраном привели к некоторой его переоценке. Вначале даже казалось, что алгол пригоден для программирования задач из всех областей применения цифровых вычислительных машин, так что его называли универсальным языком программирования. На самом деле, алгол лучше всего приспособлен для записи вычислительных алгоритмов, не требующих слишком сложных логических схем. Универсальным же он является только в том смысле, что не зависит от языка конкретной машины.

Фортран и алгол являются наиболее распространенными и используемыми языками программирования. Можно с уверенностью сказать, что область распространения каждого из них гораздо шире, чем область распространения всех остальных языков, вместе взятых. Из остальных языков мы упомянем лишь КОБОЛ, предназначенный, главным образом, для программирования задач экономического характера и отличающийся от алгола большим удобством

и гибкостью работы с большими массивами информации, характерными для экономических задач, и язык PL-1, носящий более универсальный характер, но не получивший пока у нас никакого распространения (ни одна из наших машин второго поколения не имеет работающего транслятора с языка PL-1).

Из сказанного выше понятно, почему в дальнейшем мы ограничимся рассмотрением лишь двух языков программирования — фортрана и алгола. Чтобы не излагать дважды общие обоим языкам идеи, правила и обозначения, изложение будем вести в известной мере параллельно. Многие символы и термины этих языков имеют источником своего происхождения английские слова. Это, однако, вовсе не означает, что для изучения языков программирования необходимо знание английского языка. Здесь используется небольшое число слов, точный смысл которых для языка программирования всякий раз выясняется достаточно четко и ясно; кстати сказать, он не всегда совпадает с точным переводом соответствующего слова на русский язык.

§ 2. ПЕРВОНАЧАЛЬНЫЕ СВЕДЕНИЯ О ФОРТРАНЕ. ОПЕРАТОР ПРИСВАИВАНИЯ

Создание фортрана предшествовало конструированию и производству специальных устройств для ввода алфавитно-цифровой информации. Авторы фортрана предполагали использование для ввода информации в машину пишущих машинок телетайпов. Поэтому в фортране разрешается употреблять лишь те знаки, которые есть на соответствующих машинках: заглавные буквы латинского алфавита, арабские цифры, знаки $+$, $-$, $=$, точка, запятая, открывающая и закрывающая скобки (только круглые!), наклонная черта и звездочка. Чтобы различать букву О и цифру 0, букву О при письме перечеркивают. Мы не делаем этого из-за трудностей набора.

Некоторые фортранные правила записи совпадают с автокодными. Например, для отделения дробной части десятичного числа от целой применяется точка, а не запятая. Деление обозначается наклонной чертой, благодаря чему любая формула может быть записана в одну строку. Для обозначения величин, входящих в формулы, применяются *идентификаторы*, как и в автокоде. В качестве идентификаторов можно использовать любую последовательность букв и цифр, начинающуюся с буквы. Ограничений здесь два: 1) нельзя использовать сочетания, совпадающие со стандартными функциями (например, SIN) или с *иероглифами* * *(ключевыми словами)*, и 2) длина идентификатора не должна превышать шести символов.

Программа, написанная на фортране, представляет собой последовательность *операторов* или *инструкций*. Отдельный оператор

* Об иероглифах (ключевых словах) мы поговорим подробнее в следующем параграфе.

может и не совпадать с командой машины, а соответствовать нескольким командам.

Одним из основных операторов фортрана является арифметический *оператор присваивания*. Символом оператора присваивания является знак равенства. Справа от знака равенства должно стоять арифметическое выражение, составленное из констант и переменных величин, соединенных знаками действий. Порядок выполнения операций определяется их старшинством по обычным правилам алгебры или имеющимися скобками. Скобки допускаются только круглые; правила расстановки скобок также совпадают с известными из алгебры; число открывающих и закрывающих скобок должно быть одинаковым. Знаком умножения служит звездочка *; символом возведения в степень — две звездочки **; об остальных операциях мы уже говорили.

Выполнение оператора присваивания состоит в том, что значение выражения, стоящего справа от знака равенства, вычисляется, и это значение присваивается переменной, обозначение которой стоит слева от знака равенства. Так, оператор

$$W=(A**2+B**3)/(X+Y)-(X-Y)$$

означает, что нужно вычислить значение выражения

$$\frac{A^2+B^3}{X+Y}-(X-Y)$$

и полученное значение присвоить переменной W. Отсюда, между прочим, следует, что все переменные, которые встречаются справа от знака равенства, должны к моменту выполнения операции присваивания иметь определенные числовые значения; в противном случае оператор не может быть исполнен.

Частным случаем рассмотренного оператора является присваивание некоторой переменной определенного числового значения. Этот оператор записывается в такой же форме, например,

$$A=1.25,$$

причем число справа записывается в десятичной системе. При трансляции программы эта константа будет самим транслятором переведена в двоичную систему. Подобная запись допускалась и в автокоде, с той только разницей (кроме другого порядка адресов), что в автокоде требовалось заключать десятичное число в скобки.

Одна и та же переменная может встречаться как слева, так и справа от знака равенства. Например, оператор присваивания

$$I=I+1$$

означает, что значение переменной I должно быть увеличено на единицу. Такие операторы весьма употребительны в программировании, об этом легко догадаться после знакомства с предыдущей главой, в частности с понятием цикла.

Фортран допускает запись двух типов чисел: *целых* и *действительных* (или *вещественных*). Разница между ними состоит в том, что целые числа переводятся в двоичную систему и обратно точно, тогда как действительные — только приближенно. Записывая действительное число 2.00, программист должен иметь в виду, что при печати этого числа он может получить 1.99. . .99 или 2.00. . .01. Обычно фортрановские трансляторы целые числа записывали в фиксированной форме, а действительные — в плавающей. В связи с этим в первых трех вариантах фортрана (включая ФОРТРАН-III) арифметические действия над числами, одно из которых целое, а другое действительное, запрещались. В последнем варианте (ФОРТРАН-IV) эти действия разрешены; для их выполнения транслятор переводит целое число в форму действительного.

Для десятичных констант различие между целыми и действительными числами устанавливается очень просто: действительные числа содержат в своей записи десятичную точку, а целые ее не содержат. Так, 1 или 32 — целые числа, тогда как 1.00 или 32.000 — действительные. Впрочем, нули после десятичной точки не обязательны. Действительное число 1 можно записать в виде 1.; таким образом, в первых вариантах фортрана оператор присваивания

$$A=0.25+3\ (?!)$$

записан неверно; его следует писать в виде

$$A=0.25+3.$$

так как теперь мы имеем сумму двух действительных чисел, а в первом случае — сумму действительного и целого. Также не обязателен нуль перед десятичной точкой: число 0.25 можно записать .25, опуская нуль целых.

Кроме записи действительного числа с явно указанной в ней десятичной точкой, фортран разрешает запись его в форме «с плавающей запятой». В этом случае записывается сначала мантисса числа, затем буква E, заменяющая основание и символ возведения его в степень, а затем порядок; последний должен быть целым и содержать не более двух десятичных цифр. Так, число 0.34×10^4 записывается в виде 0.34E4, а число 5.62×10^{-5} в виде 5.62E—5. Очевидно, первое из приведенных чисел является *нормализованным*, второе — нет. Как всегда, отсутствие знака перед числом означает, что оно положительно; при желании перед ним можно ставить знак +. Перед отрицательным числом знак — должен стоять обязательно. То же относится и к порядку: отсутствие знака после символа E означает, что порядок п о л о ж и т е л е н. Запись числа в форме с плавающей запятой (в форме E) автоматически означает, что оно действительное.

Иначе обстоит дело с переменными величинами, которые обозначаются идентификаторами. Для них принято следующее правило: переменные, идентификаторы которых начинаются одной из букв I, J, K, L, M, N, считаются целыми, все остальные — действитель-

ными. В последнем варианте фортрана разрешено отступать от этого правила. Тогда необходимо в начале программы дать описания переменных, указав тип каждого. Но на практике этим никогда или почти никогда не пользуются. Чаще всего, если действительную переменную удобно обозначить символом, начинающимся с буквы, обозначающей целую, например MASSE (масса), впереди него добавляют букву, изменяющую смысл идентификатора, например, пишут: AMASSE или XMASSE. Иногда при этом одну-две буквы в первоначальном слове опускают, чтобы не превзойти предельной длины идентификатора — 6 символов.

Арифметическое выражение может содержать некоторые стандартные функции, программы вычисления которых входят в библиотеку транслятора. При обращении к ним нужно брать аргумент в скобки; для тригонометрических функций аргумент предполагается в радианной мере. Показательная функция e^x обозначается EXP(X), логарифмическая — ALOG(X), арктангенс — ATAN(X). Кроме них, имеются еще две функции — ABS(X)$=|X|$ и SQRT(X)$=\sqrt{X}$.

Наряду с обычными переменными в фортране широко применяется переменная с индексами, используемая для обозначения массивов. Так как все записи должны производиться в строчку, то индекс элемента в фортране пишется после наименования переменной в скобках. Например, запись TAP (4) означает четвертый элемент (одномерного) массива TAP. Такое выражение может входить в качестве элемента вычисляемой формулы.

Индекс переменной в фортране должен быть целым числом или целой переменной. Имеет смысл запись X(K), но переменная K не может при этом принимать отрицательных или нулевых значений. Форма индекса может быть и несколько более общей. В качестве индекса разрешается одна из трех конструкций:

1) целая переменная $\pm$ целая константа, например A(K+2) или T(N—1);

2) положительная целая константа, умноженная на целую переменную, например S(2*1);

3) положительная целая константа, умноженная на целую переменную, $\pm$ целая константа, например U(3*M+2) или W(2*1—3).

В описанных формах необходимо строго соблюдать порядок расположения переменных и констант. Записи U(2+3*M) или U(M*3+2) являются недопустимыми, так как расположение отдельных элементов в них не соответствует описанным выше конструкциям.

Переменная в фортране может иметь и несколько индексов, т. е. изображать, скажем, двумерный массив (матрицу). В таком случае оба индекса пишутся после наименования переменной в скобках через запятую. Каждый из индексов может быть заданным целым числом или иметь вид описанной выше конструкции. Например, разрешаются записи вида M(3, 2), X(I, J), F(2*K—1, 3).

Как правило, фортранные трансляторы допускают использование действительных чисел, заданных с точностью семи значащих цифр, независимо от того, записаны они в фиксированной или плавающей форме. В последних вариантах фортрана имеется еще одна форма записи действительных чисел или действительных переменных — запись *с двойной точностью*. Здесь мантисса может иметь до 16 значащих цифр. Запись с двойной точностью (форма D) имеет такой же вид, что и обычная плавающая форма (форма E), только вместо E ставится буква D. Например, число *e* записывается в виде 2.718281828459045 D0, причем здесь символ D и нулевой порядок опустить нельзя. Кроме того, в последних вариантах фортрана можно рассматривать комплексные числа, записанные в виде пары действительных, но мы на этом останавливаться не будем.

§ 3. ПЕРВОНАЧАЛЬНЫЕ СВЕДЕНИЯ ОБ АЛГОЛЕ

Язык АЛГОЛ был создан позже фортрана, поэтому при создании алгола можно было учесть достоинства и недостатки уже используемого языка. Прежде всего, к моменту внедрения алгола уже имелись специальные входные перфораторы, поэтому отпала необходимость использовать звездочку вместо знака умножения. Умножение в алголе обозначается обычным косым крестом $\times$. Для возведения в степень используется специальный знак $\uparrow$; в остальном правила записи арифметических выражений остаются такими же, как и в фортране. Например, выражение

$$\frac{a^3+3abc}{x-y}$$

на алголе запишется в виде

$$(a\uparrow 3+3\times a\times b\times c)/(x-y).$$

Отсюда видно, что и алфавит алгола шире фортранного. В нем разрешается использовать не только прописные, но и строчные буквы *, так что A и a означают разные переменные. Кроме того, в алгольных идентификаторах разрешается использовать пробелы. Длина идентификатора в алголе не ограничивается (тем не менее некоторые алгольные т р а н с л я т о р ы предполагают, что идентификаторы имеют длину не свыше 6 или 7 символов).

Как и фортран, алгол различает числа *целые* и *действительные (вещественные)*. Однако алгольные трансляторы обычно и целые числа записывают в плавающей форме. Поэтому в отличие от фортрана в алголе целые числа являются частным случаем действительных и действия, в которых встречаются и те и другие, в алголе допускаются. Например, можно писать 3×5.75 или $2\uparrow 0.5$. В связи

* В языке АЛГАМС, который является одним из вариантов алгола, использование малых латинских букв не разрешается.

с этим, если в действительном числе стоит десятичная точка, то после нее непременно должна идти дробная часть; запись 2. , означающая на фортране действительное число 2, на алголе не допускается. Следует писать 2.0 или просто 2, без точки в конце. Запись действительного числа в плавающей форме применяется в алголе так же, как и в фортране, с той только разницей, что вместо символа E здесь используется специальный знак $_{10}$ — опущенная десятка, означающая, как и E в фортране, основание системы. После этого символа должен идти порядок числа, который обязан быть целым. Так, число $0{,}25 \cdot 10^4$ запишется на алголе в виде $0.25_{10}4$ или $.25_{10}4$.

Любой идентификатор, означающий арифметическую переменную, независимо от того, какой буквой он начинается, может обозначать как целое, так и действительное число. Чтобы установить тип величины, изображаемой данным идентификатором, последний должен быть описан. Для этого в начале блока, где действует данный идентификатор, помещается *указатель типа*, например **integer** *(целый)* или **real** *(действительный)* и перечисление идентификаторов, которые к этому типу относятся. В конце списка должен стоять знак ; (точка с запятой).

Указатель типа относится к числу *ключевых слов* или *иероглифов* языка, о чем мы бегло упоминали в предыдущем параграфе. Остановимся на этом немного подробнее. В каждом алгоритмическом языке, в частности и в фортране и алголе, имеется целый ряд понятий и терминов, которые должны восприниматься как единые символы, независимо от тех букв, из которых они состоят. К таким терминам относятся уже упомянутые **real** и **integer** и многие другие, например **if** *(если)* или **go to** *(перейти к ...)*, о которых речь будет ниже. Вначале даже предполагалось, что каждому такому символу будет соответствовать специальная отдельная клавиша входного перфоратора, однако эта идея реализована не была и эти термины по-прежнему составляются из отдельных букв. Их мы и называем *иероглифами* (этот термин был впервые предложен А. Л. Брудно). Все иероглифы алгола при печатании набираются жирным шрифтом, а при письме от руки — подчеркиваются. Иероглифы фортрана набираются и пишутся от руки прописными (большими) буквами прямым шрифтом.

Наряду с арифметическими (целыми или действительными) переменными алгол предусматривает возможность использования самостоятельных переменных, означающих истинность или ложность некоторого высказывания. Такие *логические* переменные называют в алголе *булевскими*. Чтобы указать, что данная переменная является булевской, используется указатель типа **boolean**. также являющийся иероглифом алгола.

Значениями булевской переменной, как мы знаем, могут служить 1 и 0. Но, чтобы отличать логические переменные от арифметических, в алголе введены в качестве значений булевских переменных два *булевских числа*, которые обозначаются так: **true** (*да*; буквальный перевод — *истина*) и **false** (*нет*; буквально — *ложь*). Эти символы также являются иероглифами и воспринимаются независимо от букв, которыми они изображаются. При распределении памяти транслятор отводит для булевской переменной один разряд в некоторой

ячейке памяти, где соответствующие булевские числа изображаются, естественно, обычными 1 или 0.

Булевские переменные могут входить в состав *булевских выражений*, которые образуются при помощи *булевских (логических)* операций. Алгол допускает следующие операции над булевскими переменными:

$\equiv$ *совпадает*	$\wedge$ *и*
$\supset$ *влечет*	$\neg$ *не*
$\vee$ *или*	

Результатом операции $a \equiv b$ будет **true**, если a=**true**, b=true или a=**false** и b=**false**. Операция $a \supset b$ часто называется *импликацией* и обозначается $a \rightarrow b$ (*если a, то b*). Она имеет значение **true** во всех случаях, кроме a=**true**, b=**false**. Остальные операции уже рассматривались нами в § 4 гл. III. В приведенном списке логические операции расположены в порядке возрастания старшинства — раньше всех выполняется операция $\neg$ (*не*). Для изменения порядка выполнения операций используются обычные круглые скобки, как и в арифметических выражениях.

Уже три последние логические операции образуют, как нам известно, функционально полную систему, поэтому любая логическая операция может быть выражена комбинацией имеющихся в алголе. Например, часто встречающаяся в программировании операция $a \not\equiv b$ (отрицание равнозначности) может быть представлена в виде $\neg (a \equiv b)$.

Источником булевских чисел в алгольных программах обычно служат *отношения*: два арифметических выражения, соединенные одним из *знаков отношения*. В алголе используются следующие знаки отношения:

$=$ *равно*	$\geqslant$ *больше или равно*
$\neq$ *не равно*	$<$ *меньше*
$>$ *больше*	$\leqslant$ *меньше или равно*

Примерами отношения являются:

$$a=3,\ x \geqslant y,\ p \neq q,\quad u \uparrow 2+v \uparrow 2 \leqslant r \uparrow 2,\quad \exp(x+y) < x/y.$$

Каждое такое отношение представляет собой булевское число, которое равно **true** или **false** в зависимости от конкретных значений входящих сюда арифметических переменных. В соответствии с этим булевские в ы р а ж е н и я описывают более сложные условия, например, $x<0 \vee x \geqslant 1$ — булевское выражение, имеющее значение **false** для x, принадлежащего полуинтервалу $0 \leqslant x < 1$ и **true** в противном случае.

Одним из наиболее употребительных операторов алгола, как и фортрана, является *оператор присваивания*. В алголе для него используется специальный символ — двоеточие и знак равенства, $a:=1$; или $r2:=x \uparrow 2+y \uparrow 2$; или $E:=x>0 \wedge x \leqslant 1$. После выражения, значение которого вычисляется, должен стоять знак; (точка с запятой), этот знак заканчивает каждый оператор алгола.

Если слева от символа присваивания стоит идентификатор, описанный как булевская переменная, то справа от этого символа должно находиться булевское выражение, как в последнем примере. Если же переменная, которой присваивается значение, арифметическая (целая или действительная), то справа должно находиться арифметическое выражение. Значение этого выражения будет самим транслятором преобразовано в тип переменной, стоящей слева.

Алгол, подобно фортрану, предусматривает возможность работы с массивами путем использования переменной с индексами, которая может быть как действительной или целой, так и булевской. Индексы в алголе пишутся в строке, вслед за идентификатором переменной, но в квадратных скобках, например $X[3]$, $Y[1,4]$ или $x1[i]$. При описании переменной с индексами используется иероглиф **array** *(массив)*, впереди которого должен находиться указатель типа, например **integer array**, **real array** или **boolean array**. Опускать разрешается только указатель **real**, так что просто **array** автоматически означает действительный массив. После идентификатора массива в его описании должны стоять в квадратных скобках граничные пары, указывающие границы изменения индексов. Компоненты граничной пары разделяются двоеточием, а сами пары, если их несколько, т. е. если массив не одномерный, запятыми. Например, возможны описания **integer array** $m[1 : 12]$, или **boolean array** $R[1 : 4, 2 : 8]$, или наконец **array** $x[i : j]$. В последнем случае значения переменных i и j к моменту появления в программе этого описания должны быть известны.

В отличие от фортрана индексы переменных в алголе могут принимать нулевые и целые отрицательные значения; например, возможно описание **array** $a[-5 : 0]$. Но значение второй компоненты граничной пары не может быть меньше значения первой; описание $b[0 : -2]$ неверно и недопустимо!

Роль индекса в алголе может играть любое арифметическое выражение, например, можно писать $x[a+b/c]$. Индексное выражение вычисляется в первую очередь, и за значение индекса принимается ближайшее целое. Необходимо только, чтобы оно находилось в интервале, определенном граничной парой при описании массива. Более того, индексное выражение само может являться переменной с индексами или содержать такие переменные; например, допустима конструкция $x[i[j]]$ или $y[i[3], 4]$, что не разрешается в фортране. Размерность массива, т. е. число индексов и соответственно граничных пар, при описании в алголе не ограничивается.

§ 4. РАЗВЕТВЛЯЮЩИЕСЯ ПРОГРАММЫ НА АЛГОЛЕ И НА ФОРТРАНЕ

Безусловная передача управления осуществляется и на алголе, и на фортране с помощью одного и того же *оператора перехода*, выражаемого иероглифом **go to** *(перейти к)* в алголе и GO TO — в фортране, поскольку там, как известно, не употребляются строч-

ные буквы. После этого иероглифа должна стоять метка оператора, которому передается управление или, в более общем случае, *именующее выражение*. Мы ограничимся пока рассмотрением *меток*. Помечать оператор на разных языках приходится уже по-разному.

На алголе *метка* ставится перед оператором, к которому она относится, и отделяется от него двоеточием. Например, в программе

$$K{:}\,x := x + y; \quad L{:}\,y := x - y; \quad \textbf{go to } K;$$

после выполнения операторов, помеченных метками K и L, управление будет снова передано оператору K. Правила алгола разрешают использовать в качестве метки любой идентификатор или целое число без знака. Однако алгольные трансляторы обычно запрещают последнюю возможность, оставляя для метки право быть произвольным идентификатором.

Фортран, напротив, использует в качестве метки только целое число без знака. Программа фортрана пишется на бланке, каждая строка которого имеет 80 пронумерованных слева направо позиций, в которых могут быть помещены различные символы языка. Оператор, входящий в программу, может писаться, начиная с 7 колонки; если его требуется пометить, то метка (номер) ставится в той же строке в колонках 1—5. Цифры в этих колонках всегда воспринимаются как метка оператора; буквы здесь не допускаются *. Так как на фортране положение метки фиксировано, то двоеточие между меткой и текстом оператора не ставится. Та же программа на фортране запишется так:

```
10   X = X + Y
12   Y = X - Y
     GO TO      10
```

Метки-номера, как видно из примера, можно выбирать совершенно произвольно.

Разветвление на алголе осуществляется при помощи единой и весьма общей конструкции *условного оператора*. Для записи этой конструкции используются три новых алгольных иероглифа: **if** *(если)*, **then** *(то)*, **else** *(иначе)*.

Условный оператор алгола может иметь одну из двух различных форм — *краткую*, или *неполную*, и *полную*. Краткая, или неполная, форма условного оператора имеет вид

$$\textbf{if } \langle \text{булевское выражение} \rangle \quad \textbf{then } \langle \text{безусловный оператор} \rangle;$$

Если обозначить булевское выражение буквой B, а безусловный оператор буквой P, то условный оператор в краткой форме можно записать в виде

$$\textbf{if } B \textbf{ then } P;\ Q;$$

* Кроме одного случая, о котором будет сказано в § 6.

где Q — следующий оператор программы, безусловный или условный. Отметим, что под *безусловным оператором* мы понимаем оператор одного из известных нам видов — присваивания или перехода.

Выполнение неполного условного оператора определяется значением булевского выражения B. Если B имеет значение B=**true** *(да)*, то оператор P выполняется. После него выполняется оператор Q, если только P не содержит оператора перехода, передающего управление в другое место программы. Если же B=**false** *(нет)*, то оператор P пропускается, а выполняется сразу оператор Q.

С помощью описанной конструкции можно написать некоторые простые программы. Рассмотрим, например, задачу нахождения наибольшего из чисел a, b. Ее можно решить с помощью такой алгольной программы:

$$M:=a;\ \textbf{if}\ a<b\ \textbf{then}\ M:=b;\ \textit{конец:}$$

Немногим сложнее другая программа, решающая задачу упорядочения: положить M равным большему, а m — меньшему из чисел a, b. Это делается программой

$$\textbf{if}\ a\geqslant b\ \textbf{then go to}\ K;\ M:=b;\ m:=a;\ \textbf{go to}\ \textit{конец};$$
$$K:M:=a;\ m:=b;\ \textit{конец:}$$

Работа этих программ достаточно проста и понятна. Сделаем несколько замечаний по поводу второй из них. Если условие $a\geqslant b$ выполнено, то оператор перехода передаст управление на оператор с меткой K; здесь выполнится присваивание $M:=a$, затем $m:=b$ и программа переходит на конец. Если же $a\geqslant b$ не выполняется, то оператор **go to** K пропускается. Вместо него будут выполняться присваивания $M:=b$; $m:=a$, после чего оператор перехода передаст управление на конец, обходя ненужный уже оператор K и следующий за ним.

Перейдем теперь к полной форме условного оператора. Пользуясь аналогичными предыдущим обозначениями, полную форму условного оператора можно записать в виде

$$\textbf{if}\ B\ \textbf{then}\ P\ \textbf{else}\ Q;\ R;$$

Здесь B — булевское выражение; P — безусловный оператор; Q и R — любые операторы, условные или безусловные. Предыдущая фраза означает, что оператор, стоящий между *то* и *иначе*, не может быть условным, т. е. не может начинаться с *если*. Иначе говоря, сочетание *то если* в алголе запрещено. В тех случаях, когда необходимость такой конструкции вызывается требованиями задачи, условный оператор, стоящий после *то*, следует заключать в скобки. Оператор, заключенный в скобки, полагается безусловным. Таким образом, конструкция *то (если. . .)* правилами алгола допускается.

Легко догадаться, как выполняется полный условный оператор: если булевское выражение B имеет значение B=**true** *(да)*, то выполняется оператор P, а после него — оператор R; оператор Q

пропускается. Наоборот, при B=**false** *(нет)* пропускается оператор P, выполняется Q, а после него R. Впрочем, в обоих случаях оператор R будет выполняться, если P (соответственно Q) не содержит передачи управления в другое место программы.

Использование полной формы условного оператора позволяет упростить программирование некоторых задач. Например, выбор наибольшего из чисел a, b может быть выполнен с помощью одного оператора

$$\textbf{if } a \geqslant b \textbf{ then } M := a \textbf{ else } M := b;$$

Этой же формой условного оператора удобно пользоваться при программировании счета функции, заданной на разных участках разными формулами. Например, вычисление функции

$$u = F(y) = y^3 - y + 5,$$

где

$$y = \begin{cases} x^2 + 2x + 2 & \text{при } x > 2, \\ 2 & \text{при } |x| \leqslant 2, \\ -x^2 - x - 4 & \text{при } x < -2 \end{cases}$$

можно осуществить с помощью программы

$$\textbf{if } x > 2 \textbf{ then } y := x \uparrow 2 + 2 \times x + 2 \textbf{ else if } x > -2$$
$$\textbf{then } y := 2 \textbf{ else } y := -x \uparrow 2 - x - 4;\ u := y \uparrow 3 - y + 5;$$

Обе формы условного оператора позволяют при соблюдении (или несоблюдении) некоторого условия пропустить или выполнить некоторый о д и н оператор. Как быть, если нужно пропустить н е с к о л ь к о? Для этого их достаточно объединить в *составной* оператор при помощи *операторных скобок*. Они состоят из двух иероглифов алгола **begin** *(начало)* и **end** *(конец)*. Весь текст между этими символами рассматривается как единый составной оператор. Например, программа упорядочения двух чисел при помощи операторных скобок может быть записана так:

$$\textbf{if } a \geqslant b \textbf{ then begin } M := a;\ m := b \textbf{ end else}$$
$$\textbf{begin } M := b;\ m := a \textbf{ end};$$

При помощи операторных скобок **begin** и **end** можно объединить в составной оператор любую последовательность алгольных операторов. В частности, любой из операторов, входящих в составной, может быть также составным оператором.

Язык фортран не содержит такой общей конструкции, как условный оператор алгола, и в организации условных передач управления он ближе к непосредственному программированию. Обычно используемыми в фортране инструкциями для условной передачи управления являются *арифметическая* условная передача управления и *логическая* условная передача управления. Эти инструкции имеют существенные различия, но обе содержат иероглиф фортрана IF *(если)* и поэтому их кратко называют *арифметическая* IF и *логическая* IF.

Арифметическая IF описывается конструкцией вида

$$\text{IF}\,(A)\ \text{I},\ \text{J},\ \text{K},$$

где A — арифметическое выражение, а I, J, K — метки, т. е. целые числа без знака — номера соответствующих инструкций. Например, возможно выражение

$$\text{IF}\,(\text{X}*\text{Y})\ 5,\ 7,\ 14.$$

Выполнение этой инструкции происходит следующим образом. Прежде всего, вычисляется арифметическое выражение A (в примере — произведение X*Y). Далее, если имеет место неравенство $A<0$, то управление передается оператору, помеченному меткой I (соответственно при $\text{X}*\text{Y}<0$ управление передается оператору 5), при $A=0$ управление передается оператору J (соответственно при $\text{X}*\text{Y}=0$ — оператору 7), а при $A>0$ — оператору K (соответственно при $\text{X}*\text{Y}>0$ — оператору 14).

Заметим, что метки I, J, K не обязаны быть различными. Некоторые из них могут совпадать. Например, программа вычисления функции $u=F(y)$, которую мы рассматривали на стр. 175, запишется с помощью арифметической IF так:

```
      IF (X—2.) 2, 2, 1
1     Y = X ** 2—2. * X + 2.
      GO  TO 3
2     IF (X + 2.) 4, 5, 5
5     Y = 2.
      GO  TO 3
4     Y = — X ** 2— X — 4.
3     U = Y ** 3— Y + 5.
```

Логическая IF оказывается уже ближе к алгольной конструкции условного оператора в неполной форме. Она имеет вид

$$\text{IF}(B)\ S,$$

где B — логическое выражение, а S — произвольный оператор (инструкция) фортрана, кроме IF (и оператора цикла DO, о котором речь будет идти в следующих параграфах). Простейшие логические выражения получаются в фортране путем использования отношений, которые мы определили в предыдущем параграфе. В фортране используются те же отношения, что и в алголе, однако они обозначаются другими символами, что связано, как мы уже говорили в § 2, с отсутствием соответствующих символов на клавиатуре пишущей машинки. Список символов отношений, используемых в фортране, приводится ниже в таблице.

Название	Алгебраическое обозначение	Обозначение на фортране
Равно	$=$	.EQ.
Не равно	$\neq$	.NE.
Больше	$>$	.GT.
Больше или равно	$\geqslant$	.GE.
Меньше	$<$	.LT.
Меньше или равно	$\leqslant$	.LE.

Точки, окаймляющие обозначения операторов отношения, являются обязательными. Наряду с отношениями, которые представляют собой два арифметических выражения, соединенные оператором отношения, в фортране могут использоваться и логические константы .TRUE. и .FALSE. , смысл которых известен. Точки в их написании также обязательны.

Более сложные логические выражения могут быть получены, как и в алголе, путем использования логических операций. Фортран допускает применение трех основных логических операций: *и*, *или* и *не*, которые в алголе обозначались через $\wedge$, $\vee$, $\neg$, а в фортране обозначаются соответственно символами .AND., .OR., .NOT. .

Рассмотрев понятие логического выражения фортрана, возвратимся к логическому IF. Его выполнение соответствует выполнению условного оператора алгола в краткой форме: если B=.TRUE., то оператор S выполняется, а если B=.FALSE., то он пропускается. Например, оператор

```
IF (X**2+Y**2.LE.4..AND.X.GE.0.AND.Y.GE.0) GO TO 7
```

для точки с координатами (X, Y) передаст управление оператору 7 тогда и только тогда, когда точка лежит в первой четверти круга радиуса 2 с центром в начале координат. В противном случае передача управления не произойдет, и программа перейдет к следующей инструкции. Действительно, логическое выражение в обычных обозначениях имеет вид:

$$(x^2+y^2\leqslant 4)\wedge(x\geqslant 0)\wedge(y\geqslant 0).$$

§ 5. ОПЕРАТОР ЦИКЛА

Как уже говорилось раньше, основу программирования составляет написание циклических программ. Разобранных в предыдущем параграфе приемов условной передачи управления вполне достаточно для организации любых циклов как на фортране, так и на алголе. Например, для рассмотренной в § 5 гл. V задачи нахождения суммы

$$S=1+1/2^2+1/3^2+\ldots+1/100^2 \qquad (1)$$

программа на алголе может быть написана так:

$n:=1$; $S:=1$; $M:n:=n+1$; $S:=S+1/n\uparrow 2$; **if** $n<100$ **go to** M;

Та же программа на фортране может быть написана следующим образом:

```
      N = 1
      S = 1.
1     N = N + 1
      R = N
      S = S + 1./R ** 2
      IF (N — 100) 1, 2, 2
2     продолжение
```

Присваивание R=N понадобилось нам для того, чтобы перевести целое число N, являющееся здесь параметром цикла, в действительную форму. В остальном обе программы достаточно просты и будут понятны читателю без дальнейших комментариев. Однако ввиду особой важности и широкого использования циклов для программирования авторы обоих рассматриваемых нами языков позаботились о существовании специальной конструкции — *оператора цикла*, который существенно облегчает программирование циклов на этих языках. Для этой цели используется алгольный иероглиф **do** (на фортране DO). В точном переводе с английского это слово означает *делай, выполняй*. Но так как этот иероглиф связан именно с циклическими программами и встречается только при написании циклов, то в большинстве наших руководств этот термин переводится на русский язык как *цикл*.

Рассмотрим структуру и выполнение оператора цикла сначала на алголе. Общий вид оператора цикла на алголе можно представить в виде

$$\textbf{for}\ k:=E_1,\ E_2,\ \ldots,\ E_n\ \textbf{do}\ S;$$

Здесь **for** (для) и **do** (цикл) — вновь вводимые иероглифы алгола, k — идентификатор переменной, которую называют *параметром цикла*, S — алгольный оператор, E_1, E_2, . . ., E_n составляют *элементы цикла*, которыми определяется характер выполнения оператора цикла в каждом из отдельных случаев.

Элементы цикла бывают трех различных типов: *арифметического, прогрессии* и *пересчета*. Познакомимся с каждым из них подробнее.

Элемент цикла *арифметического типа* представляет собой произвольное арифметическое выражение. В процессе выполнения цикла с элементами арифметического типа происходит присваивание параметру цикла значения очередного арифметического выражения, после чего выполняется оператор S, написанный после иероглифа **do**. Например, пусть требуется вычислить произведение

$$P=f(-1.5)\times f(3)\times f(a^2+1)\times f\left(\frac{a+b}{a-b}\right),$$

где a и b — известные числа, а функция $f(x)$ задана формулой

$$f(x)=\frac{x^2}{2}+e^x\cos x.$$

Величину P можно получить с помощью такой программы:

$$P:=1;\ \textbf{for}\ x:=-1.5,\ 3,\ a\uparrow 2+1,\ (a+b)/(a-b)\ \textbf{do}$$
$$P:=P\times(x\uparrow 2/2+\exp(x)\times\cos(x));$$

Выполнение оператора цикла состоит здесь в том, что параметру x присваиваются поочередно перечисленные значения, после чего выполняется оператор присваивания, в котором вычисляется значение функции в нужной точке, и переменной P присваивается значение, равное произведению всех предыдущих на вновь полученное.

Значительно большие возможности предоставляет использование элемента цикла *типа прогрессии*. Оператор цикла с одним элементом типа прогрессии имеет вид:

$$\textbf{for}\ k:=A\ \textbf{step}\ B\ \textbf{until}\ C\ \textbf{do}\ S,$$

где **step** *(шаг)* и **until** *(до)* — новые иероглифы. Выполнение оператора цикла в этом случае можно описать так. Параметру цикла присваивается значение A и выполняется оператор S. Затем значение параметра увеличивается на B и снова выполняется оператор S и т. д. до тех пор, пока значение параметра не выйдет за заданную границу C.

Используя цикл с элементом типа прогрессии, мы можем написать программу для вычисления суммы (1), приведенную в начале параграфа, в виде:

$$S:=0;\ \textbf{for}\ n:=1\ \textbf{step}\ 1\ \textbf{until}\ 100\ \textbf{do}\ S:=S+1/n\uparrow 2;$$

Другим примером такого же цикла является получение компонентов вектора, равного сумме двух заданных n-мерных векторов. Это можно сделать с помощью программы

$$\textbf{for}\ i:=1\ \textbf{step}\ 1\ \textbf{until}\ n\ \textbf{do}\ c[i]:=a[i]+b[i];$$

Отметим, между прочим, что здесь параметром цикла является и н д е к с. Легко понять, что этот цикл представляет собой ц и к л с п е р е а д р е с а ц и е й; отсюда видно, что на алголе циклы с переадресацией не отличаются от арифметических циклов.

Шаг параметра цикла может быть отличен от 1, а начальное или конечное значения не обязаны быть целыми. Поэтому значения параметра цикла могут никогда не совпадать с указанным последним значением и следует уточнить, как именно производится в цикле проверка окончания. Для цикла с элементом типа прогрессии

каждый раз проверяется выполнение условия *

$$\operatorname{sign}(C-k)\times B\geqslant 0$$

и оператор S выполняется лишь тогда, когда приведенное булевское выражение имеет значение **true**. Скажем, в примере

for $i:=5$ **step** 1 **until** 2 **do** S

оператор S не выполнится ни разу, так как $\operatorname{sign}(2-5)\times 1<0$. В примере

for $i:=5$ **step** -1 **until** 5 **do** S

оператор S выполнится один раз, так как для $i=5$ получаем $\operatorname{sign}(5-5)\times(-1)=0$, а для следующего значения $i=4$ имеем $\operatorname{sing}(5-4)\times(-1)<0$. То же самое имело бы место и при другом знаке шага.

Не следует думать, что параметр цикла для элемента типа прогрессии принимает значения, непременно образующие арифметическую прогрессию. Так будет лишь в том случае, если шаг цикла является постоянным, что вовсе не предусматривается правилами. Вообще говоря, не только A, но также и B, и даже C в определении элемента цикла типа прогрессии являются арифметическими выражениями и, вообще говоря, пересчитываются заново при каждом шаге выполнения цикла. Например, возможен цикл

$U:=0$; **for** j:=1 **step** j **until** 16 **do** $U:=U+\mathrm{j}\uparrow 2$,

который подсчитает сумму

$$U=1+2^2+4^2+8^2+16^2=341.$$

Для элемента цикла *типа пересчета*, который имеет вид

for $k:=A$ **while** V **do** S;

где **while** *(пока)* — новый иероглиф алгола; A — арифметическое, а V— булевское выражение, выполнение цикла состоит в следующем: параметру цикла присваивается значение A и выполняется оператор S. Затем снова параметру цикла присваивается значение A и выполняется оператор S до тех пор, пока булевское выражение V сохраняет значение **true**. Значение арифметического выражения A может изменяться оператором S, так что параметр цикла при каждом исполнении вообще может принимать различные значения. Например, программа

$x:=a$; **for** $u:=(x+a/x)/2$ **while** $\operatorname{abs}(u-x)\geqslant_{10}-5$ **do** $x:=u$

вычисляет значение $\sqrt{a}$ по формуле Герона, причем итерационный процесс будет продолжаться до тех пор, пока соседние приближения различаются больше чем на 10^{-5}.

Цикл с элементом типа пересчета, как видно из приведенного примера, очень удобен для программирования итерационных про-

* Функция sign (x) (*знак* x) определяется равенствами

$$\operatorname{sign}(x)=\begin{cases}+1, & \text{если } x>0,\\ 0, & \text{если } x=0,\\ -1, & \text{если } x<0.\end{cases}$$

цессов, поэтому элемент типа пересчета иногда называют элементом *итеративного типа*. Арифметические циклы и циклы с переадресацией обычно лучше укладываются в схему циклов с элементами типа прогрессии.

Отметим некоторые особенности оператора цикла в алголе, которые следует иметь в виду при его использовании:

1) Один и тот же цикл может содержать несколько элементов различного типа.

Например, возможен цикл

for $x:=A, B, C, D$ **step** M **until** P, Q, R **while** W **do** S;

Здесь оператор цикла будет выполняться сначала для значений x, равных A, B, C, D, затем для $x=D+M$, $x=D+2M$, ... до тех пор, пока параметр x не дойдет до значения P, затем для $x=Q$ и $x=R$ до тех пор, пока булевское выражение W будет сохранять значение **true**.

2) В цикле выполняется лишь один оператор, стоящий после **do**. Если требуется выполнить в цикле несколько операторов, то их следует объединить в один составной оператор с помощью операторных скобок **begin . . . end**.

3) Оператор S, выполняющийся в цикле, может в свою очередь быть оператором цикла.

Например, для вычисления произведения

$$P=a_1(a_1+a_2)(a_1+a_2+a_3)\ldots(a_1+a_2+\ldots+a_n)$$

можно написать такую программу:

```
P:= 1; for i:= 1 step 1 until n do
    begin c[i]:= 0; for j:= 1 step 1 until i do
        c[i]:= c[i]+a[j]; P:= P×c[i];
    end;
```

Здесь оператор цикла состоит из трех операторов. Первый заносит нуль в очередной множитель; второй вычисляет этот множитель и сам является циклом, подсчитывающим требуемую сумму; наконец, третий умножает значение P на полученный множитель.

4) К оператору S, выполняющемуся в цикле, нельзя обратиться из другого оператора, не находящегося в этом же цикле. В цикл можно войти только через его начало.

5) После выполнения оператора цикла параметр цикла имеет неопределенное значение. Если же выход из цикла происходит не после его завершения, а с помощью оператора перехода до окончания цикла, то параметр цикла сохраняет свое значение.

Например, неверной является программа

for $i:=1$ **step** 1 **until** N **do** $a[i]:=\sin(x[i]); b[i]:=0;$

Оператор присваивания $b[i]:=0$, стоящий после цикла, выполниться не может, так как параметр цикла i не имеет предполагавшегося значения $N+1$, а является не-

определенным. Вместе с тем в программе

for $x:=A$ **step** 2 **until** B **do**
begin $a[x]:=a[x]+T$; **if** $a[x]=TK$ **then go to** M **end**;
$M:z:=x+p$;

выход из цикла произойдет при выполнении условия $a[x]=TK$ независимо от выполнения условия окончания цикла, и параметр цикла x при выходе сохранит свое значение.

6) Оператор цикла не является безусловным оператором и поэтому его нельзя писать в условном операторе после **then**. Если же это необходимо по существу задачи, то его следует заключить в скобки — обычные круглые или операторные **begin . . . end**.

Фортранный оператор цикла DO отличается от алгольного и по внешнему виду и в первую очередь по предоставляемым возможностям. В фортране нет циклов с элементами арифметического типа и типа пересчета, а есть только с о д н и м элементом типа прогрессии и с постоянным шагом. Параметр цикла в фортране должен быть обязательно целым и может изменяться только в сторону возрастания, зато число операторов, выполняемых в цикле, может быть произвольным. Сравнивая фортранный цикл с циклом в содержательных обозначениях, можно сказать, что фортранный цикл есть арифметический цикл с целочисленным счетчиком, меняющимся в прямом направлении.

Сказанное выше отнюдь не означает, что на фортране нельзя написать итерационного цикла. Речь идет совсем о другом: организацию итерационного цикла на фортране программист должен производить самостоятельно, используя условные передачи управления, тогда как на алголе эта работа может быть выполнена транслятором, если воспользоваться оператором цикла с итеративным элементом.

Обращение к оператору цикла на фортране имеет вид

DO M N=I, J, K.

Здесь N — параметр цикла, который должен быть целой переменной; I — начальное, а J — конечное его значения, причем должно выполняться условие $J \geqslant I$ (в противном случае цикл выполняется только один раз); K — шаг изменения параметра. Величины I, J, K могут быть как постоянными, например

DO M N=5, 73, 4,

так и простыми целыми переменными без индекса. В последнем случае все переменные должны принимать определенные значения до упоминания их в команде обращения к оператору цикла, так что в процессе выполнения цикла они остаются постоянными.

Если шаг параметра цикла равен единице, что случается довольно часто, то переменную K в обращении к оператору цикла можно опускать, так что обращение

DO M N=I, J

равносильно обращению DO M N=I, J, 1.

Буква M, стоящая между иероглифом DO и обозначением N параметра цикла, означает метку оператора, заканчивающего цикл. Иначе говоря, в цикле будет выполняться группа операторов (инструкций) до помеченного номером M включительно. При этом среди выполняемых в цикле операторов могут, в свою очередь, находиться другие циклы. Например, программа для вычисления произведения

$$P = a_1 (a_1 + a_2) (a_1 + a_2 + a_3) \ldots (a_1 + a_2 + \ldots + a_n)$$

на фортране может быть написана так:

```
      P = 1
      DO 1 I = 1, N
      C(I) = 0
      DO 2 J = 1, I
    2 C(I) = C(I) + A(J)
    1 P = P × C(I)
```

Инструкции, выполняемые внутри цикла, могут быть совершенно произвольными. Ограничение накладывается лишь на последнюю, ту самую, номер (метка) которой указывается в обращении к циклу после иероглифа DO. Эта последняя не должна быть инструкцией условной или безусловной передачи управления IF или GO TO. Если же это необходимо по смыслу задачи, то после инструкции передачи управления в качестве последней инструкции в цикле помещается иероглиф CONTINUE *(продолжать)*. Эта инструкция не исполняется и используется лишь в том случае, когда цикл заканчивается (или *может закончиться*) передачей управления.

Пусть, например, в массиве X (I) требуется найти номер первого отрицательного элемента. Это можно сделать при помощи следующей программы (как и в алголе, здесь параметр цикла приобретает неопределенное значение после окончания цикла, но сохраняет свое значение, если произошел выход из цикла до его завершения):

```
      DO 3 I = 1, N
      IF (X (I)) 7, 3, 3
    3 CONTINUE
    7 M = I
```

Здесь рабочая часть цикла состоит из одной передачи управления, поэтому инструкция CONTINUE необходима.

Найдем теперь в том же массиве X (I) наименьшее по абсолютной величине число. Это можно выполнить так:

```
      XMIN = 1. E 19
      DO 2 I = 1, N
      IF (ABS (X (I)) — XMIN) 2, 2, 3
    3 XMIN = X (I)
    2 CONTINUE
```

Формально последней написанной инструкцией в цикле не является передача управления. Однако при ABS (X (I))<XMIN нельзя обес-

печить возврата на начало цикла и смены значения параметра без использования инструкции CONTINUE, которая здесь, следовательно, также необходима; здесь программа может оканчиваться передачей управления.

При наличии в программе вложенных друг в друга циклов допускается, чтобы два цикла имели один и тот же конец. Например, программу пересылки n-мерной матрицы X в n-мерную матрицу Y можно написать так:

```
      DO 10 I = 1, N
      DO 10 J = 1, N
10  Y(I, J) = X(I, J)
```

§ 6. СТРУКТУРА ФОРТРАННОЙ ПРОГРАММЫ

При написании программ большое значение имеет возможность использования одних и тех же элементов несколько раз. Как правило, в задаче приходится несколько раз вычислять одни и те же выражения для различных аргументов или совершать однотипную обработку различных массивов — переносить их с одного места на другое, сортировать по каким-либо признакам, выбирать наибольший или наименьший элемент и т. п. Такого рода работа значительно облегчается при использовании *подпрограмм*, о которых мы уже говорили в предыдущей главе.

Фортранные программы допускают использование различных типов подпрограмм. Настоящий параграф посвящен знакомству со способами построения таких подпрограмм, способами обращения к ним и их использованию.

Наиболее простым случаем являются *библиотечные подпрограммы*. Сюда относятся подпрограммы, имеющиеся в составе транслятора; для обращения к ним достаточно упомянуть их название в тексте программы. Самыми распространенными и широко используемыми библиотечными программами являются программы вычисления элементарных функций, о которых мы уже говорили в § 2 настоящей главы. Стандартные обозначения таких функций могут встречаться в любом арифметическом выражении в составе оператора присваивания. Например, можно написать

```
X = A * SIN (OMEGA * T)/B — ALOG (T).
```

При вычислении значения арифметического выражения транслятор сам обратится к библиотечным программам вычисления синуса и логарифма (напомним, что программа вычисления логарифма обозначается ALOG, а не LOG, так как значением функции должно быть действительное, а не целое число).

Список библиотечных программ определяется транслятором и у разных трансляторов может быть различным. Обычно, кроме основных элементарных функций и функций ABS(X) и SQRT(X), о которых мы говорили в § 2, в число библиотечных подпрограмм

включаются еще некоторые подпрограммы, реализующие часто встречающиеся алгоритмы, например перевод целого числа в действительную форму или обратно (быть может, с выделением целой части), нахождение минимального или максимального элемента массива и т. п. Перечень библиотечных подпрограмм и их стандартные обозначения приводятся в руководстве для пользования конкретным транслятором.

Разумеется, список библиотечных подпрограмм не может быть слишком большим, так что ограничиться одними лишь библиотечными подпрограммами удается не часто, и программисту приходится писать собственные. Из них мы упомянем, прежде всего, *арифметические подпрограммы*, подобные арифметическим операторам присваивания.

Арифметическая подпрограмма отличается по внешнему виду от арифметического оператора присваивания тем, что в операторе присваивания слева от знака равенства стоит только идентификатор переменной, а в арифметической подпрограмме после наименования подпрограммы — список аргументов, заключенный в скобки, и лишь затем знак присваивания. Например,

```
FIC(X, Y)=(X-Y)/(X**2+Y**2).
```

Другим важным отличием арифметической подпрограммы от оператора присваивания является то, что она должна быть помещена один раз в самом начале программы перед всеми исполняемыми операторами. Ей может предшествовать лишь описание типов переменных, если оно имеется в программе. Таким образом, эта строка является описанием функции.

Поместив арифметическую подпрограмму требуемым образом в требуемом месте, мы приобретаем право обращаться с ней так же, как и с библиотечной подпрограммой. Аргументы арифметической подпрограммы рассматриваются как *искусственные* (или *формальные*) и при обращении заменяются на фактические. Например, поместив в начале программы написанную выше арифметическую подпрограмму FIC, мы можем в программе пользоваться оператором присваивания

```
Y=FIC(A, B)*FIC(U, V),
```

который сам дважды обратится к написанной подпрограмме, сначала с аргументами A и B вместо X, Y, а затем — с аргументами U, V. Нужно только иметь в виду, что арифметическая подпрограмма не может быть слишком сложной и должна состоять из о д н о г о оператора.

Другим типом собственных подпрограмм, представляющих гораздо более широкие возможности, нежели арифметические подпрограммы, являются *подпрограммы типа* FUNCTION. Обращение к подпрограммам этого типа не отличается от обращения к арифметическим подпрограммам, но написание самих программ и способ их оформления совсем другие.

Стандартная подпрограмма типа FUNCTION может состоять из любого числа операторов и представляет собой самостоятельную программу. Первая инструкция этой программы должна иметь вид

FUNCTION F(X,Y, ...).

На месте F должно находиться наименование подпрограммы, первая буква которого, как обычно, определяет тип вычисляемоего значения — целое или действительное. Далее, в скобках идет список аргументов, которые разделяются запятыми; впрочем, некоторые из аргументов могут оказаться, на самом деле, выходными величинами, если данная программа выдает их несколько.

Если выходная величина подпрограммы единственна, то ее наименование должно совпадать с наименованием подпрограммы. Подпрограмма должна содержать, следовательно, по крайней мере один оператор присваивания, в котором встречается эта переменная — слева от знака равенства. Если выходных величин несколько, то одна из них должна иметь наименование, совпадающее с наименованием подпрограммы.

Имеются еще два условия, которым должна удовлетворять стандартная подпрограмма типа FUNCTION. Последней исполняемой инструкцией такой подпрограммы должна быть специальная инструкция, выражаемая иероглифом RETURN *(возвратиться)*. Наличие этой инструкции обеспечивает возврат из данной подпрограммы в основную программу, из которой произошло обращение. При трансляции программы эта строка обеспечивает место для засылки команды возврата. Вследствие возможного использования передач управления оператор RETURN может находиться и в середине программы, однако в большинстве случаев он замыкает перечень операторов данной подпрограммы. Заметим, между прочим, что оператор RETURN следует рассматривать как оператор передачи управления. Поэтому если подпрограмма содержит цикл, то оператор RETURN не может быть последним оператором в цикле и его следует помещать после обязательной в таком случае инструкции CONTINUE. Наконец, последнее требование — текст стандартной подпрограммы типа FUNCTION должен заканчиваться неисполняемой инструкцией END, показывающей для транслятора границу текста данной подпрограммы.

Входными переменными стандартной подпрограммы типа FUNCTION могут быть как простые переменные, так и переменные с индексами, т. е. массивы. В последнем случае в программе необходимо присутствие еще, по крайней мере, одной неисполняемой инструкции, цель которой — указать размеры употребляемых массивов. Например, если переменные U и V из числа аргументов программы являются массивами, то после заголовочной строки подпрограммы следует поместить операторы вида

DIMENSION U(100)
DIMENSION V(150)

или, что то же самое, один оператор

```
DIMENSION U(100), V(150),
```

откуда видно, во-первых, что U и V являются массивами и, во-вторых, что максимально требуемый размер массива U есть 100 ячеек, а массива V — 150. Фактически используемые размеры могут оказаться и меньшими.

Массивы допускаются лишь в качестве входных переменных подпрограмм рассматриваемого типа. Выходными переменными могут быть только простые переменные; переменные с индексами служить выходными переменными в подпрограммах типа FUNCTION не могут.

Так, например, пусть требуется вычислять модуль n-мерного вектора, координаты которого заданы массивом X. Стандартную подпрограмму для вычисления модуля можно написать так * (предполагая, что размерность вектора никогда не превосходит 100):

```
      FUNCTION AMOD(X, N)
        DIMENSION X(100)
          S=0
          DO 1 I=1, N
    1   S=S+X(I)**2
          AMOD=SQRT(S)
          RETURN
          END
```

Чтобы использовать эту стандартную подпрограмму, программисту достаточно в своей программе написать AMOD(A, K), где A — фактический массив координат вектора и K — его фактическая размерность. Искусственные переменные X,N в подпрограмме будут заменены фактическими и произойдет вычисление модуля K-мерного вектора A и присвоение соответствующего значения переменной AMOD. При этом AMOD(A, K) следует сразу помещать в правой части соответствующего арифметического оператора присваивания, не выделяя обращения в отдельный оператор. Например, если нужно найти направляющие косинусы вектора, то можно писать (направляющие косинусы обозначаем через CSN)

```
      DO 1 I=1, K
    1 CSN(I)=A(I)/AMOD(A, K).
```

---

* Читателю должно быть понятно, что мы присоединили букву A к названию подпрограммы, чтобы ее результат воспринимался как действительное число. Правилами фортрана допускается и другая возможность

```
REAL FUNCTION MOD (X, N),
```

но ею практически не пользуются.

Обратим внимание на то, что и в написанном цикле, и в цикле подпрограммы AMOD, к которой он обращается, используются одни и те же номера операторов и один и тот же индекс I. Это не ошибка и никаких нарушений работы программ от этого не произойдет. Обе программы, и основная, и подпрограмма FUNCTION, совершенно независимы и могут использовать любые обозначения, хотя бы одни и те же.

Кроме библиотечных, арифметических и программ типа FUNCTION, в фортране используется еще один, четвертый, тип подпрограмм — *подпрограммы типа* SUBROUTINE. Последние имеют много общего с подпрограммами типа FUNCTION: и те и другие должны иметь одинаково формируемые наименования и список формальных (искусственных) аргументов, которые могут быть простыми переменными или массивами, одинаково описываемыми при помощи оператора DIMENSION. Оба типа подпрограмм должны иметь завершающие операторы RETURN и END, в равной мере независимые от основной программы, и могут использовать любые номера операторов или любые переменные, независимо от употребляющихся в основной программе.

Укажем теперь различия в работе с этими подпрограммами. Первым является то, что подпрограммы типа SUBROUTINE могут иметь в качестве выходных данных переменные с индексами, т. е. массивы, что для подпрограмм типа FUNCTION запрещается. Размерность выходных массивов должна быть описана оператором DIMENSION так же, как и входных. Другое отличие — в обращении. Обращение к подпрограммам типа SUBROUTINE в основной программе требует отдельной строчки и использования нового фортранного иероглифа CALL *(вызвать)*, т. е. должно иметь вид

CALL SUBROUTINE RS(X, Y, ...)

где RS — наименование подпрограммы, к которой происходит обращение, а X, Y, . . . — список фактических параметров, которыми будут замещены формальные (искусственные) параметры, использованные в тексте подпрограммы. Сам же текст подпрограмм как типа SUBROUTINE, так и типа FUNCTION может быть помещен в любом месте программы, в том числе и п о с л е обращения к соответствующим подпрограммам.

Нужно только иметь в виду, что при обращении к стандартным подпрограммам всех типов необходимо соблюдать соответствие типов фактических и искусственных переменных. Например, если в приведенной выше подпрограмме AMOD(X, N)первый аргумент означает действительный массив, а второй — целую простую переменную, то обращение AMOD(X, Y) во всяком случае является неверным, так как аргумент Y не может означать целой переменной. Тем более должно совпадать число аргументов в описании и обращении. Имеет значение также и порядок аргументов, поскольку замещение искусственных переменных фактическими производится точно в порядке написания без каких-либо перестановок.

Распределение памяти при программировании на машинном языке или в содержательных обозначениях делается вручную самим программистом. При программировании на автокоде распределение памяти можно возложить на транслятор или же выполнить вручную — по усмотрению программиста. Программирование на алгоритмических языках предполагает, что распределение памяти полностью выполняется транслятором. Тем не менее фортран сохраняет некоторую возможность участия программиста в распределении памяти.

Для этой цели в фортране имеются два специальных неисполняемых оператора, обозначаемых иероглифами EQUIVALENCE и COMMON. Эти же операторы применяются для обмена информацией между блоками программы.

Инструкция EQUIVALENCE позволяет обязать транслятор разместить в одной и той же ячейке памяти различные переменные данной программы. Например, инструкция

EQUIVALENCE (ALPHA, X, KOL)

означает, что трем перечисленным в скобках переменным следует отвести одну ячейку памяти. Этому совершенно не противоречит то обстоятельство, что две из них действительные, а одна — целая.

Указанную возможность можно использовать различными способами. Прежде всего, в прямом смысле, для экономии памяти. Если, скажем, переменная ALPHA используется в первой части программы в качестве промежуточного результата, который в дальнейшем уже не требуется, то эту ячейку можно использовать для другой промежуточной переменной, которая в начале не требовалась. Наряду с этим можно применять оператор EQUIVALENCE и для обмена информацией между блоками программы, указав, что действующие в этих блоках переменные, обозначенные различными идентификаторами, есть на самом деле одна и та же переменная.

Оператор EQUIVALENCE можно использовать одновременно для отождествления нескольких групп переменных. Например, можно писать

EQUIVALENCE (X, T, A), (Y, S, B), (M, N),

что равносильно трем отождествлениям — переменных X, T и A между собой, Y с S и B, M с N. Для правильного использования этого оператора надо иметь в виду, что отождествление хотя бы по одному элементу двух массивов влечет распространение соответствующего отождествления и на остальные элементы. Например,

EQUIVALENCE (X (2), Y (3))

автоматически означает, что отождествляются и помещаются в одной ячейке памяти не только X(2) и Y(3), но также и X (1) с Y(2), X (3) с Y(4) и т. д.

Другим оператором аналогичного назначения, который чаще всего используется для обмена информацией между программой

и подпрограммами типа SUBROUTINE, к которым программа обращается, является оператор COMMON. Этот оператор помещается дважды — один раз в основной программе перед обращением и второй — в подпрограмме сразу после заголовка. Если в программе написано

```
COMMON ALPHA, BETA
```

а в подпрограмме, к которой обратились,

```
COMMON X, Y,
```

то это означает, что переменные X, Y должны размещаться в памяти на тех же местах, что и переменные ALPHA, BETA, т. е. что X отождествляется с ALPHA, а Y — с BETA. Впрочем, оператор COMMON можно использовать для обмена информацией не только между основной программой и подпрограммой, но также и между двумя подпрограммами.

Легко понять, что с помощью оператора COMMON можно производить замену искусственных переменных фактическими. Поэтому в подпрограммах типа SUBROUTINE некоторые (формально можно и все) переменные можно исключить из списка формальных параметров и передать соответствующие значения через оператор COMMON.

Например, напишем подпрограмму для нахождения направляющих косинусов n-мерного вектора. Это не может быть подпрограмма типа FUNCTION, так как выходом ее является массив. Если выдавать также и модуль, то подпрограмму можно написать так:

```
      SUBROUTINE CSN (X, AMOD, N)
    DIMENSION X (100), CSN (100)
        S = 0
        DO 1 I = 1, N
    1   S = S + X (I) ** 2
        AMOD = SQRT (S)
        DO 2 I = 1, N
    2   CSN (I) = X (I)/AMOD
        RETURN
        END
```

При обращении к этой программе для вычисления направляющих косинусов K-мерного вектора A следует писать:

```
CALL SUBROUTINE CSN (A, D, K),
```

причем значение модуля вектора будет присвоено переменной D. Здесь обмен информацией идет через список параметров. Можно исключить, скажем, размерность из числа параметров и передавать

ее через оператор COMMON. Тогда обращение к подпрограмме должно выглядеть так:

```
CALL SUBROUTINE CSN (A, D)
COMMON K
```

а начало подпрограммы должно иметь вид:

```
SUBROUTINE CSN (X, AMOD)
DIMENSION X (100) CSN (100)
COMMON N
. . . . . .
```

Здесь передача координат вектора и выходных данных происходит через список параметров, а размерности — через оператор COMMON. Аналогично можно поступить с любыми другими входными или выходными переменными. Передаваемые через COMMON переменные, естественно, исключаются из списка параметров, так что список параметров после наименования программы может совсем исчезнуть, и мы получим подпрограмму без параметров.

Сделаем теперь несколько завершающих замечаний, касающихся организации фортранной программы в целом. Реальная программа обычно делится на самостоятельные и независимые *сегменты*. Роль первого из них обычно играет основная программа. Она может быть написана по типу собирающей и, во всяком случае, содержит ряд обращений к различным подпрограммам. В начале первого сегмента помещается оператор PROGRAM, который может сопровождаться наименованием программы. Затем идут различные неисполняемые инструкции, вроде описаний типов величин, если они есть, описаний размерностей и операторов типа COMMON или (и) EQUIVALENCE, а лишь затем — фактический текст программы, содержащий выполняемые операторы. Текст программы должен кончаться оператором END, а последним выполняемым оператором программы всегда является оператор STOP.

После остановки машины, вызванной исполнением оператора STOP, программу можно пустить вновь лишь с самого начала. Фортран допускает и другой способ остановки машины, с помощью оператора PAUSE. Выполнение этого оператора также вызывает остановку машины, но если после этого нажать пуск, то программа будет выполняться дальше, т. е. управление передается следующему оператору (инструкции) программы. Такие остановки удобно использовать в процессе отладки.

В тексте фортранной программы можно поместить комментарии, поясняющие программу и предназначенные не для транслятора, а для человека, ее читающего. Текст комментариев может размещаться в любом месте программы. При трансляции этот текст пропускается и не оказывает никакого действия ни на трансляцию, ни на выполнение программы. Указателем для транслятора, что данная строка занята не оператором фортранной программы, а текстом

комментария, который должен быть пропущен, служит буква С в первой левой колонке бланка. Это и есть тот единственный случай, когда в колонках 1—5 бланка разрешается писать букву, о чем говорилось в подстрочном примечании на стр. 173.

§ 7. ОПИСАНИЕ ВЕЛИЧИН В АЛГОЛЕ. БЛОКИ АЛГОЛЬНОЙ ПРОГРАММЫ

Выше уже указывалось, что все идентификаторы, встречающиеся в алгольной программе, должны быть описаны. Таким образом транслятор получает информацию для распределения памяти. На самом деле роль описаний в алголе еще более существенна, так как описания влияют и на характер выполнения некоторых операторов, и на структуру программы, и на обмен информацией между блоками программы. Поэтому остановимся на описаниях более подробно.

Заметим, что в алголе мы имеем возможность дать точное определение понятия *блока*. До сих пор мы пользовались этим термином, понимая под блоком некоторую часть программы, не определяя, какую именно. Для алгольной программы можно дать точное определение: *блоком* алгольной программы называется *составной оператор*, в начале которого помещены *описания действующих в нем идентификаторов*. Приведенное определение можно продолжить до определения термина «программа», который до сих пор мы понимали в известной мере интуитивно. *Программа — это составной оператор или блок, не содержащийся ни в каком другом составном операторе или блоке и не обращающийся к операторам, не содержащимся в нем.*

Описания переменных действуют лишь в том блоке, в начале которого они помещены. После выхода из блока память, отведенная для описанных в нем величин, освобождается, так что не только перестают действовать приведенные описания, но исчезают и значения переменных. Например, пусть программа состоит из двух отдельных, не вложенных друг в друга блоков, состоящих в выполнении операторов S_1 и S_2

```
begin real a, b, c; integer k; array x[1:10];
      S1;
end;
begin real i, k; boolean a; array x[1:10];
      S2;
end;
```

Тогда перед выполнением каждого блока память будет распределяться заново и в каждом блоке будут действовать свои переменные.

Так, идентификатор a в первом блоке означает действительную переменную и для нее будет отведена некоторая ячейка. Во втором блоке тот же идентификатор означает булевскую переменную и для

ее хранения будет отведен лишь один разряд. Аналогично идентификатор k в первом блоке описан, как целая переменная, а во втором — как действительная. Естественно, что a и k в разных блоках — это совсем различные переменные, не имеющие между собой ничего общего.

Но мы сейчас хотим подчеркнуть другое. И в том и в другом блоке действует массив $x[1 : 10]$ действительных чисел (так как перед **array** нет указателя типа, то следует считать, что он **real**, поскольку другие указатели типов опускать нельзя), и машина воспринимает эти два массива как различные. Если бы речь шла о программе, в первом блоке которой некоторый массив получается, а во втором — как-то обрабатывается, то написанная только что программа была бы для этой цели решительно непригодна! Передать информацию из одного блока алгольной программы в соседний с помощью переменных, описанных в этих блоках, невозможно.

Для выполнения указанной выше задачи необходимо включить два написанных блока в третий, который содержал бы их внутри себя, хотя и не имел бы ни одного выполняющегося оператора, кроме S_1 и S_2, выполняющихся во внутренних блоках. Программу следовало бы писать так:

```
begin array x[1:10];
      begin real a, b, c; integer k;
            S1;
      end;
      begin real i, k;      boolean a;
            S2;
      end
end;
```

Таким образом, внутри блока могут действовать переменные, описанные в данном блоке, либо в другом, объемлющем по отношению к данному. Первые называют *локальными* относительно данного блока, а вторые *глобальными* по отношению к нему. Локальные переменные действуют лишь внутри данного блока и после завершения его работы полностью исчезают. Передать информацию из одного блока в другой можно лишь при помощи переменных, идентификаторы которых являются глобальными по отношению к каждому из них.

Если один и тот же идентификатор встречается в наружном и внутреннем блоках, как в примере

```
A { begin integer i, k;
    . . . . . .
      B { begin integer k, r;
          . . . . . .
          end;
        . . . . . .
    end;
```

то во внутреннем блоке действует только локальная переменная, а глобальная переменная с тем же идентификатором в нем недопустима; эти переменные, вообще говоря, имеют различные значения. Так, в приведенном примере внутри блока *B* действует локальная переменная *k*, а значения переменной *k*, действовавшей в блоке *A*, ни использовать, ни изменить внутри блока *B* нельзя; она действует только вне *B*. Что же касается глобальной переменной *i*, то она действует во всем блоке *A*, в том числе внутри блока *B*, так что ее можно менять и (или) использовать.

Метка является единственной алгольной переменной, не требующей явного описания, которым служит ее появление между знаком ;, заканчивающим предыдущий оператор или описание, и знаком :, отделяющим ее от оператора, ею помеченного. Поэтому считается, что метка локализована в блоке, внутри которого она помещена. Следовательно, из одного блока нельзя обратиться к метке, находящейся внутри другого.

Правила алгола, как и фортрана, разрешают включение в текст программы комментариев, облегчающих понимание и использование программы другим программистом. Так как алгольная программа пишется не на бланке и специально отведенного места для признака того, что некоторый текст не следует воспринимать как исполняемые операторы, здесь нет, то для внесения комментариев в алгольный текст используются следующие положения.

Как правило, комментарий может быть помещен в любом месте программы (после знака ;), он предваряется специальным иероглифом **comment** (комментарий) и заканчивается знаком ;. При работе транслятора весь текст между этими знаками пропускается. Чаще всего комментарий помещают в начале или в конце какого-либо блока. В начале блока иерогриф **comment** и последующий текст комментария помещают непосредственно после **begin** ; в этом случае текст комментария может содержать любые знаки, слова и иероглифы алгола, кроме точки с запятой ; этот разделитель воспринимается как конец комментария. В конце блока комментарий можно помещать п о с л е **end**, причем здесь разрешается даже опускать иероглиф **comment**. Но в этом случае транслятор может воспринять как конец комментария не только точку с запятой, но также иероглифы **end** или **else**, если они встретятся.

§ 8. ПРОЦЕДУРЫ АЛГОЛА

О роли стандартных подпрограмм мы говорили уже достаточно подробно в § 6 предыдущей главы и в § 6 настоящей. В алголе вопросы использования стандартных подпрограмм решены с большой широтой и универсальностью при помощи понятия *процедуры*, в известной степени аналогичной подпрограммам фортрана.

Процедура должна иметь наименование, для указания которого используется идентификатор, и характеризуется совокупностью

формальных параметров, для обозначения которых также применяются идентификаторы. Описание процедуры состоит из *заголовка* и *тела процедуры*. Заголовок использует новый иероглиф алгола **procedure** (*процедура*) и имеет вид

procedure<название процедуры>$(x_1, x_2, \ldots, x_n)$; где $x_1, x_2, \ldots, x_n$ — обозначения формальных параметров. Например,

procedure *J* (*x*, *y*, *h*);
procedure *корень* (*x*, *y*, *eps*);

Следующий за заголовком процедуры простой или составной оператор или блок *Q* называется *телом процедуры*. Тело процедуры представляет собой описание вычислительного процесса над формальными параметрами. Например, описание процедуры вычисления модуля *n*-мерного вектора может иметь вид

procedure *Modul* (*x*, *n*, *mod*); **comment** идентификатор *mod* предназначен для значения выходной величины;
begin integer *i*, **real** *s*; *s*:= 0;
for *i*:= 1 **step** 1 **until** *n* **do** *s*:=*s*+*x* [*i*] ↑ 2; *mod*:= sqrt (*s*);
end;

Формальные параметры процедуры вполне аналогичны формальным или искусственным переменным фортранных подпрограмм типа SUBROUTINE или FUNCTION и описания не требуют. Вместе с тем в теле процедуры могут действовать некоторые внутренние *локальные параметры*, которые и описываются в теле. В приведенном примере телом процедуры является блок с локальными параметрами *i*, *s*. Описание процедуры помещается в начале блока, как и все прочие описания, и при выполнении программы пропускается вместе с телом процедуры.

Для обращения к процедуре используется *оператор процедуры*, который имеет вид

<название процедуры> $(a_1, a_2, \ldots, a_n)$;

где $a_1, a_2, \ldots, a_n$ — *фактические параметры*. Например, для вычисления модуля *k*-мерного вектора с компонентами *p* в ячейку *q* можно написать

Modul (*p*, *k*, *q*);

Выполнение оператора процедуры происходит следующим образом:

1. Отыскивается описание процедуры, и формальные параметры приводятся в соответствие с фактическими параметрами из оператора процедуры.

2. В теле процедуры *Q* формальные параметры заменяются соответствующими фактическими. Получающийся после замены оператор $\tilde{Q}$ называют *модифицированным телом процедуры*.

3. Оператор $\tilde{Q}$ выполняется, как будто он стоит на месте оператора процедуры.

Характер фактических параметров должен быть согласован с характером формальных так, чтобы модифицированное тело процедуры не содержало неверных или бессмысленных операторов. Например, если тело процедуры содержит оператор $z := x+y$, то фактический параметр в операторе процедуры, который в модифицированном теле должен быть поставлен на место z, может быть лишь идентификатором переменной, а не арифметическим выражением, так как в противном случае оператор присваивания не будет иметь смысла.

Чтобы избежать несоответствий, в описание процедуры включается *спецификация формальных параметров*. Спецификация помещается между заголовком и телом процедуры и состоит в указании типа объектов, которые могут быть использованы в качестве фактических параметров, заменяющих формальные данные. Таким образом, спецификация формальных параметров напоминает описание переменных, тем более что для этого используются те же уже известные иероглифы:

integer — для спецификации формального параметра как целой арифметической переменной,

real — для действительной,

boolean — для булевской,

array — для массива,

procedure — для процедуры,

и только один новый символ

label *(метка)* — для спецификации формального параметра как метки. Однако спецификация отличается от описаний и внешне (например, в спецификации массива отсутствует указание границ) и по существу— спецификация лишь сообщает информацию и не вызывает никаких действий.

В зависимости от спецификации формальных параметров фактические параметры должны удовлетворять следующим условиям:

1. Если формальный параметр специфицирован как целый, действительный или булевский, то в качестве фактического параметра может быть использовано любое арифметическое, соответственно булевское, выражение. При этом исключается случай, когда такое выражение может появиться в левой части оператора присваивания.

2. Если формальный параметр специфицирован как массив, то в качестве фактического параметра может быть взят только идентификатор массива.

3. Если формальный параметр специфицирован как процедура или как метка, то роль фактического параметра может исполнять лишь идентификатор соответствующего класса.

Различные объекты программы могут оказаться обозначенными одними и теми же идентификаторами. Поэтому полезно привести

некоторые уточнения, для чего следует сначала дать точную классификацию объектов, встречающихся в теле процедуры. Разобьем объекты, встречающиеся в теле процедуры, на следующие классы:

1. *Локальные* объекты. Сюда относятся идентификаторы и метки, описанные в самом теле процедуры или помечающие операторы тела процедуры. Надо отметить, что тело процедуры всегда считается блоком, хотя бы оно и не содержало описаний.

2. *Формальные* объекты. Сюда относятся объекты тела, обозначенные идентификаторами формальных параметров процедуры, если только они не попали уже в число локальных.

3. Прочие объекты. Это объекты тела процедуры, не попавшие в две предыдущие группы. Их описание должно находиться в объемлющем процедуру вместе с ее описанием блоке, так что они являются глобальными.

При модификации тела процедуры выполняются следующие правила:

1. Формальные объекты тела заменяются фактическими параметрами. Объекты, вставленные на место формальных, понимаются в том смысле, который действует там, где расположен оператор процедуры (а не ее описание). Обозначение вставленного объекта может совпасть с обозначением локального объекта тела. Несмотря на это, их следует считать различными.

2. Прочие объекты понимаются в том смысле, который действует там, где расположено описание процедуры. После этих преобразований модифицированное тело исполняется так, как если бы оно находилось на месте оператора процедуры.

Случается, что к некоторой процедуре приходится на протяжении всего вычислительного процесса обращаться с одними и теми же фактическими параметрами, которые только изменяют свои числовые значения. В таких случаях преимущество формальных параметров теряет силу и удобно пользоваться *процедурой без параметров*. Описание такой процедуры имеет вид

procedure ⟨название⟩; Q;

где Q — тело процедуры. Оператор процедуры состоит при этом из одного лишь названия. Такая процедура не имеет фактических параметров, а ее описание не имеет формальных параметров. В теле процедуры Q действуют лишь локальные и прочие (глобальные) объекты. Для процедур без параметров остаются в силе все правила модификации тела и выполнения оператора процедуры, перечисленные выше.

Разновидностью оператора процедуры является *оператор функции*, который отличается от оператора процедуры видом описания и использованием в программе. В описании функции перед иероглифом **procedure** пишется указатель типа **integer**, **real** или **boolean**. Это уже является полным описанием переменной с идентификатором

функции. Иначе говоря, название процедуры-функции одновременно должно служить идентификатором значения функции, т. е. одной из выходных переменных, чаще всего единственной. Здесь имеется полная аналогия с фортрановскими подпрограммами типа FUNCTION, так же как и типа SUBROUTINE, что вовсе не требуется для произвольных процедур алгола, как это видно из приводимых примеров. В теле функции должно выполняться хотя бы одно присваивание какого-либо значения идентификатору функции.

Отличие в использовании состоит в том, что оператор процедуры представляет собой специальный отдельный оператор программы, т. е. обычная процедура требует отдельного обращения. Оператор функции может употребляться в любых арифметических или булевских выражениях. Упоминания функции в каком-либо выражении справа от символа присваивания достаточно для обращения программы к соответствующей процедуре. В этом смысле процедура-функция аналогична подпрограмме типа FUNCTION фортрана, тогда как обычная процедура аналогична подпрограмме фортрана типа SUBROUTINE.

Аналогично обычным процедурам, процедура-функция может быть как с формальными параметрами, так и без них. При пользовании процедурой-функцией без параметров необходимо ее аргументы всегда обозначать одними и теми же идентификаторами.

Наиболее употребительные и чаще всего используемые процедуры обычно описаны в трансляторе, так что нет нужды описывать их в программе. Такие процедуры аналогичны библиотечным подпрограммам фортрана. Отличие их от всех прочих процедур состоит лишь в том, что они имеют уже установленные названия, так же как и список формальных параметров. Как правило, процедуры, описанные в трансляторе, без параметров не употребляются. Важнейшими из таких процедур являются процедуры ввода и вывода, которым посвящен следующий параграф.

§ 9. ВВОД И ВЫВОД

Операции ввода и вывода, так же как и обмена информацией оперативной и внешней памяти, тесно связаны с особенностями устройства машины. Поэтому в таком языке, как алгол, который игнорирует особенности устройства конкретной машины, процедуры ввода и вывода разработаны недостаточно подробно. Правила алгола предусматривают существование двух процедур самого общего вида, конкретные способы работы с которыми определяются уже конкретными трансляторами.

Первая из них — read (x, y, z, . . ., w) — производит следующие действия: вводит с читающего устройства числовые значения и присваивает их переменным, стоящим в скобках. Переменные могут быть при этом простыми или с индексами (т. е. массивы) и относиться к типу целых или действительных. Вторая процедура —

print (*a*, *b*, *c*) —процедура вывода * . В результате ее работы на печатающем устройстве машины будут отпечатаны значения переменных, перечисленных в скобках. Разумеется, в конкретных случаях транслятор предусматривает ряд различных возможностей как для ввода, так и для вывода, но это уже относится к действиям конкретных трансляторов, а не общим правилам языка.

В противоположность этому правила фортрана имеют более разработанные процедуры ввода и вывода, входящие в состав языка, а не предоставленные авторам трансляторов на собственное усмотрение. С ними мы сейчас и познакомимся.

Инструкция ввода на фортране занимает две строки, расположенные независимо друг от друга. Исполняемая инструкция имеет вид

$$\text{READ}(M,\ N)\ \text{X},\ \text{Y},\ \text{Z}\ \ldots,\ \text{W},$$

где READ — фортранный иероглиф ввода; X, . . ., W — список вводимых (п р о с т ы х) переменных; M — номер устройства ввода, к которому программа должна обратиться, и N — метка второй, неисполняемой инструкции FORMAT, определяющей характер ввода. О ней речь будет идти ниже.

Несколько иначе выглядит инструкция READ при вводе массива. Ввод массива можно было бы осуществить при помощи цикла

```
DO 1 K=I, J
1 READ (M, N) X(K).
```

Взамен такого цикла обычно используется прием *автоматической индексации*, состоящий в замене приведенного цикла инструкцией вида

$$\text{READ}(M, N)\ (\text{X}\ (\text{K}),\ \text{K} = \text{I},\ \text{J}).$$

Более того, граница индекса тоже может быть прочитана и введена тут же, так что можно, например, писать

$$\text{READ}\ (M,\ N)\ L,\ (\text{X}\ (\text{I}),\ \text{I}=1,\ \text{L}).$$

Рассмотрим теперь неисполняемую инструкцию FORMAT, определяющую способ прочтения числа, набитого на внешнем носителе. Предположим сначала, что речь идет о перфокартах.

Как уже говорилось в § 3 предыдущей главы, информацию на перфокартах можно размещать построчно или поколонно. Элементом вводимой информации является отдельный с и м в о л — цифра, буква или знак (+, точка, запятая и т. п.).

Для изображения вводимого числа фортран разрешает отводить любое нужное число символов (в пределах ограничений, наложен-

---

* По-английски слово read означает *читай*, а print — *печатай*.

ных транслятором) и распределять их между целой и дробной частью любым требуемым образом. Кроме того, число можно набивать на карте в фиксированной или плавающей форме или с двойной точностью. Именно такого рода сведения о вводимых числах и содержатся в инструкции FORMAT.

Определенное количество символов, относящихся к одному числу, составляют *зону перфокарты*. Понятие зоны не зависит от способа расположения информации. Просто при поколонном расположении зона будет состоять из определенного количества колонок, а при построчном — из определенного количества восьмиразрядных двоичных чисел, занимающих одну или несколько строк, полных или неполных. Если вводимое число X имеет шесть цифр, из которых четыре отведены для записи дробной части, то для его ввода можно воспользоваться командами

```
      READ     (1,5) X
5 FORMAT  (F 6.4)
```

Как уже говорилось, 1 в команде READ означает, что число вводится с устройства, имеющего номер 1. Буква F в инструкции FORMAT указывает, что число задается в фиксированной форме.

Заметим, что инструкция FORMAT вовсе не обязательно следует за инструкцией READ. Она может быть помещена и много позже или, наоборот, раньше. Важно только, чтобы она обязательно имела соответствующую метку (номер), по которой и будет найдена транслятором, где бы она не находилась.

Если на перфокарте набито несколько чисел, то содержимое скобок в инструкции FORMAT должно полностью описывать всю карту. Так, инструкция

```
4 FORMAT (F 10.3, F 8.3, F 8.1, F 10.5)
```

показывает, что на карте набито 4 числа в фиксированной форме, занимающие соответственно 10, 8, 8 и 10 символов, в которых для дробных частей отведены соответственно 3, 3, 1 и 5 разрядов. Если все или несколько зон одинаковы, то перед буквой можно ставить (непременно целый!) коэффициент. Например, инструкция

```
6 FORMAT (4 F 10.3)
```

равносильна инструкции

```
6 FORMAT (F 10.3, F 10.3, F 10.3, F 10.3).
```

Можно вводить также числа с плавающей запятой, т. е. в форме *E*. Для этой цели в инструкции FORMAT ставится буква *E*, затем снова размер зоны и количество знаков дробной части. Мантисса числа в плавающей форме не обязательно нормализована, и указа-

ние количества знаков дробной части показывает место десятичной точки в мантиссе в том случае, если она не отперфорирована на карте. Если же десятичная точка перфорирована, то указание числа знаков дробной части во внимание не принимается. Необходимо только помнить, что десятичная точка входит в число символов, так же как и символ *E*, входящий в изображение числа в плавающей форме, как, кстати говоря, и знак числа, если он есть, не только в плавающей, но и в фиксированной форме.

До сих пор речь шла о вводе с перфокарт. Собственно говоря, при вводе с перфоленты инструкция FORMAT используется таким же точно образом. Так же устроены и инструкции вывода, также состоящие из двух инструкций — исполняемой и неисполняемой. Исполняемая инструкция состоит из иероглифа WRITE *(писать)* и имеет вид

WRITE (*M*, *N*) A,

вполне аналогичный инструкции READ. Здесь *M* — номер устройства вывода; *N* — метка соответствующей инструкции FORMAT и A — печатаемая переменная. Инструкция FORMAT определяет количество печатаемых символов и форму числа точно таким же образом, как и для ввода.

Глава VII

СЕГОДНЯ И ЗАВТРА

§ 1. НА ПОРОГЕ НОВЫХ ПОКОЛЕНИЙ

Машины второго поколения, которым были посвящены предыдущие главы, полностью «оформились» к середине шестидесятых годов. По сравнению с первыми образцами автоматических цифровых вычислительных машин они обладали громадным быстродействием, колоссальной памятью, хорошо развитой системой команд и удобными языками программирования. Вместе с тем примерно в это же время выяснилось, что эти машины не удовлетворяют целому ряду новых желаний и требований (аппетит приходит во время еды!), что вызвало смену поколений.

Широкое проникновение математики и математических методов в самые разнообразные области техники, и не только естественных, но и гуманитарных наук, обусловленное, в частности, возможностями уже существующих вычислительных машин, поставило перед вычислительной техникой новые задачи, для решения которых скоростей и объема памяти имеющихся машин оказалось недостаточно. Но и при тех же возможностях нужно было работать над сокращением габаритов, уменьшением стоимости и повышением надежности машин. Вследствие большого их распространения надо было облегчить взаимодействие человека с машиной и расширить возможности этого взаимодействия.

Целый ряд проблем носит «чисто внутренний» характер. Повышение эффективности использования вычислительных машин требовало дальнейшей автоматизации их работы. В машинах первого, да и второго поколения большую роль играла работа оператора за пультом управления, отнимавшая относительно много времени. Для машин первого поколения, имевших скорость порядка тысяч операций в секунду, потери нескольких секунд по сравнению с часами работы особой роли не играли. Но когда скорость выросла в сотни раз, то потеря оператором секунды стала означать потерю нескольких сот тысяч операций. Другой «канал потерь», с которым следовало бороться — взаимодействие основных устройств машины с внешними, периферийными устройствами. По сравнению со скоростью работы арифметического устройства и оперативной памяти

внешняя память и устройства ввода-вывода машин второго поколения работали недопустимо медленно. Хотелось бы организовать работу машины так, чтобы во время работы устройства печати, выдающего результаты одной задачи, арифметическое устройство и оперативная память вели бы уже вычисления по программе следующей.

Использование вычислительной машины для целей управления требовало присоединения к машине каналов связи, по которым в машину будет поступать информация о ходе управляемого процесса. Необходимо было организовать прием поступающей информации таким образом, чтобы не повредить работе машины в момент поступления этой информации и вместе с тем своевременно использовать полученные данные для очередных расчетов. Таким образом, здесь речь шла о возможности прерывания работы машины, разделения времени и снова, как и в предыдущем абзаце, о параллельной работе нескольких устройств.

Требовали заметного улучшения и устройства ввода и вывода. Перфорация данных на исходном носителе весьма трудоемка и занимает много времени. Кроме того, даже самый быстрый фотоввод воспринимает перфорированные данные со скоростью нескольких миллисекунд на машинное слово, тогда как для записи в оперативную память на машинное слово расходуется несколько микросекунд, т. е. в тысячу раз меньше. Выходит, что при работе читающего устройства память занята лишь 0,1% времени, а остальные 99,9% простаивает. Сейчас конструируются устройства подготовки данных без перфорации, например, позволяющие использовать в качестве первичного носителя информации магнитную ленту. Делаются также попытки сконструировать читающее устройство, позволяющее читать и вводить в память печатный текст, хотя бы напечатанный стандартным образом.

Аналогичные пожелания имеются и к устройствам вывода. Алфавитно-цифровая печать дает возможность удобно представлять таблицы. Но часто бывает нагляднее представить результаты в виде графика; для этого можно, конечно, использовать и печатающее устройство, но лучше иметь в числе устройств вывода также графопостроитель, специально предназначенный для построения графиков. В других случаях бывает удобно выводить полученные результаты на экран электронно-лучевой трубки, аналогично выводу на аналоговых вычислительных машинах. Такое устройство вывода получило название *дисплей* (от английского *display* — показывать, выставлять). В некоторых случаях аналогичное устройство можно использовать и для ввода данных в машину.

Все сказанное привело к совершенствованию способов организации и структуры устройств и изменению элементно-технологической базы. Переход от первого поколения вычислительных машин ко второму основывался, собственно, только на втором из этих направлений и состоял, главным образом, в переходе от электронных ламп к полупроводниковым приборам; схемы организации взаи-

модействия устройств подверглись не слишком серьезным изменениям. Переход к следующим поколениям, во-первых, углубил требования к улучшению организации, а во-вторых, предоставил для такого улучшения больше возможностей.

Элементарным примером улучшения организации взаимодействия отдельных устройств можно считать следующий. Время выполнения операций в арифметическом устройстве заметно меньше, чем время выборки команд и исходных чисел из памяти. Поэтому арифметическое устройство некоторое время «простаивает» в ожидании «материала для переработки». Один из путей устранения таких «простоев» состоит в создании небольшой *сверхоперативной памяти*, обладающей особо большим быстродействием, для использования ее в качестве рабочих ячеек, предназначенных для хранения промежуточных результатов вычислений. Такое устройство значительно ускорит выполнение большинства операций и в заметной степени снизит простои.

Настоящая глава посвящена беглому знакомству с тем новым, что внесли в вычислительную технику машины следующих поколений,— новыми элементами, новыми физическими принципами, новшествами в вопросах организации и эксплуатации машин, изменениями в программировании.

§ 2. ИНТЕГРАЛЬНЫЕ СХЕМЫ

Быстродействие цифровых вычислительных машин определяется быстродействием логических элементов и составленных из них операционных схем. Поэтому повышение скорости работы машин в заметной степени упирается в новые принципы подхода к физической реализации полупроводниковых схем.

Легко заметить, что быстродействие полупроводникового логического элемента определяется, прежде всего, физическими размерами транзистора, точнее, размерами области, занятой пространственным зарядом на границах зон различной проводимости. Транзистор представляет собой плоский квадратик со стороной менее 1 *мм* и толщиной 0,2 *мм*, что позволяет получить время переключения порядка 0,1 *мксек*, т. е. 10^7 переключений в секунду; дальнейшее уменьшение размеров кристалла затрудняет работу с ним (к кристаллу необходимо припаять три вывода — эмиттер, коллектор и базу, и поместить его в герметичный металлический корпус, защищающий его от загрязнения и механических повреждений, а также выполняющий роль теплоотвода).

Таким образом, первая важнейшая задача, стоящая перед технологами, состоит в уменьшении геометрических размеров полупроводниковых элементов. Но есть и еще одна, не менее важная задача: уменьшить расстояние между компонентами схем.

Электрические сигналы распространяются по проводнику со скоростью, сравнимой со скоростью света (несколько меньшей ее), которая, как известно, составляет $3 \cdot 10^{10}$ *см/сек*. Представим себе, что нам удалось создать транзистор со скоростью переключения 1 наносекунду, т. е. 10^{-9}*сек*. За это время сигнал пройдет $3 \cdot 10^{10}$ *см/сек* $\times$ $\times 10^{-9}$*сек* $= 30$ *см*. Если расстояние между элементами схемы превы-

шает половину метра, что вполне возможно, то сигнал потратит на такое расстояние около 2 наносекунд и мы потеряем весь выигрыш, который удалось получить за счет высокой скорости переключения — она будет «съедена» временем передачи сигнала. Следовательно, компоненты схем следует размещать на максимально возможно близком расстоянии друг от друга.

Решение этих задач на традиционных элементах второго поколения практически невозможно. Транзисторы и другие компоненты схем имеют сравнительно большие размеры. Значительное повышение плотности компоновки схем увеличивает рассеивание мощности в электрической цепи, что приводит к нарушению теплового режима. Увеличение числа соединительных проводов затрудняет монтаж схем. В конечном счете все эти обстоятельства приводят к ухудшению качества работы элементов и резко снижают надежность работы машины. Так как повышение мощности вычислительных машин связано с увеличением количества элементов, то для сохранения надежности машины в целом на том же уровне необходимо повысить надежность отдельных элементов и схем, что составляет еще одну задачу, подлежащую решению.

Перечисленные задачи удалось решить благодаря введению и широкому распространению *интегральной технологии*, послужившей основой для создания машин следующих поколений. Эта технология позволяет получить на одной небольшой пластинке полный логический элемент или даже целую операционную схему, причем все соединения между элементами схемы будут являться неразрывными частями той же пластинки.

Полупроводниковые элементы получают добавлением в кристалл четырехвалентного германия или кремния трехвалентных или пятивалентных примесей. При этом размеры областей с примесями могут быть значительно меньше, чем указано выше, если только не нужно изготовлять из них конструктивно отдельные элементы. Поэтому естественно возникает мысль «организовать» на одной германиевой или кремниевой пластинке несколько транзисторов, тем более что чистый материал пластинки является отличным изолятором. Вместе с тем на той же пластинке можно создать и другие нужные элементы схемы — резисторы, конденсаторы, а также и соединения между ними (рис. 86).

Интегральную технологию изготовления схемы можно представить себе следующим образом. Берется квадратная пластинка германия или кремния размером в несколько (3÷5) миллиметров и толщиной 0,2÷0,3 *мм*. Поверхность пластинки закрывают маской с несколькими отверстиями, через которые вводятся примеси одного типа, например пятивалентные. Затем заменяют маску и через новые отверстия вводят примеси другого типа, создавая, таким образом, *n*- и *p*-области транзисторов. Через новую маску напылением в вакууме наносят участки золота или серебра, которые представляют выводы электродов транзистора. К этим выводам можно присоединить также путем распыления резистор или конденсатор.

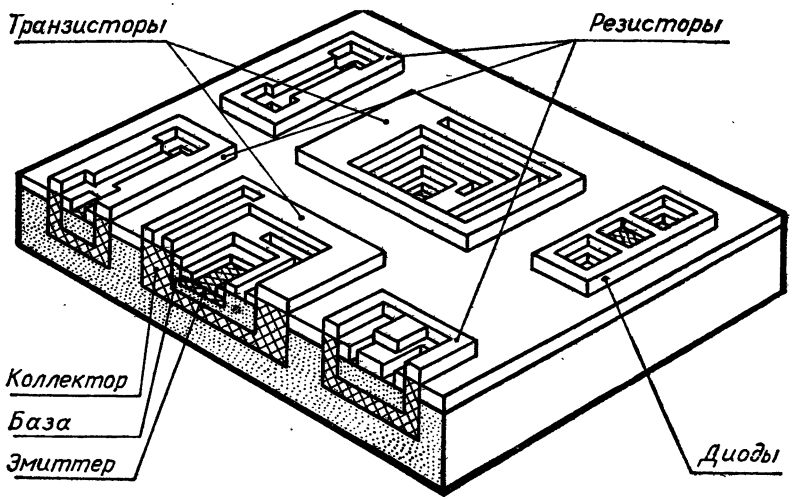


Рис. 86

В качестве резистора можно использовать тонкопленочную металлическую полоску шириной в несколько сотых миллиметров и длиной, обеспечивающей нужное сопротивление. Если требуются большие сопротивления, то полоску можно положить зигзагообразно или заменить угольным порошком. Чтобы сделать конденсатор, можно положить тонкую пленку диэлектрика, например окиси кремния, между двумя металлическими пленками. Величина емкости такого конденсатора регулируется площадью поверхности и толщиной пленки диэлектрика. Тонкие пленки металла или диэлектрика осаждаются испарением или, в случае тугоплавкого вещества, катодным напылением.

Заряженные положительно ионы газа используются для обстрела отрицательно заряженного катода, состоящего из тугоплавкого материала, который должен быть осажден в виде тонкой пленки. В результате обстрела этими «снарядами» катода его материал испаряется и подложка покрывается атомами, вырванными из катода. Распыление позволяет получить нужные пленки в местах, не покрытых маской, в течение нескольких минут. Изготовленные таким образом схемы помещают в металлический корпус, который может иметь примерно те же размеры, что и корпус транзистора. В результате интегральная технология позволяет достичь плотности размещения до 100 транзисторов вместе с подсоединенными пассивными элементами на 1 квадратный сантиметр.

Надежность такой интегральной схемы примерно равна надежности одного транзистора, изготовленного по прежней технологии. То же самое относится и к стоимости. Вместе с тем размеры активных элементов значительно уменьшаются, что способствует сокращению времени переключения схемы, т. е. повышению быстродей-

ствия. Ко всем перечисленным достоинствам интегральных схем следует добавить удобство монтажа и скромность в рассеивании мощностей, а значит, благоприятный тепловой режим.

Описанные интегральные схемы могут состоять из весьма различного числа элементов и в зависимости от этого иметь различное предназначение. Общепринятые градации интегральных схем примерно таковы:

малые интегральные схемы содержат до 10 логических элементов;

средние интегральные схемы содержат от 10 до 100 логических элементов, образующих отдельные операционные схемы — регистры, счетчики и т.п.;

большие интегральные схемы содержат порядка 1000 логических элементов, используемых уже в роли отдельных устройств цифровых вычислительных машин, например арифметического устройства.

Активными элементами описанных интегральных схем являются обычные *p—n—p-* или *n—p—n*-транзисторы. Их называют также *биполярными* транзисторами, так как в них используются носители зарядов обоих знаков: электроны и дырки. В последние годы все большее применение в интегральных схемах находят так называемые *униполярные транзисторы*, в которых используются носители только одного знака. Это приборы со структурой *металл — окисел—полупроводник*, и поэтому их называют МОП-транзисторами; такое сокращение является общепринятым. Именно МОП-транзисторы являются основой упомянутых выше больших интегральных схем, которые, в свою очередь, служат элементно-технологической базой машин четвертого поколения в отличие от машин третьего поколения, использующих преимущественно малые и частично средние интегральные схемы.

Элементами МОП-транзистора являются *исток*, *сток* и *затвор* — это суть аналоги *катода*, *анода*, *сетки* — вакуумного триода и *эмиттер*, *коллектор*, *база* (выполняющие те же функции) — биполярного транзистора. МОП- транзистор образован из пластинки *p*-кремния, т. е. чистого кремния с *p*-примесью, в которой методом диффузии образована *n*-область с избытком электронов, называемая *каналом*. К двум концам канала подведены электроды *исток* и *сток*. Центральная часть канала очень тонкая и над нею расположен еще один электрод — *затвор*, который изолирован от канала тонкой пленкой окиси кремния SiO_2 (рис. 87).

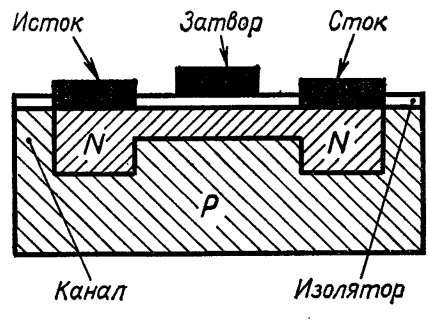


Рис. 87

Сток соединяется с положительной клеммой источника пита-

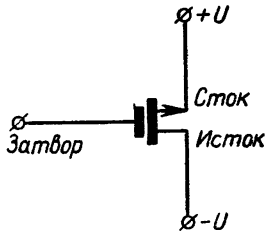


Рис. 88

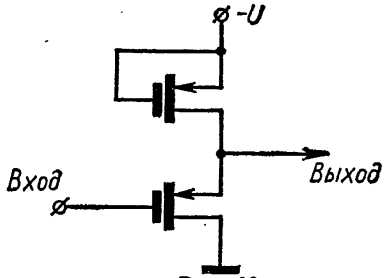


Рис. 89

ния, исток — с отрицательной. Если приложить к затвору положительный потенциал, то возникшее поле откроет канал для прохождения электронов от истока к стоку и далее через сопротивление нагрузки. Если же к затвору приложен отрицательный потенциал, то канал будет заперт. Таким образом, в этом приборе «работают» носители заряда одного знака — электроны, и его называют МОП-транзистором с *n*-каналом. Его условное изображение на схеме показано на рисунке 88. Аналогично можно создать МОП-транзистор с *p*-каналом, в котором носителем будут дырки. При его изображении на схеме используется обратное направление стрелки стока.

Ценным качеством МОП-транзисторов является их высокое входное сопротивление, которое объясняется тем, что у него входная цепь изолирована от выходной. Поэтому МОП-транзистор более близок по своим характеристикам к вакуумным электронным лампам, нежели к биполярным транзисторам. Благодаря высокому входному напряжению МОП-транзистор с фиксированным смещением можно использовать вместо сопротивления нагрузки, что позволяет сократить количество элементов схемы. В качестве примера укажем схему инвертора, приведенную на рисунке. 89. Замена резистора МОП-транзистором устраняет необходимость изолировать друг от друга элементы схемы, что позволяет на одной и той же площади располагать большее количество элементов и более сложные схемы.

К числу достоинств МОП-транзисторов следует отнести то, что они занимают примерно в десять раз меньшую площадь, чем биполярные, и значительно более технологичны, что особенно важно для интегральных схем. Так, при стандартной технологии изготовление биполярного транзистора требует 130 технологических операций, тогда как для изготовления МОП-транзистора достаточно всего 38. Кроме того, МОП-транзисторы потребляют мощность в 10—20 раз меньшую, чем биполярные.

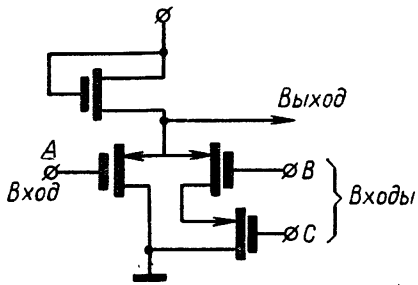


Рис. 90

К сожалению, МОП-транзисторы не сотканы из одних достоинств; можно указать и их недостатки. Как высокоомный прибор, управляемый напряжением, МОП-транзистор более «чувствителен» к «паразитным емкостям», которые влияют на быстродействие. Поэтому схемы на МОП-транзисторах обладают несколько меньшим быстродействием, нежели схемы с биполярными транзисторами.

Пример логической схемы на МОП-транзисторах, реализующей логическую функцию $y=A \vee B \wedge C$, приведен на рисунке 90. Читатель сумеет самостоятельно разобраться в работе этой схемы.

§ 3. ТОНКИЕ ПЛЕНКИ. ОПЕРАТИВНАЯ И ПОСТОЯННАЯ ПАМЯТЬ

Оперативная память на ферритовых сердечниках продолжает оставаться вне конкуренции и в машинах третьего и четвертого поколений. Тем не менее продолжающиеся попытки увеличить быстродействие и уменьшить размеры памяти, а также уменьшить потребление энергии привели к ряду многообещающих новых решений.

Быстродействие ферритового сердечника определяется, в частности, его размерами. Как показали исследования, ферромагнитный материал будет сохранять свои свойства до толщины порядка нескольких сот ангстрем *. Ясно, что сердечники такого размера изготовить невозможно. Но можно поступить иначе — нанести тонкий слой ферромагнитного материала на немагнитную, например медную, проволоку; тогда ее можно будет рассматривать как медный проводник, на который нанизано множество кольцевых сердечников, толщина которых равна толщине слоя нанесенного ферромагнитного материала. Подобные идеи привели к созданию *запоминающего устройства на тонких магнитных пленках*; есть все основания полагать, что они являются серьезным конкурентом для памяти на ферритовых сердечниках.

Магнитные пленки, применяемые в запоминающих устройствах, делятся на плоские и цилиндрические. Плоские пленки в виде круглых или прямоугольных пятен толщиной от 200 до 10 000 Å наносятся на плоскую полированную подложку. Время перемагничивания элемента колеблется от единиц до десятков микросекунд **. О цилиндрических пленках мы уже упоминали.

Замечательным свойством тонких магнитных пленок является их *анизотропия*, т. е. изменение свойств (в данном случае магнитных) в зависимости от направления намагничивания. У пленки имеются два взаимно перпендикулярных направления намагничивания, которые называются направлениями *легкого* и *трудного намагничивания* (рис. 91). Если пленка перемагничивается полем, направление которого совпадает с направлением легкого намагни-

---

* 1Å=0,0001 *мкм*=10^{-10} *м*.

** Такая скорость объясняется не только размером элемента, но и несколько иным по сравнению с ферритовым сердечником физическим механизмом перемагничивания, на чем мы не останавливаемся.

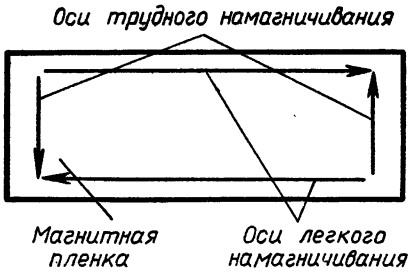


Рис. 91

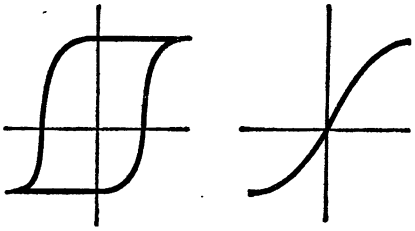


Рис. 92

чивания, то петля гистерезиса имеет прямоугольную форму с высоким коэффициентом прямоугольности. Если же направление поля совпадает с направлением трудного намагничивания, то петля гистерезиса либо сильно отличается от прямоугольной, либо даже отсутствует (рис. 92).

Анизотропия магнитной пленки позволяет использовать ее для запоминания информации. В направлении легкого намагничивания элемент пленки эквивалентен ферритовому сердечнику и может «запоминать» одно из состояний петли гистерезиса. Поэтому одному из направлений намагниченности вдоль этой оси можно поставить в соответствие код «1», а противоположному «0». Направление оси трудного намагничивания используется для считывания информации, как о том будет сказано ниже.

На рисунке 93 изображен тонкопленочный прямоугольный элемент, на который наложены две шины: числовая, направленная по оси легкого намагничивания, и шина чтения, направленная перпендикулярно числовой, т. е. вдоль оси трудного намагничивания. Предположим, что вектор намагниченности направлен по направлению легкого намагничивания влево и что это состояние соответствует коду «1». Импульс тока через числовую шину порождает маг-

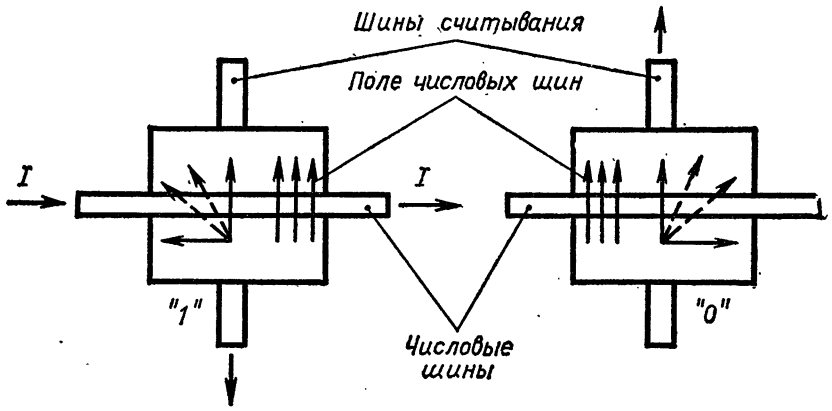


Рис. 93

нитное поле, направленное вдоль «трудной» оси, которое поворачивает вектор намагниченности элемента пленки по часовой стрелке, чтобы ориентировать внутреннее поле параллельно созданному импульсом тока. В шине считывания возникает при этом импульс тока (рис. 93, слева). Если элемент пленки находился в состоянии «0», то вектор намагниченности повернется против часовой стрелки и импульс тока в шине считывания будет иметь противоположную полярность (рис. 93, справа). Таким образом, кодам «1» и «0» соответствуют здесь импульсы разной полярности, что увеличивает надежность работы запоминающего устройства.

Вектор намагниченности будет направлен по оси трудного намагничивания лишь до тех пор, пока не прекратится действие поля, порожденного импульсом в числовой шине, после чего он возвратится в одно из устойчивых состояний. Необходимо принять специальные меры к тому, чтобы он возвратился именно в то состояние, в котором он находился до импульса. Для этого параллельно шине считывания накладывают *разрядную шину записи* (рис. 94).

Чтобы обеспечить возврат вектора намагниченности после снятия внешнего поля, вызванного импульсом тока, в прежнее положение, в разрядную шину, не прерывая действия числового тока, подают импульс тока меньшей амплитуды и такой полярности, чтобы создаваемое им поле было направлено в ту же сторону, что и первоначальный вектор, скажем вправо, если был записан «0», и влево, если была записана «1». На вектор намагниченности будет, следовательно, действовать результирующее поле, равное векторной сумме двух полей. Под действием результирующего поля вектор намагниченности окажется направленным не точно по оси трудной намагниченности, а несколько повернутым в нужную сторону (рис. 94), благодаря чему после прекращения тока и снятия внешнего поля вектор намагниченности возвратится в прежнее положение. Таким образом, произойдет чтение информации без ее разрушения.

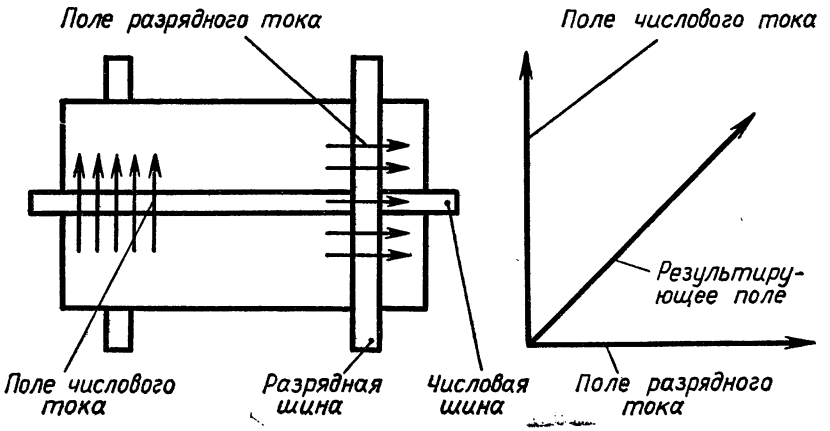


Рис. 94

Конструктивно пленочная память выполняется в виде подложки, на которую через маску напыляется тонкая ферромагнитная пленка, в результате чего получается *плата*, на которой расположены ряды отдельных элементов (пятен) различной формы. Затем наносятся числовые, разрядные и считывающие шины, представляющие собой напыленные через маску полоски золота.

Еще один вопрос, который мы упомянем в настоящем параграфе, относится к так называемым *постоянным запоминающим устройствам*. Под *постоянным запоминающим устройством*, или *постоянной памятью*, мы будем понимать устройства, предназначенные для хранения неизменяемой (или п о ч т и неизменяемой) информации, которые, следовательно, практически работают лишь на считывание. Характерным признаком такой памяти является то, что хранящаяся в ней информация не теряется при выключении питающих устройств.

Постоянная память отнюдь не является принадлежностью только машин последних поколений. Скорее, наоборот, она имеет свои истоки еще в коммутационных досках табуляторов и счетно-аналитических машин. К постоянной памяти можно отнести и «паяные» программы на релейных вычислительных машинах и электронных машинах первого поколения. Тем не менее постоянная память машин последних поколений заслуживает отдельного разговора; дело в том, что на этих машинах для устройств постоянной памяти используются все современные физические элементы и схемы, используемые также и для оперативной памяти. Это дает новые возможности.

Большое значение имеет постоянная память для хранения библиотеки стандартных программ, трансляторов с алгоритмических языков, программы-диспетчера и других частей *операционной системы*, о которой мы будем говорить подробнее в последних параграфах настоящей главы. При таком ее использовании оказываются весьма существенными возможность доступа к многим различным ячейкам постоянной памяти, достаточно большое быстродействие и надежность запоминания информации, что характерно для постоянной памяти машин последних поколений. Дальнейшее развитие этих идей приводит к созданию специализированных устройств, предназначенных для аппаратной реализации целого ряда «обязанностей» вычислительных машин, ранее выполнявшихся программным путем. В качестве примера можно упомянуть хотя бы перевод вводимой с внешнего носителя (например, с перфоленты) информации во внутренний код машины.

§ 4. НОВОЕ В ЗАПОМИНАЮЩИХ УСТРОЙСТВАХ

В § 1 мы уже касались вопроса о том, что скорость работы арифметического устройства много выше, чем памяти, и поэтому для обеспечения работы арифметического устройства без ожидания и простоев необходима *сверхоперативная память* с особо большим быстродействием при не слишком большом объеме.

Физической основой таких устройств могут быть магнитопленочные элементы и полупроводниковые триггера. О пленках мы уже говорили выше; сверхоперативная память на тонких пленках в принципе не отличается от оперативной. Запоминающее устройство на триггерных схемах с обычными транзисторами, хотя и могло обладать требуемым быстродействием, но было бы очень дорогим, занимало много места и потребляло бы много энергии. Совсем иное дело интегральные схемы.

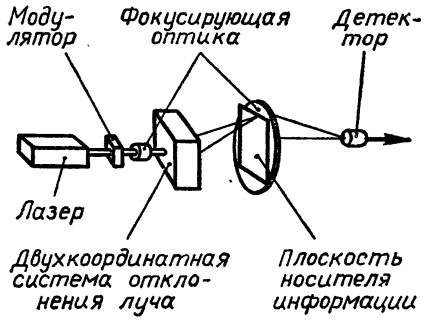


Рис. 95

Остановимся на некоторых экспериментальных работах по созданию новых типов запоминающих устройств, основанных на других принципах физической реализации. Эти устройства не используются еще в серийно выпускаемых электронных вычислительных машинах, но любое из них может найти широкое применение в вычислительных машинах следующих поколений.

Наиболее перспективными, по-видимому, являются *устройства памяти с использованием оптических явлений*, которые обладают не только достаточно высоким быстродействием, но и очень большой плотностью записи информации, гораздо большей, чем для систем магнитной записи. В качестве источника света в оптических запоминающих устройствах обычно применяется лазер, дающий излучение высокой интенсивности.

Лазерные оптические запоминающие устройства делятся на два типа: запоминающие устройства *со сканированием* и *без сканирования*. Типичная схема лазерной оптической памяти со сканированием изображена на рисунке 95. Модулятором светового потока является быстродействующий оптический переключатель. Двухкоординатная система отклонения луча направляет модулированный луч на носитель информации. Записанная на носителе информация меняет характер луча — например его интенсивность или фазу. Далее луч фокусируется на детектор, который преобразует световой поток в электрические сигналы, различные состояния которых интерпретируются как цифры двоичной системы.

Запись информации основана на прожигании с помощью более мощного лазера, чем при считывании, отверстий в непрозрачном покрытии прозрачной пленки, которая сама при этом не разрушается. Скорость записи в такой системе составляет несколько миллионов двоичных единиц информации в секунду при плотности более 15 тысяч двоичных единиц на квадратный миллиметр. Такая пленка, очевидно, служит постоянной памятью, поскольку записанная на ней однажды информация может только считываться и не может быть изменена.

В устройстве памяти без сканирования неподвижный луч лазера пересекается движущейся фотографической пленкой. По внешнему виду и во многих отношениях по функционированию оптическая память без сканирования подобна памяти на магнитной ленте.

Электролюминесцентные или *оптоэлектронные* запоминающие устройства отличаются от оптических тем, что в них источником света являются сами запоминающие элементы, например светодиоды. Их можно располагать в виде матриц по типу ферритовых сердечников. Для выбора элемента через координатные шины пропускается половинный ток, который в точке пересечения шин вызывает вспышку светодиода или участка люминофора. Свет попадает на расположенную между подложкой и светоприемником маску с записанной на ней информацией. Такие устройства имеют небольшое быстродействие из-за длительности времени высвечивания люминофора.

К оптическим запоминающим устройствам относятся также устройства, использующие методы *голографии*. Прежде чем знакомиться с голографическими запоминающими устройствами, которые отличаются необычайно большой плотностью записи информации и могут быть использованы для хранения очень больших мас-

сивов информации, нам следует хотя бы поверхностно познакомиться с физической сущностью явления голографии.

Голография основана на интерференции света. Как хорошо известно, явление интерференции состоит во взаимном усилении или ослаблении двух волн в зависимости от соотношения фаз. Луч света представляет собой электромагнитное колебание, электрическую составляющую которого можно в простейшем случае изобразить в виде обычной синусоиды. Если в некоторой точке пространства соответствующие двух лучей совпадают по величине и по фазе (т. е. одинаково направлены), то амплитуда их суммы удвоится (рис. 96, *а*). Если же лучи точно противоположны по направлению — находятся в противофазе, т. е. отличаются точно на половину периода, то их сумма будет тождественно равна нулю

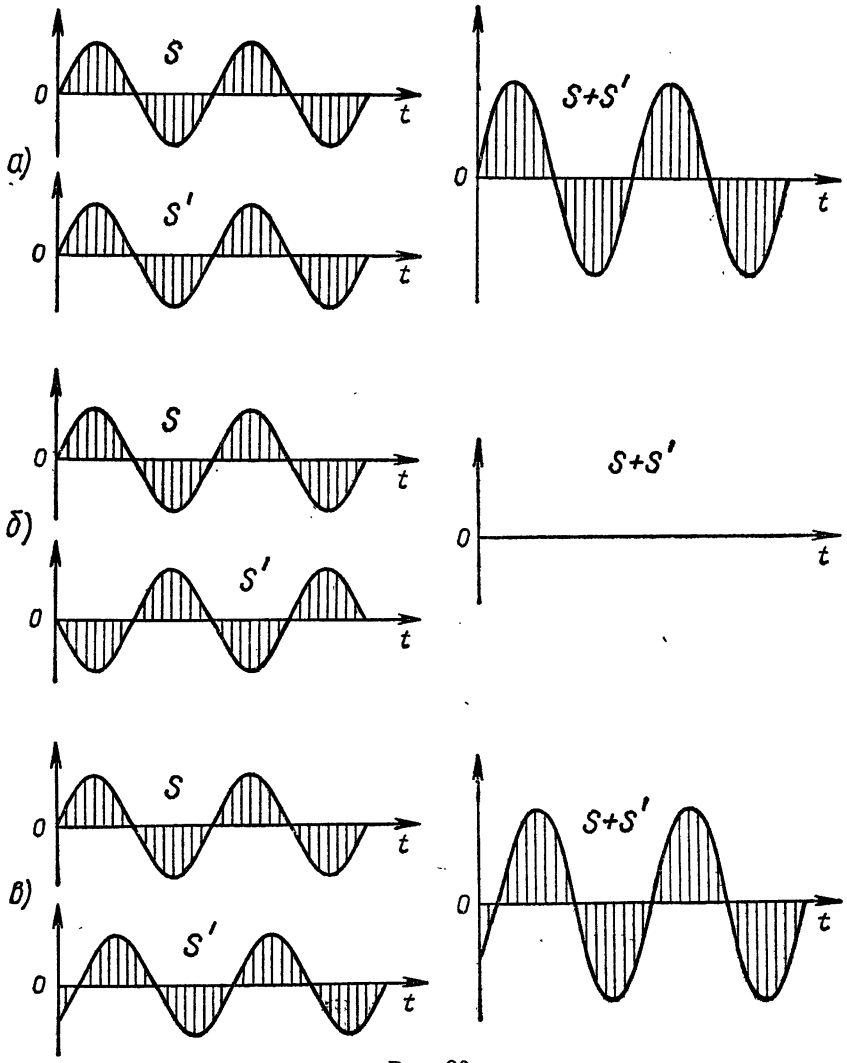


Рис. 96

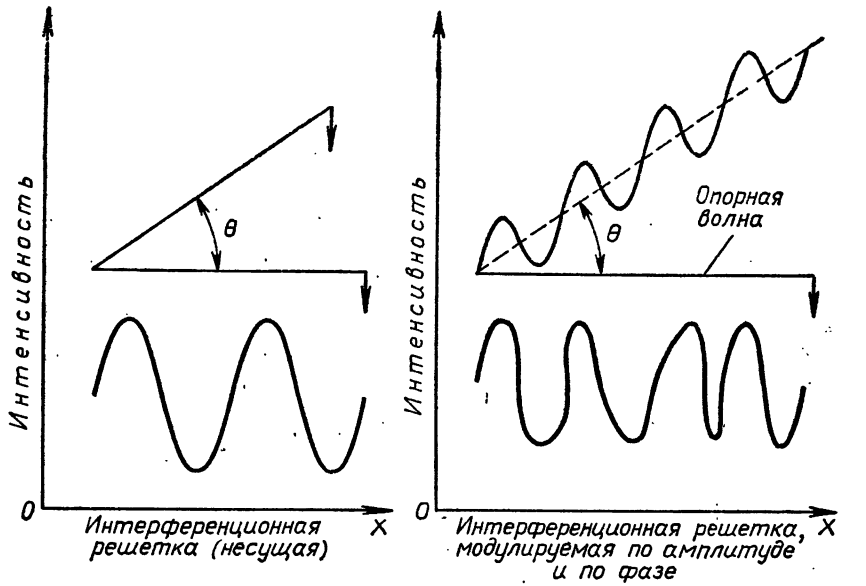


Рис. 97

(рис. 96, *б*). В общем случае при сложении лучей с разной длиной волны и меняющейся разностью фаз картина будет гораздо сложнее. На рисунке 96,*в* изображен более простой случай: сложение двух лучей с одинаковой длиной волны и постоянной в каждой точке разностью фаз.

Источник света, испускающий свет строго фиксированной длины волны, называется *монохроматическим*. Если разность фаз двух лучей из пучка постоянна, то пучок называют *когерентным*. При использовании монохроматического когерентного света от лазера получается интерференционная картина, представляющая собой чередование светлых и темных полос с высокой контрастностью. Такую картину называют *интерференционной решеткой*. На рисунке 97 изображена интенсивность интерференционной решетки при интерференции двух плоских волновых фронтов.

Если один из волновых фронтов направить на объект или пропустить через маску, на которой записана информация, то волновой фронт первоначальной плоской волны исказится. Этот волновой фронт интерферирует со вторым, заранее известной формы — в простейшем случае неискаженным плоским волновым фронтом, который называется *опорным* и составляет с искаженной (рассеянной) волной некоторый угол. Распределение яркостей в образующейся интерференционной картине определяется как амплитудным, так и фазовым распределением поля в рассеянной волне. Полосы полученной интерференционной решетки промодулированы как по положению, так и по интенсивности (рис. 97) и содержат информацию об объекте. Эта картина записывается на носитель, чувствительный к электрической составляющей электромагнитного поля, например на фотопластинку; такая запись и называется *голограммой*.

Для восстановления записанной на голограмме информации нужно осветить голограмму плоской когерентной волной света. В результате дифракции света на решетке образуется система плоских волн, которые и представляют восстановленные исходные волны: фазовая и амплитудная модуляции системы дифрагированных волн совпадают с аналогичными модуляциями волны, рассеянной объектом, и создают изображение объекта. Типичная схема для записи голограммы маски приведена на рисунке 98. Особым достоинством голографической записи является

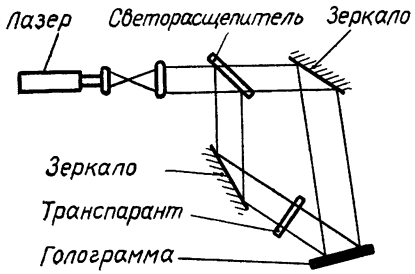


Рис. 98

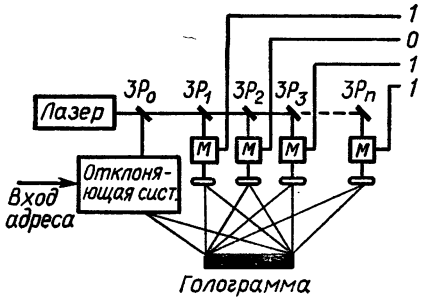


Рис. 99

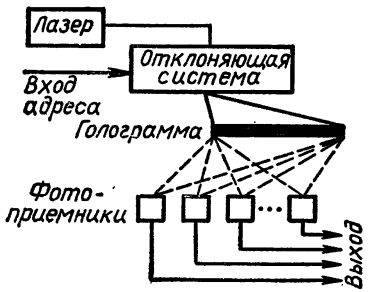


Рис. 100

возможность последовательной записи на одну голограмму нескольких изображений при помощи изменения угла падения опорного луча. В процессе восстановления эти изображения легко могут быть разделены.

Блок-схема записи в голографическое запоминающее устройство с адресной системой и записью информации в тонком слое носителя показана на рисунке 99. Луч лазера расщепляется делителем на опорный и сигнальный. Опорный луч проходит через отклоняющую систему, которая в соответствии с введенным адресом устанавливает направление опорной волны. Сигнальный луч расщепляется на каналы в соответствии с числом разрядов записываемого слова. В каждом из каналов имеется модулятор, который пропускает свет от лазера, когда к модулятору приложено управляющее напряжение, и остается непрозрачным, когда напряжение отсутствует. Каждый разряд записываемого слова подводится к своему модулятору. На выходе возникает комбинация лучей, которая вместе со своим опорным лучом записывается на голограмму.

Считывание информации с голограммы выполняется в устройстве, блок-схема которого приводится на рисунке 100. В соответствии с заданным адресом отклоняющее устройство устанавливает угол падения считывающего пучка на голограмму. Восстановленное изображение двоичного слова в виде световых пятен проектируется на линейку фотоприемников так, чтобы каждое пятно попало в свой фотоприемник. На выходе фотоприемников появляется комбинация электрических сигналов, соответствующая записанному в данной ячейке двоичному слову.

§ 5. МИКРОПРОГРАММИРОВАНИЕ

Идеи микропрограммирования возникли еще при работах по модернизации вычислительных машин первого поколения. Речь идет о том, чтобы не фиксировать жестко набор элементарных операций машины,

предоставив программисту возможность самостоятельно «составлять» по крайней мере некоторые из них путем комбинации микроопераций. Для этого необходимо, чтобы программист имел доступ не только к элементарным операциям, но и к микрооперациям и мог программно обеспечить выполнение нужной последовательности микроопераций. Естественно, что такая работа предполагала непосредственное программирование в кодах и требовала от программиста очень высокой квалификации и хорошего знания различных тонкостей работы вычислительной машины. Вместе с тем основным направлением развития программирования на рубеже первого и второго поколений было стремление освободить программиста от необходимости знать устройство машины путем различных методов автоматизации программирования. Казалось, что внедрение микропрограммирования противоречит этому направлению.

В последние годы интерес к микропрограммированию и микропрограммным принципам управления возродился, в частности, благодаря созданию очень быстрой постоянной памяти большой емкости, позволяющей хранить длинные микропрограммы. Кроме того, изменился и подход к использованию микропрограммирования. Последнее используется не как средство повышения гибкости программирования, а как удобный метод управления работой процессора *. Благодаря возможности иметь программный доступ к состоянию процессора после каждой микрооперации составитель постоянных программ математического обеспечения (например, трансляторов) в состоянии более гибко использовать возможности машины. Программист, работающий на такой машине, может, вообще говоря, и не подозревать о ее «микропрограммной природе», используя любые алгоритмические языки, трансляторы с которых имеются в составе математического обеспечения данной машины.

Различают два основных вида микропрограммного управления: *горизонтальное* и *вертикальное микропрограммирование*. Каждая микрокоманда соответствует одной микрооперации машины, для осуществления которой необходимо подать управляющий импульс на соответствующую управляющую шину. При *горизонтальном микропрограммировании* каждой микрокоманде соответствует двоичное слово, число разрядов которого равно числу управляющих шин в устройстве управления машины. Каждый разряд этого слова закрепляется за определенной управляющей шиной, и наличие единицы в каком-либо разряде слова означает, что при выполнении данной микрокоманды по закрепленной за этим разрядом шине посылается управляющий импульс. Таким образом, одна микрокоманда

---

* Под *процессором* в современной вычислительной машине понимается комплекс основных устройств, фактически осуществляющих переработку информации в машине. Обычно к процессору относят комплекс, состоящий из арифметического устройства и устройства управления. Более подробно об этом будет идти речь в следующих параграфах.

может вызвать одновременное выполнение нескольких микроопераций.

Пусть, например, очередные разряды микрокоманды соответствуют следующим микрооперациям.

1. Гашение суммирующего регистра арифметического устройства.

2. Пересылка содержимого регистра числа запоминающего устройства в первый регистр для исходных чисел.

3. Аналогичная пересылка во второй регистр исходных чисел.

4. Пересылка содержимого первого регистра исходных чисел в суммирующий регистр прямым кодом.

5. Пересылка содержимого первого регистра исходных чисел в суммирующий регистр инверсным кодом.

6. Сдвиг содержимого суммирующего регистра на один разряд влево.

7. Сдвиг содержимого суммирующего регистра на один разряд вправо.

8. Прибавление в суммирующий регистр единицы младшего разряда.

. . . . . .

Тогда микрокоманда 11000000... означает одновременное выполнение двух микроопераций: гашение суммирующего регистра и пересылка числа из памяти в первый регистр исходных чисел. Микрокоманда 00101000... вызовет пересылку числа из памяти во второй регистр исходных чисел и одновременно пересылку из первого регистра исходных чисел в суммирующий регистр инверсным кодом.

Конечно, выполняемые микрооперации должны быть совместимы. Так, микрокоманда 00010100... невыполнима, так как она предполагает одновременную пересылку содержимого первого регистра исходных чисел в суммирующий регистр и сдвиг содержимого суммирующего регистра на один разряд влево, что нельзя выполнить в течение одного и того же такта.

Достоинством горизонтального микропрограммирования является возможность одновременного выполнения микроопераций и простота формирования функциональных импульсов: последние могут возбуждаться непосредственно от микрокоманд без промежуточных схем дешифрации. Недостатком его является большая длина микрокоманды, поскольку число функциональных импульсов в современном процессоре может достигать нескольких сотен. Для хранения микрокоманд потребуются запоминающие устройства с очень большой разрядностью ячейки; к тому же это запоминающее устройство будет использоваться крайне неэффективно, так как в каждой микрокоманде очень небольшое число разрядов будет отмечено единицами — из-за ограничений в совместимости микроопераций и последовательного характера алгоритмов их выполнения.

При *вертикальном микропрограммировании* выполняемая микрооперация определяется не состоянием одного из разрядов микрокоманды, а двоичным кодом, содержащимся в определенном поле микрокоманды. Формат микрокоманды подобен здесь формату обычной команды вычислительной машины: он состоит из кода микрооперации и адресной части, определяющей источник участвующих в микрооперации элементов и место назначения результатов. Различие между ними заключается в том, что производится более элементарное действие (микрооперация), а адресная часть чаще всего определяет не ячейки памяти, а операционные регистры процессора. Как правило, вертикальная микрокоманда реализует одну микрооперацию.

Легко представить себе необходимость условного выполнения отдельных микроопераций. В качестве простого примера можно указать прибавление множимого к частичному произведению в зависимости от значения соответствующего разряда множителя. Принципиально возможно применение микрокоманд с условными кодами, но это невыгодно, так как при невыполнении условия исполнение микрокоманды было бы пустой тратой времени. Поэтому обычно микрокоманды имеют безусловные коды микроопераций, но последовательность выполнения микрокоманд устраивается разветвляющейся в зависимости от выполнения тех или иных условий.

Эффективной процедурой разветвления для микропрограмм является *принудительное управление*, которое состоит в том, что в команде (в данном случае — в микрокоманде) всегда указывается адрес следующей, которая должна выполняться после данной. Именно такая процедура применяется в советских малых вычислительных машинах *Наири* и *Мир*, в которых используется микропрограммирование при создании трансляторов с применяемых алгоритмических языков.

Для управления последовательностью выборки микрокоманд в ее формат вводятся дополнительные поля. Предположим, для простоты, что *адресное поле* содержит n разрядов, в которых указывается адрес следующей микрокоманды, а *управляющее поле* состоит из одного разряда. Управление можно организовать так: если в управляющем поле стоит 0, то передача управления происходит безусловно, по адресу, указанному в адресном поле. Если же в управляющем поле записана 1, то фактическое содержимое самого младшего разряда адресного поля игнорируется и оно принимается равным нулю или единице в зависимости от того, выполнено или нет соответствующее данной микрооперации условие. Если управляющее поле состоит из нескольких разрядов, то можно обеспечить проверку нескольких условий и соответствующее разветвление на 4, 8 и т. д. лучей.

На основе микропрограммирования создаются весьма эффективные системы тестов для автоматической диагностики работы отдельных устройств и, в особенности, процессора вычислительной машины. Особенно важным достоинством микропрограммных систем

является то, что они в заметной степени облегчают создание *программ-имитаторов*, позволяющих имитировать одну вычислительную машину с помощью другой, т. е. выполнять на данной машине программу, составленную в кодах другой машины (эту способность иногда называют *эмуляцией*).

§ 6. ВНЕШНИЕ УСТРОЙСТВА

Устройства ввода и вывода информации по сравнению с основными устройствами машины (арифметическим устройством и памятью) работают чрезвычайно медленно. Чтобы нагляднее уяснить различие в скоростях работы этих устройств, представим себе, что скорость работы цифровой машины замедлилась в миллион раз. Тогда арифметическое устройство, скажем, машины БЭСМ-6 будет выполнять в среднем одну операцию в секунду. Вместе с тем читающее устройство, способное вводить при обычной скорости до 1200 перфокарт в минуту, будет при таком замедлении вводить по одному машинному слову реже чем за час, а быстропечатающее устройство, выдающее 20 чисел в секунду, будет печатать одно число приблизительно за 14 часов. Такое «медленнодействие», обусловленное применением чисто механических приспособлений и элементов, иногда ставит под сомнение само применение вычислительных машин.

Другое неудобство «классических» устройств вывода состоит в том, что они представляют информацию в виде слов и чисел, тогда как очень часто осмысление человеком полученных результатов требует совсем других средств их представления: диаграмм, графиков, чертежей, эскизов, которые приходится все равно строить по полученным числам, но уже вручную. Точно так же и при вводе — цифровая вычислительная машина может воспринимать только алфавитно-цифровую информацию, и чтобы ввести в машину, например, график, его приходится предварительно вручную перекодировать в числовую форму.

Эффективное взаимодействие человека с машиной в форме «диалога» практически невозможно, ввиду медленности работы внешних устройств и трудности обмена информацией. Перечисленные затруднения направили усилия конструкторов машин следующих поколений на:

1. Увеличение быстродействия устройств ввода и вывода.

2. Создание устройств, позволяющих вводить в машину или выводить из нее информацию в виде графика или чертежа (или другого вида изображения).

3. Использование устройств и программных средств, позволяющих вести эффективный диалог человек — машина.

Работы в первом из указанных направлений привели к созданию значительно более быстрых, нежели барабанные, *цепных* печата-

ющих устройств. Цепное печатающее устройство содержит несколько полных шрифтов на одной цепи или ремне (рис. 101). Молоточки, находящиеся за бумагой, кратковременно прижимают ее к соответствующим знакам шрифта в нужное время. Схемы управления такого устройства более сложны, чем барабанного, так как они должны принимать много решений «печатать — не печатать» для каждого молоточка в процессе печати одной строки.

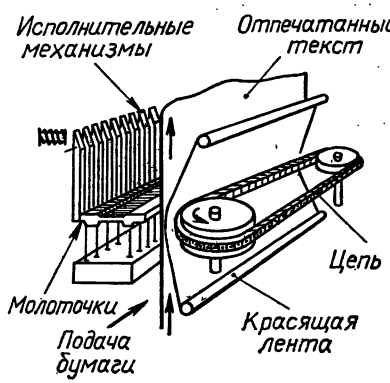


Рис. 101

Наряду с ударными печатающими устройствами в последнее время начали производиться и *безударные*, которые отличаются бесшумностью, значительно большим быстродействием и высокой надежностью, хотя и более дорогостоящи. К безударным печатающим устройствам относятся электростатические, термографические, струйные и магнитографические. В *электростатическом* печатающем устройстве имеются иголки, с помощью которых на бумагу наносятся электрические заряды, образующие рисунок знака; эти заряды притягивают порошковый краситель, который под воздействием тепла образует законченные изображения.

В *термографическом* печатающем устройстве используется тепло для получения темных изображений знаков на предварительно обработанной специальной бумаге. Печатающая головка содержит ряд миниатюрных нагревательных элементов. В *струйном* печатающем устройстве краситель проходит через форсунку, которая заряжает его частицы. Распыленная струя проходит затем между отклоняющими электродами, под управлением которых она меняет свое направление и «рисует» требуемые знаки на бумаге. Разработанные *магнитографические* печатающие устройства представляют собой магнитный аналог электростатического. Они отличаются от электростатических тем, что содержат промежуточный носитель — магнитную ленту. Изображение сначала формируется на ленте, а затем переносится на бумагу.

Характерной особенностью машин третьего и четвертого поколений является наличие у них устройств для ввода и вывода графических изображений. Остановимся сначала на устройстве вывода с помощью электроннолучевой трубки, которое носит название *дисплея*. О нем мы уже упоминали в § 1. Блок-схема дисплея приведена на рисунке 102.

Массив данных, подлежащий выводу на экран дисплея, по сигналу устройства управления передается из памяти машины в буферную память, емкость которой соответствует размеру кадра, изображаемого на экране электроннолучевой трубки. Обычно это составляет около 2 тыс. знаков. После заполнения буферной памяти дисплея он работает автономно: под воздействием управляю-

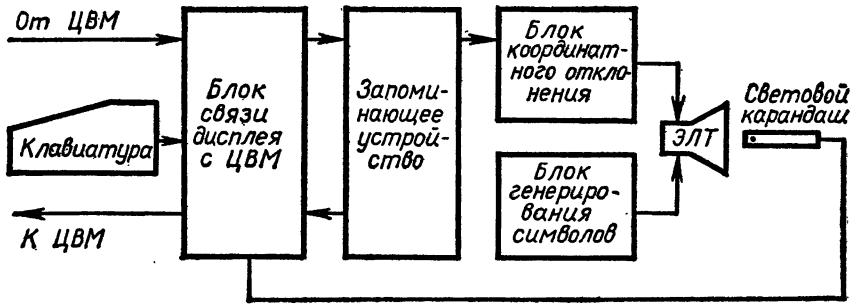


Рис. 102

щих сигналов *блока связи* циклически производится опрос ячеек буферной памяти, извлечение кодов символов, их расшифровка и изображение на экране. Частота опроса выбирается так, чтобы на экране не возникало мерцание изображения.

Формирование изображения на экране определяется двумя блоками дисплея: *блоком координатного отклонения* и *блоком генерирования символов*. Блок координатного отклонения содержит два цифроаналоговых преобразователя (см. § 9 следующей главы), на входы которых поступают коды координат. Выходные напряжения усиливаются и подаются на отклоняющие пластины электроннолучевой трубки, которые направляют электронный луч в нужную точку экрана.

Блок генерирования символов бывает *растровым* или *функциональным*. В первом случае на экране получают развертку, подобную телевизионной. Электронный луч прочерчивает по экрану горизонтальные линии. Изображение возникает благодаря подсвечиванию отдельных точек или линий растра, который развертывают по всему экрану трубки. При функциональном способе генерирования символов электронный луч «рисует» требуемый знак на экране под действием переменных напряжений, отклоняющих его вдоль координатных осей.

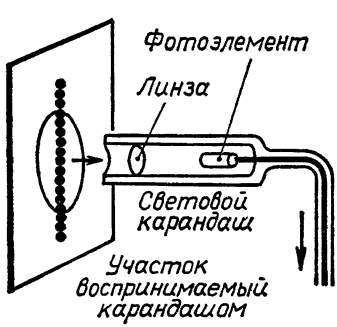


Рис. 103

Чаще всего дисплей совмещается со «световым карандашом» (рис. 103), который позволяет вводить в машину графическую информацию. На его «острие» устанавливается элемент, чувствительный к свету, например фотодиод; этот элемент выдает сигнал, когда в его «поле зрения» попадает свет. Поднеся карандаш к экрану, мы получим импульс, который передается в блок

связи (см. блок-схему на рис. 102). Полученный сигнал позволяет определить адрес ячейки буферной памяти, содержащей соответствующий символ, благодаря чему световой карандаш можно использовать для редактирования выведенной информации.

Для ввода графической информации нужно сообщить машине координаты начальной точки и «вычертить» нужную линию на экране трубки. Координаты начальной точки задаются, как описано в предыдущем абзаце. При вычерчивании острием карандаша линии на экране электроннолучевой трубки производится слежение за положением острия относительно «точечного перекрестия». Последнее формируется на экране системой управления трубки. Центр перекрестия высвечивается более интенсивно, остальные точки остаются слабо подсвеченными.

§ 7. СОВМЕЩЕНИЕ ОПЕРАЦИЙ

Предыдущие параграфы настоящей главы были посвящены в основном физическим и техническим принципам и устройствам, позволяющим ускорить работу вычислительной машины. Но новые поколения вычислительных машин характеризуются не только технологическими, но и организационными новшествами. На них мы и остановимся в этом и следующих параграфах. Прежде всего, необходимо ввести один, часто употребляемый в современной вычислительной технике, термин. Арифметическое устройство машины и устройство управления вместе образуют *процессор* вычислительной машины. *Процессор* предназначен для реализации программы, т. е. для выполнения отдельных команд в заданной последовательности. Иногда говорят даже о *центральном процессоре*, имея в виду, что некоторые функции по исполнению отдельных команд будут возлагаться на периферийные устройства, к которым, как мы уже говорили, принято относить устройства ввода и вывода и внешнюю память.

Быстродействие машины определяется не только быстродействием процессора и памяти, но и их взаимодействием. Так, в трехадресной машине исходные числа, над которыми выполняется требуемая операция, вызываются из памяти, даже в том случае, когда одно из них есть результат предыдущей операции. В одно- и двухадресных машинах это не требуется, так как результат операции хранится в сумматоре и может быть использован для следующей операции непосредственно. Но даже и для этих машин может оказаться необходимым обращение к памяти для запоминания промежуточных результатов, не говоря уже о выборке хотя бы одного из требуемых чисел.

Одним из способов ускорения работы машины в этих условиях является сверхоперативная память, о которой мы уже говорили в предыдущих параграфах. Небольшое число ячеек сверхбыстродей-

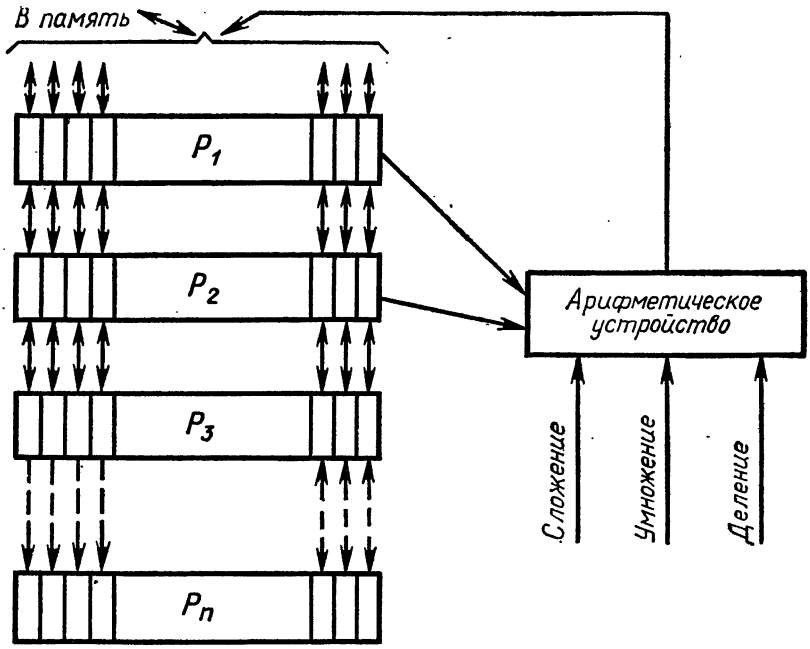


Рис. 104

ствующей памяти используется для хранения промежуточных результатов, что заметно ускоряет вычисления по формуле.

Другим осуществлением аналогичной идеи является *процессор с гнездовой памятью*. Гнездовая память представляет собой набор отдельных регистров, каждый из которых хранит одно машинное слово. Информация может передвигаться между регистрами в двух направлениях: вниз и вверх. Продвижение вниз означает передачу информации из P_{n-1} в P_n, . . ., из P_1 в P_2. При продвижении вверх крайний нижний регистр P_n заполняется нулями, при продвижении вниз регистр P_1 получает информацию из памяти. Продвижения вверх и вниз могут чередоваться в любой последовательности, обеспечивая заполнение и перераспределение гнездовой памяти.

Верхние регистры P_1 и P_2 связаны с арифметическим устройством, которое берет их содержимое в качестве исходных чисел для операции и записывает результат в P_1; одновременно осуществляется продвижение вверх, не затрагивающее регистр P_1 (рис. 104). Таким образом, эти команды могут быть безадресными, тогда как команды, осуществляющие обмен информацией между гнездовой и главной памятью, требуют указания адресов. Поэтому машины с гнездовой памятью используют команды переменной длины. Кроме того, они имеют некоторые необычные команды, например *дублирование*, состоящее в сдвиге вниз содержимого всех регистров, начиная с

P_2, и передачи содержимого P_1 в P_2 с сохранением P_1, а также *реверсирование*, состоящее в обмене содержимого регистров P_1 и P_2 без изменения остальных.

Обращение к гнездовой памяти и выполнение сдвигов и прочих команд в ней выполняется значительно быстрее, чем обращение к оперативной памяти. Поэтому процессор с гнездовой памятью оказывается более быстродействующим. Многие команды являются здесь безадресными, благодаря чему уменьшается место в памяти для хранения программы и количество обращений. Еще одно преимущество гнездовой памяти состоит в том, что при переходе к подпрограммам или в случае прерывания работы программы по каким-либо внешним причинам (см. следующий параграф) нет необходимости запоминать содержимое арифметических регистров. Программа может немедленно начать выполнение требуемых действий; данные, соответствующие предыдущей программе, автоматически продвигаются вниз и возвращаются обратно, когда новая программа закончит работу.

Серьезным резервом повышения быстродействия работы вычислительной машины за счет организации является *совмещение операций* или *организация параллельной работы*. Речь идет об одновременном выполнении некоторых операций различными устройствами машины. Остановимся на этом более подробно.

Цикл работы внешних устройств обычно значительно продолжительнее цикла передачи подготовленной ими информации в оперативную память или обратно. Это дает возможность организации параллельной работы нескольких периферийных устройств или работы одного периферийного устройства одновременно с продолжением работы процессора. Для одновременной работы периферийного устройства с процессором или нескольких периферийных устройств необходимо, чтобы процессор мог только запускать требуемое устройство, независимо от работы остальных. После этого процессор должен освобождаться для продолжения своей работы, а запущенные периферийные устройства могли бы продолжать работу автономно, занимая время процессора только на короткие «сеансы связи» для передачи информации в оперативную память и извещая программу сигналом об окончании своей работы.

Реализация этих требований вызывает выделение схем управления внешними устройствами из состава процессора и придания им известной самостоятельности. С этой целью в состав вычислительной машины или системы вводят специальные устройства, обеспечивающие обмен информацией между оперативной памятью и внешними устройствами, которые называют *каналами связи* или, более узко, *каналами ввода-вывода*. Выполнение программы с использованием канала происходит следующим образом.

Процессор выбирает из памяти команду и данные и выполняет программу. Если по ходу программы встречается необходимость в обмене информацией с внешними устройствами, то процессор посылает в канал управляющую информацию, после чего продолжает

свою работу. Канал же преобразует полученную информацию в последовательность сигналов, которые передаются в управление выбранного внешнего устройства; запуск его в работу происходит по сигналу канала, а не центрального процессора. Дальнейшая работа канала протекает совместно с работающим внешним устройством в темпе, определяемом скоростью приема или выдачи информации. Запросы на передачу информации, возникающие во внешнем устройстве, анализируются и обслуживаются каналом.

Медленная (сравнительно с каналом) работа отдельных периферийных устройств приводит к тому, что запросы на их обслуживание формируются сравнительно редко. Это дает возможность использовать канал для последовательного во времени обслуживания нескольких параллельно работающих периферийных устройств. Различные медленнодействующие периферийные устройства поочередно подключаются к быстродействующему каналу. Такой канал называется *мультиплексным*.

Физически мультиплексный канал представляет собой небольшую самостоятельную цифровую вычислительную машину, которая должна выполнять следующие функции:

1. Получать команды обмена (в частности, ввода-вывода) от центрального процессора.
2. Обращаться к периферийному устройству, указанному в команде.
3. Выбирать из оперативной памяти машины управляющую информацию, относящуюся к данному каналу, и расшифровывать ее.
4. Посылать управляющие сигналы в периферийные устройства и получать оттуда ответные.
5. Обеспечивать промежуточное хранение передаваемых в машину или из машины данных.
6. Сохранять и пересылать в центральный процессор информацию о состоянии периферийных устройств и канала в целом.

Канал связывается с оперативной памятью машины системой информационных и управляющих шин, с периферийными устройствами — через *сопряжение*, представляющее систему линий, по которым передаются управляющие и информационные сигналы в единой форме. В зависимости от путей передачи информации различают каналы с *прямым* и *косвенным доступом* к памяти. При косвенном доступе информация из канала поступает в регистры процессора, а связь с оперативной памятью имеет только процессор. Это может привести к задержкам в обслуживании периферийных устройств, так как сеанс связи может происходить только после выполнения очередной команды, когда освободятся все регистры процессора. Каналы с прямым доступом имеют собственную аппаратуру для связи и могут обращаться к памяти независимо от процессора. Это удобнее, но дороже.

Наряду с мультиплексным каналом, обслуживающим несколько одновременно работающих периферийных устройств, существует и *селекторный канал*, который позволяет выполнять свою деятель-

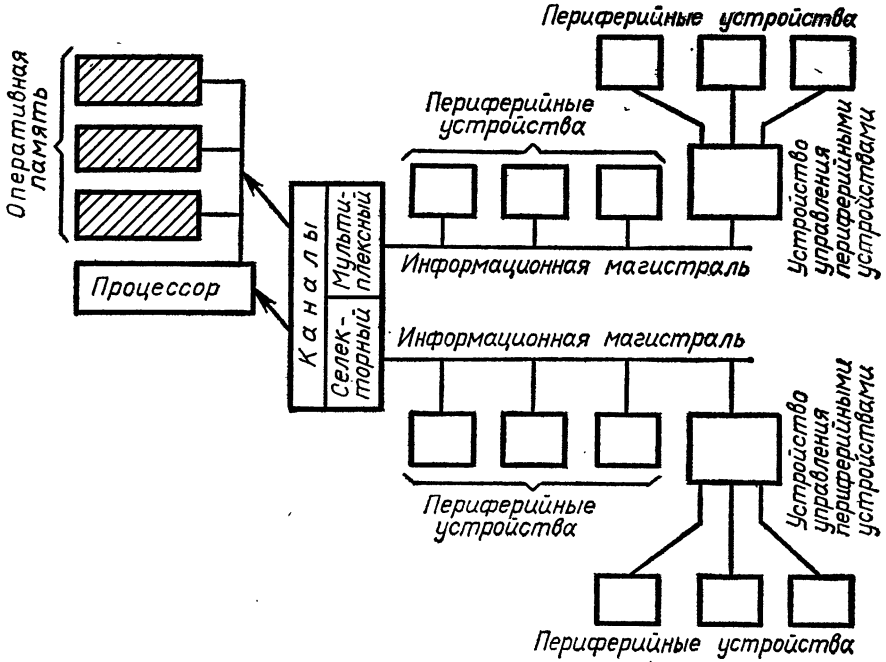


Рис. 105

ность только одному из присоединенных к нему периферийных устройств; остальные должны в это время бездействовать. Селекторные каналы обслуживают такие периферийные устройства, при обращении к которым информация передается большими массивами с высокой скоростью. При этом периферийное устройство занимает аппаратуру канала в течение всей операции обмена. Блок-схема вычислительной машины с каналами связи приведена на рисунке 105.

Совмещение операций можно распространить не только на работу периферийных устройств. Одна из возможностей такого совмещения состоит в *модульной организации памяти*, при которой запоминающее устройство собирается из нескольких автономных модулей, имеющих собственные устройства управления и потому способных работать одновременно.

§ 8. МУЛЬТИПРОГРАММИРОВАНИЕ И РАЗДЕЛЕНИЕ ВРЕМЕНИ

Совмещение операций, описанное в предыдущем параграфе, позволяет несколько уменьшить потери времени при работе вычислительной машины. Однако в начале работы, когда программа решения задачи только вводится в машину, работа процессора одновре-

менно с устройством ввода невозможна, так как ему попросту еще нечего делать. Аналогичная ситуация возникает и в конце работы: вычисления закончены, работает медленное устройство вывода результатов, а процессор простаивает.

Вынужденные простои отнюдь не исчерпываются началом и концом решения задачи. Они возможны и в процессе решения. Например, в некоторый момент необходимо обработать массив, который вводится в оперативную память из внешней; до окончания ввода процессор должен простаивать. То же самое происходит при остановке машины в результате ошибки в неотлаженной программе или при работе оператора за пультом. Каждая секунда простоя процессора означает потерю около миллиона операций. Описанные обстоятельства приводят к идее так организовать работу машины, чтобы несколько программ выполнялись ею одновременно.

Режим работы вычислительной машины, при котором не требуется окончания одной программы для пуска или продолжения работы другой, называют *мультипрограммным* или *режимом мультипрограммирования*. При этом режиме процессор машины работает, как обычно, последовательно, выполняя в каждый данный момент только одну очередную команду. Речь идет только о том, что после выполнения части программы одной задачи машина может перейти к выполнению другой, сохраняя возможность возвратиться к окончанию предыдущей.

Мультипрограммные вычислительные системы могут принадлежать к одному из двух типов: системы *с пакетной обработкой* и *системы с разделением времени*. При разделении времени некоторое число пользователей (иначе говоря, некоторое число задач) имеют одновременный доступ к машине. Каждому пользователю предоставляются собственные средства для связи с машиной — периферийные устройства и канал связи. Работа пользователей является независимой, т. е. они могут ставить совершенно не связанные между собой задачи. За каждым пользователем закрепляется определенная часть оперативной памяти и ресурсы внешней, если это требуется. Процессор ведет обработку задач поочередно, разделение времени между задачами производится так, чтобы обеспечить максимально возможное совмещение операций. Эта организация осуществляется аппаратно и требует специального дополнительного оборудования.

Пакетная обработка задач обеспечивается полностью программным путем и не требует никакого дополнительного оборудования, кроме *системы прерывания*. Решающую роль в системе пакетной обработки играет специальная управляющая программа, которая и организует поочередную передачу управления программам различных задач. Такую программу называют *диспетчером*. Подробное рассмотрение работы диспетчера мы отложим до § 10, а сейчас остановимся на системе прерывания и способах ее осуществления.

Явление *прерывания программы* состоит в том, что вычислительная машина (или система) по определенным сигналам временно при-

останавливает работу над текущей программой и передает управление другой программе. Соответствующие сигналы называются *запросами* или *сигналами прерывания*, а совокупность аппаратно-логических средств, с помощью которых реализуется прерывание, называется *системой прерывания* программы.

Основной составной частью системы прерывания является *регистр прерывания*, в котором на каждую причину прерывания отводится отдельный разряд. Причинами прерывания могут быть:

а) Запрос от канала связи; он формируется в тех случаях, когда канал или периферийное устройство должны начать выполнение соответствующей команды (ввод, вывод, обмен) или закончили ее выполнение.

б) Остановка процессора по команде *стоп* или аварийная остановка (*авост*) в результате ошибки в программе или неверных значений исходных чисел — деление на нуль, переполнение разрядной сетки и т. п.

в) Обращение к ячейке памяти, не относящейся к работе данной программы (*защита памяти*). На этом вопросе мы подробнее остановимся ниже.

г) Сигналы от внешних объектов — ключей или клавиш запроса на пульте управления, «часов» (датчиков времени) и т. п.

д) Обращение рабочей программы к диспетчеру.

е) Сигналы от схем контроля; они возникают при обнаружении схемами контроля ошибок в работе системы.

Каждый такой сигнал вызывает прерывание, т. е. прекращение процессором выполнения данной программы. После этого следует проанализировать причину прерывания и в зависимости от нее организовать дальнейшую работу процессора. Некоторую часть такого анализа можно производить аппаратно, но обычно эта деятельность поручается программе, которая носит название *программы анализа прерываний* и входит в состав диспетчера. Вместе с ним она и будет рассмотрена ниже.

Несколько запросов прерывания могут возникнуть одновременно либо один из них может прийти во время работы программы анализа, вызванной предыдущим запросом. Для того чтобы не происходило путаницы и нарушений правильности работы вычислительной системы, должен быть установлен *приоритет прерывания*, исследование которого в каждом конкретном случае также возлагается на программу анализа прерываний.

Так как при мультипрограммной работе в оперативной памяти машины нужно одновременно хранить несколько программ, то необходимо принять специальные меры для предохранения каждой из них от порчи другими. Если какая-либо из программ содержит ошибку, то в процессе своей работы она может испортить другую программу, которая в ошибке не повинна. Другой причиной нарушений может быть ошибка оператора за пультом. Особенно опасно, если испорченной программой окажется библиотечная программа или диспетчер.

Для предотвращения таких неприятностей принимаются специальные меры, называемые защитой памяти. Первая из них состоит в введении *граничных регистров*. При распределении памяти между задачами, которое производится диспетчером в начале работы над пакетом, для каждой задачи вводятся два граничных регистра, указывающие верхнюю и нижнюю границы области памяти, отведенной для данной задачи. Каждое обращение к памяти предполагает предварительную проверку того, что используемый адрес находится в отведенных границах. В противном случае формируется запрос прерывания, приостанавливающий работу программы и передающий управление диспетчеру.

Вторая мера является более гибкой; она основана на страничной структуре памяти. Вся память объемом 2^n машинных слов разбивается на 2^s страниц объемом 2^k машинных слов каждая ($s+k=n$). Каждой программе выделяется определенная совокупность страниц. В триггерном *регистре защиты* памяти каждой странице соответствует определенный разряд (триггер), куда заносится 1, если программе разрешено пользоваться данной страницей, и 0 — в противном случае. Содержимое регистра защиты для данной задачи называют ее *маской защиты памяти*. Допустимость обращения данной задачи к какой-либо ячейке памяти проверяется теперь по маске защиты.

Запись в «чужие» ячейки, конечно, является наиболее опасной ошибкой, могущей испортить другие программы. Чтение из «чужих» ячеек этим не грозит, но также является ошибкой, которую целесообразно «выловить». Поэтому часто бывает полезно усложнить маску защиты, поставив в соответствие каждой странице *указатель отношения*, содержащий д в а р а з р я д а. Четыре его состояния можно интерпретировать, например, следующим образом:

11) Разрешается и считывание, и запись в ячейки данной страницы.

10) Разрешается только считывание.

01) Разрешается обращение только по счетчику команд.

00) Разрешается любое обращение, кроме обращения по счетчику команд.

§ 9. ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

В этой главе читателю уже неоднократно встречался термин *вычислительная система* либо как синоним термина «вычислительная машина», либо в контексте «вычислительная машина или система». Дело в том, что *машине*, состоящей из определенного числа устройств определенного назначения, все в большей степени приходит на смену *система*, представляющая собой объединение различных устройств, количество и назначение которых может заметным образом изменяться в зависимости от потребностей.

Основными составляющими вычислительной системы являются *главная* (оперативная) *память* и *центральный процессор*. Кроме них, система должна содержать также *каналы связи*, которые, как мы видели уже в § 7, сами представляют собой процессоры, хотя и со специфическими функциями. Число каналов и присоединенных к ним периферийных устройств (ввода, вывода и внешней памяти) может быть разным.

Разнообразные области применения вычислительных машин (которым будет посвящена следующая глава нашей книги) предъявляют различные требования не только к составу периферийных устройств и объему оперативной памяти, но и к процессорам. Поэтому естественным развитием вычислительных систем явилось создание *рядов моделей процессоров*, обладающих различными характеристиками, к которым через унифицированные каналы могут присоединяться любые периферийные устройства, общие для всей вычислительной системы. Примером такой системы моделей является *единая система* ЕС ЭВМ, разрабатывавшаяся странами — участницами Совета Экономической Взаимопомощи в 1970—1973 годах. Состав ее приведен на рисунке 106, где 1 — процессор; 2 — основная оперативная память; 3 — мультиплексный канал; 4 — селекторный канал; 5 — накопитель на сменных магнитных дисках; 6 — накопитель на магнитной ленте; 7— устройство управления; 8 — дисплей; 9 — устройство ввода с перфокарт; 10 —устройство ввода с перфоленты; 11 — устройство вывода на перфокарты; 12 — устройство вывода на перфоленту; 13 — печатающее устройство; 14 — пишущая машинка; 15 — графопостроитель; 16 — стойка питания.

Для вычислительных систем, включающих ряд моделей процессоров, большое значение имеет их информационная и программная совместимость. *Информационная совместимость* моделей предполагает единые для всего ряда способ кодирования информации, фор-

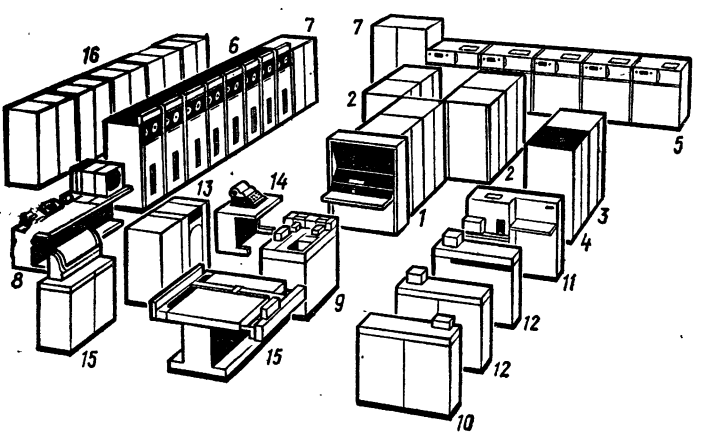


Рис. 106

маты данных и одинаковые длины машинных слов в различных моделях. *Программная совместимость* означает, что программы, составленные для машины одной модели, могут выполняться на других моделях ряда. При этом системы команд машин различных моделей могут различаться; необходимо только, чтобы на модели с меньшим набором команд отсутствующие команды могли быть промоделированы.

Так как процессор представляет собой всего лишь агрегат вычислительной системы, то вычислительная система может содержать одновременно несколько процессоров одного или различных типов, использующих общее или раздельные оперативные запоминающие устройства. Такие системы называют *мультипроцессорными (многопроцессорными)*. Функции по обработке информации и управлению могут распределяться в такой системе между отдельными звеньями подобно обычному аппарату управления. Здесь могут быть выделены системы низшего и более высокого уровня управления и между ними установлены отношения соподчинения. Такие многоуровневые мультипроцессорные вычислительные системы получили название *иерархических*.

Любые современные вычислительные машины в принципе можно объединить в мультипроцессорную вычислительную систему. Однако для этого недостаточно каналов связи и устройств сопряжения. Нужны еще программы, организующие обмен информации между машинами и распределение задач между процессорами системы. Только тогда случайное объединение разрозненных вычислительных машин превратится в вычислительную систему.

Эффект мультипроцессорной вычислительной системы весьма велик. Мы не будем подробно останавливаться на всех деталях, отметим лишь основное и наиболее существенное. Прежде всего, мультипроцессорная система дает возможность вести параллельные вычисления во всех случаях, когда это допускается задачей. Благодаря этому многие задачи могут быть решены значительно быстрее. Далее, вычислительная система может использовать единую библиотеку стандартных программ и трансляторов, а не для каждого процессора в отдельности. Наконец, в вычислительном комплексе, состоящем из различных процессоров, весьма эффективна *функциональная специализация*. Ее идея состоит в том, что выполнение различных алгоритмов целесообразно поручать различным процессорам, имеющим определенную специализацию в том или ином направлении.

Наряду с параллельным соединением процессоров, о котором мы говорили в предыдущем абзаце, возможна другая организация вычислительной системы, которую называют *магистральной*. При магистральной организации системы поток данных организуется таким образом, что выходные данные одного процессора служат входными для другого. В зависимости от конкретной задачи та или иная организация системы будет более выгодна и удобна. Но и в том и в другом случае вычислительная система обладает *модульной*

структурой: отдельные подсистемы — блоки ввода-вывода, устройства внешней памяти, процессоры, модули оперативной памяти, выносные пульты и т. п.— связаны между собой таким образом, что каждый отдельный модуль сохраняет определенную независимость.

§ 10. МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

Математическое обеспечение вычислительной машины или системы представляет собой совокупность программ, организующих нормальную и своевременную работу всех устройств и нормальную эксплуатацию системы в целом. Для современных машин и систем математическое обеспечение играет совершенно особую роль, сделавшись по существу неотъемлемой составной частью системы.

В общесистемном математическом обеспечении, которое используется большинством потребителей, следует особо выделить обслуживающие программы, которые не выполняют работы по р е ш е н и ю задач, а играют лишь вспомогательную роль. Это трансляторы с алгоритмических языков, организующие и редактирующие программы ввода-вывода и обмена, средства отладки и т. п. Значение таких программ тем больше, чем мощнее используемые в вычислительной системе технические средства и чем сложнее взаимодействие между ними.

Совокупность обслуживающих программ, организующих работу вычислительной системы, образует *операционную систему*. Операционная система планирует решение задач, следит за его выполнением, выделяет задачам различные технические средства — каналы связи, периферийные устройства, участки оперативной памяти, создает различные режимы решения задач (трансляция, отладка, счет). При мультипрограммном режиме операционная система выявляет аварийные ситуации, возникающие в процессе работы, анализирует причины прерывания и распределяет время процессора для различных задач данного пакета, а также организует первоначальный ввод пакета и вывод получающихся результатов. В мультипроцессорной вычислительной системе операционная система распределяет работу между различными процессорами, организует работу каналов связи по обмену информацией и т. п.

В операционную систему входят трансляторы с алгоритмических языков, вспомогательные программы выявления и печати синтаксических и семантических ошибок, различные отладочные программы и библиотека стандартных подпрограмм. В библиотеку обычно входят редактирующие программы ввода-вывода и обмена и программы наиболее употребительных численных методов и алгоритмов (обращение матрицы, решение систем линейных уравнений, численное интегрирование, численное решение дифференциальных уравнений, вычисление некоторых специальных функций, отыска-

ние экстремума функций нескольких переменных, статистическая обработка экспериментальных данных и др.).

Одной из основных функций операционной системы является организация непрерывной работы процессора для обработки пакета задач. Блок операционной системы, выполняющий эту функцию, обычно называют *диспетчером* или *супервизором* (иногда *супервайзером*, английское *supervisor*). Диспетчер может организовать решение пакета задач в режиме *последовательного обслуживания*, *мультипрограммирования* и *разделения времени*.

В *режиме последовательного обслуживания* (в однопроцессорной системе) процессор системы выполняет программы задач последовательно. В этом режиме каждой задаче на время ее решения может быть предоставлено все имеющееся оборудование вычислительной системы, кроме главной памяти. Главная память разделена между программами тех задач, которые находятся в памяти в настоящее время. Для каждой задачи в распоряжении диспетчера имеется *шкала защиты памяти*, в которой единицами помечены разрешенные для данной задачи страницы памяти.

Начальное распределение памяти осуществляется блоком операционной системы, который носит название *загрузчика*. Программа впервые решаемой задачи нанесена на внешнем носителе и имеет вид колоды перфокарт или бобины с перфолентой. В начале программы помещается *паспорт задачи;* в нем указывается номер или название задачи, примерное время ее решения, характер работы (трансляция, отладка, пробный счет, вариантные расчеты и т. п.), требуемый объем внешней памяти, перечень необходимых внешних устройств и прочие аналогичные сведения. Колоды перфокарт (или ленты) для нескольких задач, объединенные вместе, образуют *пакет задач*.

Загрузчик, обращение к которому происходит по сигналу оператора, подает сигнал на включение устройства ввода и организует *начальную загрузку*. Программы пакета последовательно вводятся в оперативную память и размещаются затем во внешней памяти системы. Затем некоторая часть задач пакета вызывается вновь в оперативную (главную) память и размещается таким образом, чтобы для совокупности всех вызванных задач хватило ресурсов системы — главной памяти и периферийных устройств.

Диспетчер работает в так называемом *режиме управления*, который характеризуется запрещением любых прерываний. Закончив начальную загрузку, диспетчер передает управление первой задаче, предварительно перенеся ее шкалу защиты в рабочий регистр защиты памяти. Рабочая программа математика работает уже в *рабочем режиме*, при котором возможны прерывания. Всякое прерывание вызывает приостановку выполнения программы и передачу управления диспетчеру, именно его блоку *Анализ прерываний*.

Блок *Анализ прерываний* по регистру прерывания, о котором шла речь в § 8, анализирует причины прерывания и «принимает соответствующие меры». Наиболее приятной причиной прерывания

является прерывание по команде *стоп*, что означает завершение работы по данной программе. В этом случае диспетчер включает в работу программу следующей задачи, внеся одновременно всю часть оперативной (и внешней) памяти, занятую первой программой, в свой «резерв», считая ее отныне свободной. Выбор следующей задачи может производиться различными путями. Наиболее просто принять за следующую ту задачу, которая стояла в пакете при загрузке за уже завершенной. Так обычно и поступают, если все задачи пакета можно считать равноправными. Однако это не всегда так, и тогда приходится прибегать к «системе приоритетов», в которой каждой рабочей программе ставится в соответствие определенный «приоритет», и диспетчер выбирает из числа введенных рабочих программ пакета ту, которая обладает «старшим приоритетом».

Менее приятным является прерывание по аварийной ситуации — *авост* или обращение к защищенной странице памяти. Сколь ни мало приятна такая ситуация, она, как и в предыдущем случае, завершается прекращением работы над данной задачей; но в этом случае необходимо проинформировать заказчика о происшедшем. Диспетчер производит в таком случае «аварийную выдачу», т. е. печатает все те данные, которые предусмотрены к выводу в данной ситуации, например номер или название задачи, причину прекращения работы, содержимое ячеек, участвовавших в последней операции, и т. п. сведения.

Часто встречающаяся причина прерывания — обращение к диспетчеру в связи с необходимостью использования периферийных устройств. Это может быть, например, потребность «подкачки», т. е. надобность, ввиду завершения работы над предыдущей, перенести из внешней памяти в оперативную следующую часть программы, ранее здесь не умещавшуюся. В другом случае это может быть надобность ввести для обработки новый массив числовых данных из внешней памяти или с первичного носителя через читающее устройство. Наконец, это может быть потребность вывести полученные результаты вычислений на печать или другое устройство вывода или перенести их на внешнюю память, чтобы продолжить работу над ними впоследствии. В обращении к диспетчеру указываются тип и номер периферийного устройства, которым программе необходимо воспользоваться, границы обмениваемого массива в оперативной и внешней памяти и другая требуемая информация.

По информации, содержавшейся в запросе, диспетчер формирует команды обмена, защищает обмениваемые страницы памяти от обращения к ним до завершения обмена, включает в работу требуемые периферийные устройства и передает управление продолжению программы, если это возможно. В дальнейшем может представиться один из двух следующих случаев: 1) продолжение программы возможно и обмен завершился раньше, нежели произошло обращение процессора к какой-либо из ячеек обмениваемого массива; в этом случае выполнение программы ничем не нарушается; 2) продолжение программы невозможно с самого начала обмена, либо поначалу

программа могла продолжаться, но обращение к ячейке обмениваемого массива произошло раньше, нежели обмен закончился. Этот случай требует от диспетчера особого решения.

В этом случае решение диспетчера зависит от режима работы. При работе в режиме последовательного обслуживания он должен будет после прерывания по защите памяти, которое здесь произойдет, организовать искусственное ожидание окончания обмена. Некоторое время — до завершения обмена — процессор будет работать вхолостую и лишь после прерывания по окончании обмена диспетчер организует продолжение работы над данной программой. В режиме мультипрограммирования в таком случае управление, естественно, передается следующей задаче, которая выбирается, как о том говорилось выше. Именно в такой ликвидации «простоев» процессора и заключен весь смысл организации мультипрограммной работы вычислительной системы.

Обращение к другой программе требует предварительного запоминания состояния машины, которое должно быть восстановлено при возобновлении работы по предыдущей программе. Под *состоянием машины* в данном случае мы понимаем совокупность содержимого всех основных управляющих регистров — счетчика команд, регистра команд, сумматора, регистра прерывания, регистра защиты памяти и т. п. Такое запоминание организуется диспетчером в специальных рабочих ячейках вместе с номером программы, к которой соответствующая информация относится. При новой передаче управления этой программе содержимое всех управляющих регистров восстанавливается с помощью информации, сохраненной в этих ячейках.

Еще одним возможным режимом работы вычислительной системы является режим разделения времени, о котором мы бегло упоминали в § 8. Остановимся на нем несколько подробнее. Мультипрограммный режим пакетной обработки задач, который мы рассматривали выше, предусматривает *автоматическую* работу вычислительной системы с минимальным вмешательством оператора. Между тем для некоторых типов задач весьма полезна или даже необходима работа в форме если не непрерывного диалога «человек-машина», то, во всяком случае, довольно регулярного их общения. Такую возможность предоставляет режим разделения времени.

Как мы уже отмечали в § 8, для обеспечения работы в режиме разделения времени потребители должны иметь в своем распоряжении периферийные устройства, являющиеся *выносными* или *оконечными пультами*. Чаще всего в качестве таких пультов применяются специальные электрические пишущие машинки или телетайпы, используемые в телеграфной аппаратуре. С помощью таких пультов операторы могут вводить нужную информацию непосредственно в главную память; вывод ответной информации производится с помощью печатающего механизма того же пульта. К большой вычислительной системе подсоединяется до нескольких сотен таких пультов. Запрос пульта воспринимается процессором как сигнал преры-

вания, который приостанавливает работу над программой и передает управление специальным блокам диспетчера, выполняющим требуемые команды. Периферийный процессор канала связи предоставляет возможность передавать соответствующие распоряжения всем подключенным к данному каналу пультам поочередно или в соответствии с установленными приоритетами. В промежутках же между прерываниями процессор выполняет поступившие заявки по мере их поступления, также учитывая приоритет.

Необходимой составной частью системы с разделением времени являются «электронные часы», фиксирующие как абсолютное астрономическое время, так и относительное показание времени с помощью счета числа синхронизирующих импульсов. Работа системы организуется обычно таким образом, что для каждой выполняемой задачи отводится некоторое определенное количество «чистого времени» (без учета расходов на прерывания по внешним причинам). Если в течение этого времени выполнение программы не заканчивается, то работа по ней прерывается и задача ставится в конец очереди на обслуживание, а управление передается следующей, конечно, с учетом приоритета.

Такая организация работы вычислительной системы приводит к тому, что, хотя время решения отдельной конкретной задачи, быть может, оказывается не самым коротким, суммарная производительность системы оказывается максимальной. При этом обеспечивается возможность оперативного участия человека в процессе решения задачи и регулярного диалога «человек-машина» без необходимости простоев процессора и потери времени при медленных действиях оператора за пультом.

Глава VIII

ПРИМЕНЕНИЯ ЭЛЕКТРОННЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

§ 1. ГДЕ ПРИМЕНЯЮТСЯ ЭЛЕКТРОННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ

В 1973 году сотрудники американского журнала «Computers and Automation» насчитали 2500 областей применения цифровых вычислительных машин. За прошедшие с того времени годы эта цифра возросла. Поэтому мы не можем привести исчерпывающих описаний практического применения вычислительных машин. Более того, даже попытки только классифицировать области применения приводят к неудаче, так как границы этих областей весьма расплывчаты. Все же с некоторыми допусками и оговорками можно выделить три основных направления использования цифровых вычислительных машин.

I. Научно-технические расчеты. Это исторически первая область применения цифровых вычислительных машин, которой они, собственно, обязаны своим рождением. Здесь вычислительные машины используются «по своему прямому назначению». Первые машины расходовали не менее 90% времени на задачи именно такого рода. Типичным для этой области применения машин является относительно небольшой объем входной и выходной информации, в большинстве своем сложные алгоритмы ее обработки и, следовательно, очень большое количество вычислительных операций, которые требуются в процессе решения каждой задачи.

II. Обработка данных. Работа вычислительных машин в этой области является для них, в известном смысле, «смежной специальностью», хотя и не слишком далекой от основной. Задачи этой области связаны с вводом и запоминанием очень большого количества информации, весьма простыми алгоритмами ее переработки, с небольшим числом операций и выводом опять-таки большого объема данных. Таковы задачи планирования, экономических расчетов и статистической обработки эмпирических данных. Еще недавно считалось, что для обработки данных следует создавать специализированные *информационно-логические машины* с большой памятью и небольшим быстродействием. Однако в третьем и последующих поколениях различия стерлись; современные цифровые машины поистине универсальны и могут с равным успехом

использоваться как для решения научно-технических задач и применения численных методов, так и для автоматической обработки данных.

III. Управление. Эта сфера применения вычислительных машин находится еще дальше от «собственно вычислений» и особенно бурно развивается в последние годы. Можно выделить три уровня управления, на которых используются вычислительные машины: *автоматическое управление технологическими процессами* (АСУТП), *автоматизированное управление производством* (АСУП) и *автоматизированные системы управления* (АСУ) более высокими единицами. Для этой цели созданы специализированные вычислительные машины, получившие название *управляющих*. Можно назвать и еще одну область управления, имеющую большое значение: управление научным экспериментом и автоматизация научных исследований.

В этой условной классификации сфер деятельности цифровых вычислительных машин можно указать еще одну область, наиболее далекую от расчетов и вычислений, но заманчивую по перспективам и популярную среди фантастов. Мы имеем в виду моделирование с помощью цифровых вычислительных машин тех или иных функций человеческого мозга, высшей нервной деятельности. Задачи такого рода объединяются общим наименованием *эвристического программирования* (в зарубежной литературе часто встречается термин *искусственный интеллект*, применяемый для той же цели), и, надо сказать, реальные достижения в этой области пока невелики.

В следующих параграфах рассмотрим конкретные примеры различных применений вычислительных машин, при этом, во-первых, постараемся выяснить, почему потребовалось привлечение вычислительных машин в той или иной области, для решения той или иной задачи, что вызвало необходимость их применения, и, во-вторых, как именно они применяются.

§ 2. ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ В НАУЧНО-ТЕХНИЧЕСКИХ РАСЧЕТАХ

Возникновение вычислительных машин связано с необходимостью расчетов, основной источник которых — расчеты научные и инженерно-технические. В число операций системы команд вычислительных машин входят арифметические и логические операции. Между тем для выполнения требуемых расчетов нужно справляться и с более сложными: дифференцированием, интегрированием, решением дифференциальных уравнений и т. п. Для их выполнения на цифровых вычислительных машинах они должны быть сведены к последовательности арифметических операций. Это делается с помощью методов *вычислительной математики*, иначе — *численных методов*. Прежде их называли *приближенными методами*, так как они действительно нередко носят приближенный характер.

Методы вычислительной математики позволяют свести к последовательности арифметических и логических операций, которые выполняются электронными вычислительными машинами, решение всех доступных в настоящее время задач. Многие из этих методов известны уже не одну сотню лет, некоторые предложены совсем недавно. Критерием качества этих методов являются теперь не только точность метода и количество требуемых действий, но также и удобство программной реализации.

В качестве одной из первых конкретных научных задач, для решения которых были применены цифровые вычислительные машины, можно назвать классическую *проблему N тел*. Сформулируем ее в таком виде: в пустом пространстве задано N материальных точек с известными массами $m_1, m_2, \ldots, m_N$, сила попарного взаимодействия которых определяется законом тяготения Ньютона

$$F_{ij} = \gamma \frac{m_i m_j}{r_{ij}^2};$$

заданы начальные координаты и начальные скорости всех материальных точек, требуется определить их дальнейшие движения.

Легко догадаться, что решение этой задачи имеет большое значение для астрономии, где оно определяет, например, орбиты планет Солнечной системы, и для космонавтики, где речь идет о расчетах траектории искусственных спутников или межпланетных космических станций. Так как движение каждой точки определяется тремя координатами и тремя компонентами вектора скорости (в трехмерном пространстве), то задача N тел приводится к решению системы $6N$ дифференциальных уравнений.

Для $N=2$ *(задача двух тел)* эта задача была полностью решена Ньютоном, но уже для $N=3$ *(задача трех тел)* ее аналитическое решение в конечном виде для общего случая получить не удалось, хотя над этим работали такие математики, как Л. Эйлер, Ж. Лагранж, К. Якоби и А. Пуанкаре. Между тем уже для расчета траектории искусственного спутника Земли приходится рассматривать задачу четырех тел ($N=4$), так как необходимо учитывать действие на спутник сил тяготения Земли, Луны и Солнца. Численные методы решения системы дифференциальных уравнений требуют около 10 млн. операций, что немыслимо для ручных расчетов и не слишком сложно даже для машин первого поколения. Такие расчеты проводятся сейчас систематически.

§ 3. ОБРАБОТКА ДАННЫХ И ИНФОРМАЦИОННО-ПОИСКОВЫЕ СИСТЕМЫ

«Обработка данных» — термин довольно неопределенный, и прежде чем говорить об обработке данных, следует уточнить, о чем именно будет идти речь. Эти слова употребляются в различных смыслах не только разными авторами, даже в пределах настоящего параграфа они означают различные задачи.

Первой из таких задач можно считать *статистическую обработку экспериментальных данных*. Это по существу задача научно-технических расчетов, разве что объем входных массивов может оказаться несколько большим, нежели в обычных научно-технических задачах. При элементарной обработке от массива требуются лишь простейшие статистические характеристики — *эмпирическая средняя* и *эмпирическая дисперсия*.

Более сложная статистическая обработка предполагает вычисление коэффициентов корреляции, разнообразных доверительных вероятностей и т. п. Все такие расчеты сводятся к вычислениям по известным формулам, хотя здесь и приходится иметь дело с большими массивами, которые иногда не помещаются непосредственно в оперативной памяти. В таких случаях организуют обработку их по частям, помещая сначала полный массив во внешней памяти.

Другим типом обработки данных является обработка учетных и плановых сведений. Объем такого рода работы чрезвычайно велик. Укажем в качестве примера, что средний по величине машиностроительный завод с числом работающих 4—5 тыс. человек изготовляет до 20 тыс. наименований деталей. При этом приходится учитывать около 150 млн. учетно-плановых показателей в год, совершая 320—350 млн. учетных операций. Такой объем работы требует около 100 человек счетных работников.

Уменьшения объема вычислительной работы можно добиться за счет укрупнения показателей. Например, многие машиностроительные предприятия вводят показатель потребляемой продукции — «прокат», заменяя этой строкой до 20 различных сортов и профилей проката, и считают стоимостью тонны проката некоторую среднюю величину, хотя стоимости различных сортов и профилей могут различаться в 3—5 раз. Такое укрупнение, разумеется, заметно упрощает отчетность, но сильно искажает учетные показатели и приводит к большим неувязкам в снабжении и сбыте. Впрочем, и укрупнение лишь ненамного упрощает дело. Настоящее упрощение и облегчение учетно-плановых расчетов возможно только в случае полной передачи их вычислительным машинам.

Задачи учетно-плановых расчетов характеризуются большим объемом входной и выходной информации и небольшой ее переработкой. Поэтому при решении таких задач особенно большую роль играют различные периферийные устройства и совсем мало загружен центральный процессор. Основной проблемой в области программирования таких задач является правильное и рациональное расположение соответствующих массивов во внешней памяти вычислительной системы. Таким образом, и в этой области обработки данных принципиально новых математических задач не возникает.

Настоящие проблемы обработки данных, приводящие к новым математическим проблемам, возникают тогда, когда речь идет о хранении и выдаче информации без ее числовой переработки, когда требуется только перестановка, сортировка или выборка по признакам или упорядочение. Такого рода задачи характерны, как прави-

ло, не для числовой, а для символьной информации. Они же встречаются и в информационно-поисковых системах, которые мы и объединим поэтому с задачами обработки данных.

Типичная задача обработки данных в этом смысле состоит в следующем. Один или несколько массивов, записанных на одной или нескольких магнитных лентах, состоят из *записей*; каждой из записей ставится в соответствие *ключ*, являющийся ее идентификатором. В множестве ключей имеется естественный порядок — например, это могут быть натуральные числа, которые могут располагаться по возрастанию или по убыванию, или слова обычного языка, которые могут быть расположены по алфавиту. Требуется расположить записи в такой последовательности, чтобы ключи были упорядочены. Эту задачу называют *сортировкой по ключам*.

Существует большое количество различных алгоритмов сортировки, требующих различного количества операций и разного объема памяти.

Некоторые методы требуют меньше операций, но больше памяти, иные менее требовательны к памяти, но требуют большого числа действий и, следовательно, медленнее. Поэтому выбор метода сортировки для той или иной конкретной задачи не может быть сделан раз и навсегда, он зависит от конкретных условий. Методы обработки и сортировки данных находят широкое применение в задачах машинного хранения и поиска информации и информационно-поисковых системах. Однако, прежде чем рассматривать эти применения, нам нужно предварительно подробнее познакомиться с характером этих задач, причинами их возникновения и постановками.

Активное развитие современной науки и техники сопровождается резким увеличением объема научной и технической информации, получившим название информационного взрыва. По подсчетам библиографов, половина книг, выпущенных за все время с момента изобретения книгопечатания, издана за последние 25 лет. Еще выше темпы роста периодической печати: за двести лет, с 1750 по 1950 годы, население Земли возросло в три раза, а число научных журналов — в 10 тысяч раз.

Ежегодный объем «информационного сырья» составляет сейчас около 8—10 миллиардов печатных страниц в год, удваиваясь каждые 10—15 лет. Общее число описаний изобретений и патентов доходит до полутора десятков миллионов. Все это поставило задачу автоматизации процессов хранения и поиска информации, разработки методов и технических средств для этой цели. Системы, обеспечивающие выполнение указанного комплекса работ, и получили название *информационно-поисковых систем*.

Блок-схему общей информационно-поисковой системы можно представить себе в виде, показанном на рисунке 107. Хранение и поиск информации выполняются с помощью цифровых вычислительных машин. Аналитико-синтетическая переработка исходной информации необходима для приведения ее к стандартному виду, чтобы облегчить поиск информации и сверку ее с запросом. Поэтому

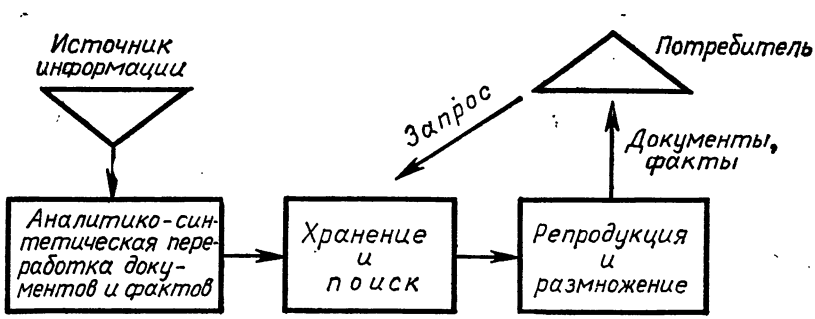


Рис. 107

(и по ряду других причин) в информационно-поисковых системах используются искусственные *информационно-поисковые языки дескрипторного типа.*

В каждом запросе, сделанном на естественном языке, можно выделить *ключевые слова* или словосочетания, отражающие суть вопроса. Например, в запросе «Какая фирма выпускает дисплеи?» ключевыми словами являются «фирма», «дисплей» и «перечислить», хотя в данной форме запроса последнее слово вообще не встречается. Иногда роль ключевых выполняют не отдельные слова, а устойчивые словосочетания, например «что известно?» или «память на ферритовых сердечниках». Такие ключевые слова, или ключевые устойчивые словосочетания, и называются *дескрипторами.*

Хранящиеся в системе документы кодируются набором дескрипторов. Запрос на поиск требуемой информации также формулируется в виде набора дескрипторов. Так как на естественном языке существует большая неоднозначность, то для используемого в данной информационно-поисковой системе дескрипторного языка необходимо иметь словарь, с помощью которого можно любой термин отнести к определенному дескриптору данного языка. Например, слова *назовите*, *перечислите*, *вывести*, *напечатать*, *сообщить*, *выдать* и т. п. в запросе следует заменять дескриптором *перечислить*. Такие словари-справочники принято называть *тезаурусами* (от греческого τεζαρωσ — клад, кладовая, сокровище).

Запрос информационно-поисковой системе формулируется в виде логической функции от дескрипторов. Например, запрос $r_1 \wedge r_2$ означает поиск документов, содержащих оба дескриптора — *и* дескриптор r_1, *и* дескриптор r_2; запрос $r_1 \vee r_2$ означает поиск документов, содержащих хотя бы один из упомянутых дескрипторов — *или* r_1, *или* r_2; возможны и любые более сложные комбинации, скажем, $r_1 \wedge r_2 \vee r_3$; здесь требуются документы, содержащие или пару дескрипторов r_1 и r_2, или дескриптор r_3.

Цифровые вычислительные машины, применяемые в информационно-поисковых системах, должны удовлетворять ряду специфических требований. Среди них в первую очередь следует назвать большую емкость запоминающего устройства и удобство ввода и вы-

вода информации. Немалое значение имеет и быстродействие; при этом необходимо обеспечить быстрое выполнение самых простых — логических — операций. Арифметические операции, которые в таких программах тоже встречаются, могут выполняться сравнительно медленно, так как их здесь бывает очень немного. Для удобства прямого диалога оператора с машиной требуется наличие системы графического взаимодействия и возможность работы машины в режиме с разделением времени. Наконец, особые требования предъявляются к печатающим устройствам, так как нужно иметь возможность дальнейшего воспроизводства и тиражирования полученных материалов.

Устройством ввода в таких системах обычно является телетайп или дисплей. В информационно-поисковых системах на устройства ввода падает особенно большая нагрузка, поэтому здесь особенно необходимы читающие устройства, позволяющие непосредственно читать типографский или машинописный текст, о которых шла речь в предыдущей главе. Сейчас имеется уже большое число моделей таких устройств. Из читающих устройств, сконструированных в Советском Союзе, упомянем РУТА-70, работающее со скоростью 50 знаков в секунду.

Многие информационно-поисковые системы, разработанные и разрабатываемые как в Советском Союзе, так и за рубежом, находятся в состоянии экспериментирования или пробной эксплуатации. Расскажем в нескольких словах о двух системах, разработанных в Центральном научно-исследовательском институте патентной информации и технико-экономических исследований, находящихся в эксплуатации в течение последних нескольких лет.

Одна из этих систем занимается классификацией изобретений по отраслям народного хозяйства. Сведения о выдаче авторских свидетельств на изобретения публикуются в «Бюллетене изобретений и товарных знаков» в порядке поступления заявок и их рассмотрения. Поэтому изобретение по электронике может находиться между новым типом чулочно-вязальной машины и новым рецептом колбасного фарша. Информационно-поисковая система по некоторому заданному перечню понятий может выдавать номера всех авторских свидетельств, в наименовании которых эти понятия присутствуют. Таким образом можно получить список изобретений, относящихся к данной интересующей потребителя области.

Другая система позволяет уже более глубоко анализировать авторские свидетельства в сравнительно узкой области — металлических сплавов. В системе хранятся все сведения об изобретенных сплавах черных и цветных металлов; о каждом сплаве записано: какие металлы в него входят, процентное содержание каждого из них и основные физико-химические свойства каждого сплава.

При поступлении новой заявки с помощью системы легко проверить, известны ли уже такие или близкие по составу сплавы и известны ли те физико-химические свойства этих сплавов, о которых говорится в новой заявке. Это позволяет эксперту судить о новизне

поданной заявки на изобретение нового сплава. Если заявка оказывается новой, то соответствующие сведения включаются затем в информационную систему, так что она все время пополняется.

Методы сортировки данных, о которых шла речь в начале настоящего параграфа, совершенно необходимы при переработке исходной информации для ее размещения, которое может обеспечить достаточно быстрый поиск. Например, если речь идет о сплавах, то естественно размещать их не в порядке поступления сведений, так как в этом случае поиск может оказаться слишком долгим, а, скажем, в порядке возрастания содержания той или иной легирующей примеси.

§ 4. ЦИФРОВЫЕ МАШИНЫ В ГУМАНИТАРНЫХ НАУКАХ

Основные применения цифровых вычислительных машин в гуманитарных науках связаны со стремлением получить к о л и ч е с т в е н н ы е о ц е н к и, чаще всего — с т а т и с т и ч е с к и е. Во многих науках приходится встречаться с массовыми явлениями, которые в той или иной мере доступны статистической обработке. Трудоемкость и неизбежность большого числа ошибок при ручной обработке ограничивали количество работ в этих направлениях и только благодаря использованию цифровых вычислительных машин эти направления получили достаточно широкое развитие.

Мы не будем пытаться очертить круг вопросов, решаемых в гуманитарных науках с помощью вычислительных машин: область их применения активно расширяется и какие бы то ни было итоги подводить рано. Ограничимся рассмотрением некоторых конкретных задач гуманитарных наук, которые были решены с помощью вычислительных машин. Начнем с задач литературоведения.

Важнейшей частью литературоведческого исследования считается составление *словаря автора*, в котором фиксируются употреблявшиеся им слова и их частота. Для этого, как и для многих других лексикографических исследований, необходимо составлять алфавитные списки слов или фраз, встречающихся в исследуемом произведении; такие списки называют *конкордансами*. Ручное составление конкордансов представляет собой огромную работу, которая выполнялась ранее, но в весьма ограниченном объеме. Составление конкордансов на электронной вычислительной машине использует алгоритмы сортировки, рассмотренные нами в предыдущем параграфе.

Статистическая обработка конкордансов позволяет составить словарь отдельного произведения или словарь автора, о котором мы упоминали. *Парный конкорданс* позволяет ответить на важный для литературоведения вопрос о влиянии творчества одного автора на творчество другого. Субъективные высказывания на эту тему составляют заметную часть литературно-критических работ. Рассмот-

рение и сравнение конкордансов двух авторов позволяет использовать объективные критерии.

Американский филолог Джозеф Рабен, изучая влияние великого английского поэта XVII века Джона Мильтона на английского поэта XIX века Перси Биши Шелли, составил конкордансы одной из поэм Мильтона и поэмы Шелли. Затем из обоих конкордансов отбирались те предложения, которые содержали о б щ и е с л о в а. Отобранные предложения сортировались в зависимости от числа совпадающих слов (наибольшее число совпадающих слов оказалось равным 17) и печатались. Последующая статистическая обработка этого парного конкорданса позволила Рабену сделать весьма интересные и, по-видимому, объективные выводы о влиянии Мильтона на творчество Шелли.

Такое же исследование было произведено для выявления цитирования древнеримских авторов Лукреция, Цицерона, Вергилия в трудах первых христианских теологов Августина, Иеронима и Тертуллиана. Было обнаружено большое число ранее неопознанных цитат и заимствований.

Еще более сложное по объему и целям машинное исследование предпринял другой американский литературовед — Карл Крёбер. Он поставил своей целью выяснить стилистические характеристики художественной прозы пяти английских писателей XIX века и установить образцы стилистического взаимоотношения между ними. Крёбер использовал в ы б о р о ч н у ю обработку исходного материала, рассматривая три типа выборок: *блочные выборки* — все предложения с последних 20 страниц романов; *случайные выборки* — по 5 последовательных предложений из произвольно выбранных страниц, не вошедших в блочные выборки, и *специальные выборки*, варьируемые по величине и характеру и отбираемые по специальным признакам (например, начала глав, речь одного героя и т. п.). Образец полученных Крёбером результатов приведен в таблице:

	Д. Остин «Убеждение»	Д. Эллист «Середина марта»	В. Вульф «Комната Якова»
Средняя длина предложения, в том числе:	25	25	16
повествование	27	36	16
диалог	15	14	8
«выравнивание»	0,37	0,31	0,36
«отклонение»	19	16	13
2 или 3 «коротких»	49%	37%	50%
2 или 3 «средних»	22%	33%	23%
2 или 3 «длинных»	13%	18%	12%
Соотношение подчиненных/главных предложений:			
повествование	1,00	0,86	0,36
диалог	0,70	0,51	0,42
Сочинительные и коррелятивные союзы	66%	57%	50%
Союзные наречия	17%	9%	10%

Интересные исследования русского стиха статистическими методами проводит группа сотрудников лаборатории статистических методов Московского университета под руководством акад. А. Н. Колмогорова.

В 1973 году группа сотрудников института прикладной математики и кибернетики в г. Горьком составила полный словарь М. Ю. Лермонтова. При этом был не только составлен конкорданс,

но и выписаны все различные словоформы, которых оказалось 53 тысячи!

Значительный интерес представляет определение с помощью машинного статистического анализа принадлежности анонимного текста тому или иному автору, давшее весьма эффективные результаты. Несколько работ в этом «детективном» направлении основываются на гипотезе шотландского священника Э. Мортона о присущих каждому автору характеристиках литературной речи. В частности, такой характеристикой, по мнению Мортона, является частота употребления некоторых ключевых слов.

Для проверки этой гипотезы Э. Мортон и Дж. Макгрегор, профессор университета в Глазго, исследовали с помощью вычислительной машины 400 отрывков (600 тысяч слов) из произведений древнегреческих писателей. В качестве ключевых слов были выбраны союзы και и δε, означающие примерно «и» и «но». Гипотеза блестяще подтвердилась и в этом случае, и при исследовании м н о г и х произведений о д н о г о автора.

Гипотеза Мортона была использована Мортоном и Макгрегором при анализе религиозных текстов. В частности, они установили, что из помещенных в «Новом Завете» 14 посланий, приписываемых апостолу Павлу, только 5 принадлежат одному автору, остальные 9 принадлежат по крайней мере еще 5 другим авторам.

Известные американские статистики Ф. Мостеллер и Д. Уоллес провели на базе этой же гипотезы исследование авторства некоторых федералистских * статей, написанных во времена Гражданской войны в США. Было известно, что авторами их могли быть видные деятели того времени А. Гамильтон или Г. Мэдисон. Мостеллер и Уоллес проанализировали частоту появления предлога *upon* (на, в) на 1000 слов текста в исследуемых статьях, а также в статьях Гамильтона и Мэдисона. Было замечено, что *upon* относительно часто появляется в статьях Гамильтона и редко — в статьях Мэдисона и так же редко в статьях анонимного автора. На этом основании исследуемые статьи были приписаны Мэдисону.

Во многом схожи с рассмотренными выше филологическими задачами проблемы музыковедения. Решение многих вопросов начинается здесь также с составления своеобразного конкорданса, который называется в музыке *тематическим указателем*: это список начальных музыкальных тем всех сочинений отдельного композитора или группы, школы. Элемент указателя содержит обычно от 7 до 12 нот. Тематический указатель используется, например, для анализа принадлежности анонимных музыкальных произведений таким же образом, как конкорданс для анализа литературных.

Профессор Жан Леру, анализируя таким образом несколько тысяч симфоний XVIII века, выяснил, что многие анонимные симфонии, ранее приписываемые Гайдну, на самом деле принадлежат менее известным композиторам гайдновской школы, как, например, Францу Покорному. Другой музыковед — Г. Б. Линкольн — аналогичным образом исследовал в итальянских архивах анонимные музыкальные рукописи XVI века. Линкольн разработал очень подробную методику составления тематических указателей с помощью вычислительной ма-

* Т. е. защищавших федеральное устройство США.

шины, с учетом большого числа музыкальных факторов, и методику их статистического анализа, создав по существу язык программирования для музыковедческого анализа.

Роль статистических методов в истории, социологии и других общественно-политических науках достаточно хорошо известна. Собственно говоря, сам термин *статистика* возник впервые как *государствоведение*. Поэтому не удивительно, что вычислительные машины начали широко применяться в историко-социологических исследованиях для обработки различных статистических материалов. В качестве примера таких исследований укажем изучение истории горной промышленности и металлургии в Западной Сибири в XVIII—XIX столетиях.

Новосибирские историки обработали на вычислительных машинах формуляры мастеровых Салаирских рудников, Гурьевского и Гавриловского металлургических заводов, относящиеся к концу XVIII и первой половине XIX столетия. Из них были выписаны и вводились в машину следующие сведения:

Фамилия, имя и отчество;
Возраст;
Социальное происхождение;
Служба;
Грамотность;
Штрафы;
Семейное положение;
Число детей.

Статистическая обработка этих данных дала обширный и убедительный материал для установления особенностей формирования и использования рабочих кадров в Западной Сибири в конкретный исторический период.

Применения статистических методов и вычислительных машин в антропологии и в археологии аналогичны описанным. В археологических исследованиях вычислительные машины использовались не только для классификации найденных при раскопках черепков, но и для такой задачи, как реконструкция целых предметов из фрагментов. Имея 2600 черепков, каждый из которых имел 50 признаков, совершенно невозможно отобрать те из них, которые могли принадлежать одному предмету, если не пользоваться техническими средствами. Для вычислительной машины же исследование матрицы размером 2600×50 совсем не представляется чересчур сложной задачей.

Эффективным результатом применения статистических методов в археологии следует считать *разгадку тайны Стоунхенджа*. Стоунхендж представляет собой один из крупнейших памятников конца каменного — начала бронзового века, относящийся к 1900—1600 годам до нашей эры. Остатки этой постройки находятся на равнине Солсбери в графстве Уилтшир в Англии, неподалеку от места действия знаменитой «Собаки Баскервилей» Конан-Дойля. Это постройка в форме кольца диаметром 29 *м*, состоящая из вертикально

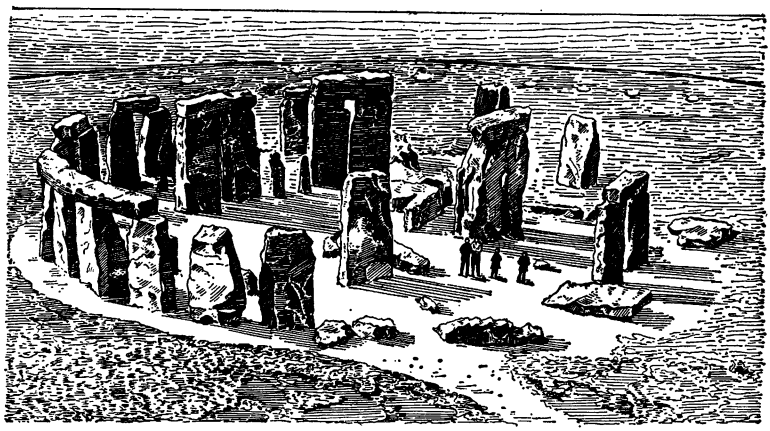


Рис. 108

врытых в землю каменных столбов (рис. 108) высотой в три человеческих роста. Каменное кольцо окружено круглым рвом, снаружи которого насыпан земляной вал, а внутри — вал из толченого мела.

Частично сохранившийся Стоунхендж систематически обследовался историками, археологами, антропологами, геологами, строителями и учеными других специальностей, но попытки объяснить назначение этого древнего памятника долгое время терпели неудачу. Среди различных гипотез, высказывавшихся по этому поводу, было и предположение, что Стоунхендж представляет собой древнюю астрономическую обсерваторию, но эта гипотеза была не более обоснована, чем многие другие.

Обоснование этой гипотезы было проведено Джеральдом Хокинсом, английским астрономом, в то время профессором Бостонского университета и сотрудником Смитсоновской астрофизической обсерватории в США. Вот как об этом пишет он сам:

«. . . Затем перешли к вычислительной машине. Ей сообщили географические сведения . . . и дали команду выполнить три операции:

1) провести прямые через 120 пар нанесенных на карту точек (некоторые пары — например соседние точки — были сочтены бесполезными для этой цели);

2) определить направления (азимуты) этих линий;

3) определить склонения точек на небесной сфере, в которые «упираются» эти линии . . .

Для ясности можно сказать так: машину как бы просили встать в каждую намеченную точку, посмотреть в направлении всех остальных точек на горизонт и сказать, какую точку на небе — с каким склонением — она видит . . .

Мы сразу же заметили, что среди значений склонений, выданных машиной, было много одинаковых. Часто попадались числа,

близкие к +29°, +24° и +19° (плюс означает северное склонение), и их южные двойники: —29°,—24° и —19°. Мы решили выяснить, какие небесные тела имеют такие склонения.

Сначала мы проверили планеты . . . Затем мы пробежались по звездам (лихо звучит!) . . . никакой связи со звездами не обнаружилось . . . Мы решили проверить наиболее очевидные небесные объекты, которые в доисторические времена считались божествами,— Солнце и Луну.

На этот раз результаты были поразительные. Склонения, вычисленные машиной, настойчиво и строго повторяли крайние положения Солнца (что я ожидал) и Луны (чего я не ожидал). Пара за парой точки Стоунхенджа, по-видимому, указывали на крайние склонения двух самых ярких небесных светил.

Я говорю «по-видимому», потому что в то время мы пользовались предварительной, поисковой программой, дававшей малую точность . . . Чтобы подтвердить это совпадение, нам нужны были точные значения крайних склонений Солнца и Луны в 1500 г. до н.э.

Конечно, пришлось снова прибегнуть к помощи машины. Мы задали ей современные крайние склонения Солнца и Луны и скорость их изменения и приказали ей определить крайние склонения в 1500 г. до н. э. Одновременно машина должна была рассчитать направления точек восхода и захода Солнца и Луны . . .

Мы тщательно сравнили все числа. Сомнений не было. Основные и часто повторяющиеся направления Стоунхенджа указывали на Солнце и Луну.

Как я уже говорил, я ожидал некоторой корреляции, но я не был подготовлен к полному совпадению результатов для Солнца, и для меня было большой неожиданностью существование почти полного совпадения для Луны. Ибо расчеты машины показали, что:

1) со средней точностью выше 1° двенадцать важных направлений Стоунхенджа указывали на крайние положения Солнца и

2) со средней точностью около 1,5° двенадцать направлений указывали на крайние положения Луны» (см.: Дж. Хокинс, Дж. Уайт. Разгадка тайны Стоунхенджа. М., Мир., 1973, с. 142—146).

Следующим примером успешного применения вычислительных машин, на котором мы хотим остановиться, является расшифровка рукописей майя, выполненная сотрудниками математического института Сибирского отделения Академии наук СССР Э. В. Евреиновым, Ю. Г. Косаревым и В. А. Устиновым.

Письменность майя, живших на территории современных Гватемалы и Гондураса, была единственной письменностью коренных жителей Америки. Цивилизация майя была уничтожена испанскими конкистадорами во время завоевания Америки; до нас дошли только три рукописи, хранящиеся в Мадриде, Париже и Дрездене. Не так давно было установлено, что письменность майя является не алфавитной, а иероглифической — отдельный знак может означать звук, слог или целое понятие.

Язык майя того времени известен по майя-испанским словарям и по книге «Чилам Балам», в которых слова майя написаны старым испанским алфавитом, а звуки языка майя, не совпадающие с испанским, переданы сочетаниями букв. Кроме того, известен и современный язык майя, конечно, сильно изменившийся с того времени (400—500 лет назад).

В основу методов расшифровки были положены хорошо известные статистические соображения о частоте распределения букв в различных языках. Впервые вопросом о частоте распределения букв в различных языках занимался еще Чарльз Бэббедж, опубликовавший исследование английского и французского языка в 1829 году. Наиболее часто во всех языках встречаются гласные, однако порядок их оказывается различным. В английском языке чаще всего встречается буква *е*, затем *а*, *о*, *i*. В русском языке на первом месте (не считая промежутка между словами) идет буква *о* (частота 0,090), затем *е* (частота 0,072), а затем *а*, *и* (частота 0,062). При расшифровке алфавитной письменности на известном языке естественно следует отождествлять наиболее часто встречающийся символ с наиболее часто встречающейся буквой.

Но в данном случае речь шла не об алфавитной письменности, так как иероглифы майя могли изображать целые слоги. Простейшая гипотеза, которая была принята исследователями, заключалась в том, что один иероглиф изображает п а р у б у к в. Тогда необходимо подсчитать и сравнить с частотой иероглифов частоту пар букв.

Такие подсчеты тоже уже выполнялись. Для русского языка эта работа была проведена в 1913 году академиком Андреем Андреевичем Марковым в его статье «Пример статистического исследования над текстом «Евгения Онегина», иллюстрирующий связь испытаний в цепь», напечатанной в «Известиях Академии наук». Любопытно отметить, что это был первый пример применения предложенной А. А. Марковым схемы «событий, связанных в цепь», которая носит сейчас название «цепи Маркова» и представляет весьма важный математический аппарат современной физики.

Ясно, что для языка майя такие вычисления частот двухбуквенных сочетаний приходилось производить заново. Еще одна трудность, которая встретилась исследователям, состояла в специфичности текста. Существо этой трудности легко пояснить на примере. Наименьшей частотой в русском языке обладает буква *ф* — 0,002. Представим себе теперь, что расшифровываемый текст является отрывком из математического учебника, посвященным свойствам функций. Тогда слово *функция*, а значит, и буква *ф* будет встречаться в этом отрывке очень часто и частота буквы *ф* будет заметно отличаться от 0,002.

Частоты иероглифов, подсчитанные в расшифровываемой рукописи, сильно отличались от частот пар букв, подсчитанных по словарям и книге «Чилам Балам». Тогда, вместо того чтобы сравнивать отдельные иероглифы с отдельными парами, исследователи стали

сравнивать группы иероглифов с группами пар. Здесь обнаружилось очень хорошее совпадение.

Например, половина слов (частота 0,5) текста «Чилам Балам» начиналась с одной из 70 пар букв, а половина комплексов иероглифов в Мадридской и Дрезденской рукописях, с фотокопиями которых работали в Новосибирске,— с одного из 73 иероглифов. Следовательно, расшифровку этих 73 иероглифов следует искать среди данных 70 пар букв. Такие же сравнения производились по последним парам букв и по ряду других признаков. Таким путем был найден ключ к загадке рукописей майя с помощью современных цифровых вычислительных машин.

Заметим в заключение, что применение вычислительных машин в гуманитарных науках не только позволило решить ряд их актуальных задач и дало новое орудие для работы в этой области, но, что не менее важно, во многом изменило постановку этих задач, подход к их формулировкам и сами методы мышления в области этих наук. Вместо того чтобы углубляться в эту сторону вопроса, приведем высказывание уже упоминавшегося нами К. Крёбера:

«Я пришел в вычислительный центр Висконсинского университета и сказал: «Я хочу использовать Ваши машины для анализа стиля художественной прозы». Я получил немедленный ответ: «Well (хорошо, ладно), мы думаем, что сможем Вам помочь, но, прежде всего, скажите, что Вы понимаете под словом *стиль*». С момента нашего разговора прошло много месяцев, а я все пытался понять, что же я *в действительности* понимаю под *стилем* (курсив Крёбера). В вычислительном центре я осознал, как мало я знаю о своем собственном предмете, и вынужден был критически отнестись к положениям, которые я *опрометчиво* (курсив наш.— *Авторы)* использовал в течение многих лет».

§ 5. ПРОГРАММИРОВАННОЕ ОБУЧЕНИЕ И ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ

Программированное обучение возникло в результате стремления повысить эффективность обучения и попыток использовать для этой цели современные технические средства.

Идеальными условиями обучения являются такие, при которых на одного преподавателя приходится один ученик и оба они согласованно участвуют в процессе обучения. Такие условия, однако, давно уже практически недостижимы почти на любом уровне обучения, и вместо индивидуального обучения приходится иметь дело с групповым. В таком случае необходимо прибегать к о р г а н и з а ц и и учебного процесса.

Во всяком обучении с участием обучающего и обучаемых можно выделить следующие элементы: сообщение нового учебного материала, закрепление и практическое применение нового материала, контроль правильности и прочности его усвоения учащимися. При переходе от низших форм обучения к более высоким эти три элемента все

дальше удаляются друг от друга; если в начальных классах задачи на новое правило выполняются на том же уроке, а запоминание правила проверяется на следующем, то в вузе семинарские или лабораторные занятия по лекционному материалу могут быть через неделю, а то и две, а зачет или экзамен —один раз в конце семестра.

Текущий контроль успеваемости — коллоквиумы и контрольные работы в течение семестра — бесспорно, способствуют повышению качества обучения, но требуют затраты очень большого количества сил и времени обучающих и не всегда эффективны, особенно когда речь идет о проверке практических навыков. Программированное обучение и ставило первой своей целью облегчение и упорядочение контроля работы обучаемых, играющего роль обратной связи в процессе обучения.

Большинство специалистов в области программированного обучения считают начальным этапом развития этой области изобретение С. Пресси, описание которого было опубликовано в 1926 году в американском журнале «School and Society» («Школа и общество»). Дело в том, что в американских школах и колледжах экзамены в значительной мере были вытеснены различными тестами, в которых испытуемый должен был выбирать верный ответ из нескольких предложенных.

Одна из первых машин Пресси имела размеры пишущей машинки. В левой части ее имелось небольшое окошко, в котором появлялись вопросы и несколько предлагаемых ответов. Учащийся должен был, прочтя вопрос, нажать одну из находящихся в правой части кнопок, номер которой совпадает с номером правильного, по его мнению, ответа. Число правильных ответов регистрировалось специальным счетчиком, расположенным сзади. Преподаватель должен был заранее вставить карточки с вопросами и ответами и списать со счетчика число правильных ответов после окончания работы учащегося.

Дальнейшее усовершенствование позволило управлять подачей карточек так, чтобы очередная карточка появлялась в окошке только после правильного ответа на предыдущий вопрос. Автоматический счетчик регистрировал число попыток. Таким образом, учащийся сразу же узнавал о своих ошибках и получал правильный ответ. Такая контролирующая машина была и *обучающим устройством.*

Машину Пресси относят к предыстории программированного обучения. История его начинается с работ американского психолога Б. Ф. Скиннера и другого американца — доктора Н. Краудера, сформулировавших два основных направления в этой области: *линейное программированное обучение* и *разветвленное обучение.*

Линейное программированное обучение, предложенное Скиннером, предполагает разбиение изучаемого материала на маленькие порции, усвоение которых не представляет труда. Порции располагаются в линейной, строго фиксированной последовательности. Не делается никаких попыток дифференцировать изучение материа-

ла для различных учащихся. На каждом шагу от учащегося требуется какая-то активная ответная реакция. Чтобы помочь, учащемуся даются намеки, подсказки или указания и сразу сообщается, верен ли его ответ. Вопросы формулируются таким образом, чтобы по крайней мере девять раз из десяти ответ был правильным. Это дает учащимся удовлетворение своими успехами.

Разветвленное программированное обучение, предложенное Краудером, также предполагает разбиение материала на небольшие порции. Однако следующий материал, который будет предложен, определяется не заранее фиксированной последовательностью, а характером ответа на контрольный вопрос по предыдущему. Если ответ на предыдущий вопрос правилен, то учащемуся выдается следующая порция материала и следующий проверочный вопрос. Если же ответ неверен или содержит какие-либо неточности, то учащемуся даются дополнительные разъяснения и предлагаются новые вопросы по тому же материалу. Таким образом, каждый учащийся движется со всей скоростью и проходит обучение по своему пути.

Собственно говоря, программированное обучение в описанных выше формах, хотя оно и хорошо приспособлено для «машинизации», никак не связано с вычислительными машинами. Прежде всего, принципы и формы программированного обучения могут быть и, действительно, многократно были реализованы путем издания соответствующих *программированных учебников*, построенных, главным образом, по принципу разветвляющегося программирования Краудера. Да и различные *обучающие* и в особенности *экзаменующие машины*, изготовлявшиеся и изготовляющиеся в США, Англии, СССР и других странах, не представляют собой вычислительных машин.

Это, однако, не означает, что вычислительные машины не применяются в программированном обучении. Оба метода программированного обучения и обучающие машины, их реализующие, обеспечивают лишь весьма ограниченную индивидуализацию обучения. Для подлинной индивидуализации необходимо накопление и быстрая переработка большого числа данных, относящихся как к данному отрезку обучения, так и к личности обучающегося. А это уже входит в область возможностей современных цифровых вычислительных машин.

Цифровые машины могут классифицировать допущенные ошибки и выяснять их возможные причины, накапливать в памяти и перерабатывать любые данные о каждом обучающемся. Это позволит реализовать такое управление обучением каждого учащегося, при котором в максимальной степени учитываются его индивидуальные особенности.

Оптимистически настроенные сторонники «машинизации» обучения считают, что основные проблемы применения вычислительных машин в педагогическом процессе уже решены. По их мнению, школьный урок ближайшего будущего удачно изображен в стихотворении Л. Е. Карлтон (вольный перевод с английского И. М. Липкина):

Да, было, было так когда-то...
Уютно дребезжал звонок,

Словами: «Здравствуйте, ребята!» —
учитель начинал урок...

Иные дни теперь настали,
К познанью изменился путь,
Взять вилку надо нам вначале
И в сеть учителя воткнуть.

Уже сейчас универсальные цифровые вычислительные машины используются при проведении приемных экзаменов в вузы. Несколько лет приемные экзамены по математике и физике в Московском экономико-статистическом институте проводятся с помощью вычислительной машины Минск-22. Вот как организована первая из двух письменных работ по математике.

Заранее составленные свыше 2 тыс. задач сформулированы таким образом, чтобы их ответами были целые числа, и разбиты на 10 групп, каждая из которых относится к какому-либо разделу курса элементарной математики. Все эти задачи вместе с ответами записываются на магнитную ленту. Экзаменующийся получает листок, на котором с помощью случайной выборки указывается номер аудитории и места, и текст 20 задач — по две из каждой группы. Текст работы сдается экзаменатору, а полученные ответы выписываются на отдельном листке, по которому затем производится перфорация. Ответы вводятся в машину и автоматически сверяются с записанными ранее, в результате чего машина печатает оценку. В случае сомнений или апелляций выполненная работа проверяется преподавателями, однако число апелляций крайне незначительно.

Такая организация экзаменов обеспечивает полную объективность и единый уровень требований ко всем поступающим, в огромной мере снижает трудоемкость и увеличивает скорость проверки экзаменационных работ.

§ 6. ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ В МЕДИЦИНЕ

Первой причиной привлечения вычислительных машин к медицинской деятельности была *медицинская статистика*. Медицинская статистика требует обработки большого статистического материала из различных клиник, больниц и амбулаторий о числе и характере заболеваний, их причинах, массовости, о применении различных лекарств, физиотерапевтических процедур и хирургических операций. Характер переработки такой информации вполне походит на ту обработку данных, о которой уже шла речь в § 3 настоящей главы.

Следующим шагом использования машин является создание специализированных медицинских информационно-поисковых систем. Такая система может накапливать многочисленные данные о конкретных случаях протекания тех или иных заболеваний и выдавать

сведения не только о возможных симптомах, но и о возможных течениях болезни, возможных побочных действиях тех или иных лекарств и процедур и т. п., которые помогут врачу при постановке диагноза и при выборе способа лечения.

Наконец, возможно и фактически имеет место непосредственное применение цифровых вычислительных машин для диагностики, в особенности *дифференциальной диагностики*, когда требуется различить два различных, но схожих по внешним проявлениям и симптомам заболевания. На этом мы остановимся несколько позже.

Еще одна обязанность, которую можно возложить на вычислительные машины,— это непрерывный контроль за состоянием тяжело больных. Здесь вычислительная машина должна действовать по существу как *управляющая*. К ней подсоединяются *датчики* — приборы, следящие за функциями организма больного: частотой пульса, температурой, кровяным давлением, частотой и глубиной дыхания и т. п. При выходе какого-либо из этих параметров за заданные пределы машина может подать сигнал медперсоналу или принять какие-либо немедленные меры, например увеличить подачу кислорода. Мы не будем подробнее останавливаться на этой деятельности, так как она мало отличается от управления вообще; применению вычислительных машин в области управления будут посвящены следующие параграфы настоящей главы.

Остановимся несколько подробнее на задачах диагностики, являющихся центральными в специфически медицинских применениях цифровых вычислительных машин. Возможны весьма различные подходы к этой проблеме и разнообразные алгоритмы ее решения, основанные на разных принципах. Рассмотрим некоторые из них.

Разработанный в Институте кибернетики АН УССР алгоритм диагностики, предложенный Н. М. Амосовым и Е. А. Шкабарой, состоит в следующем. В оперативную память машины ввели 108 основных признаков ста сорока сердечно-сосудистых заболеваний. Наборы признаков объединили в комплексы симптомов, соответствующих каждому заболеванию, т. е. каждому диагнозу. Симптомы разбили на важнейшие и второстепенные и каждому из них поставили в соответствие «весовые коэффициенты» в соответствии с их значимостью. Кроме этого, в память машины ввели также «избыточные признаки», не входившие в комплекс симптомов, но могущие иметь место при указанных заболеваниях; каждому из них также соответствует определенная значимость.

Алгоритм постановки диагноза состоит в сравнении комплекса симптомов, имеющихся у данного больного, с комплексами симптомов, хранящихся в памяти машины, и оценки значимости совпадающих и несовпадающих признаков, отдельно по важнейшим и отдельно по второстепенным. По максимальным суммам значимостей выбирается пять наиболее вероятных диагнозов. Эти последние проверяются затем по избыточным признакам, и на основании этой проверки делается выбор окончательного диагноза.

Другой возможный подход к решению задач диагностики состоит в вероятностной ее постановке и оценке вероятностей различных диагнозов. При этом обычно пользуются так называемой *формулой вероятностей гипотез*, которую называют также *формулой Бейеса*.

Эта формула находит сейчас очень широкое применение в задачах обработки экспериментальных данных не только в медицинских исследованиях.

Дадим несколько требуемых пояснений из теории вероятностей, не останавливаясь на выводах приводимых формул и предполагая известным понятие вероятности события.

Предположим, что некоторое событие A может наступить вместе с одним из событий H_1, H_2, . . ., H_n, которые мы будем называть *случаями* или *гипотезами*. Пусть нам известны вероятности отдельных гипотез, причем одна из них непременно должна реализоваться, т. е.

$$P(H_1)+P(H_2)+\ldots+P(H_n)=1. \qquad (1)$$

Предположим, далее, что нам известна вероятность наступления события A при каждой из этих гипотез, т. е. условные вероятности $P(A/H_1)$, $P(A/H_2)$, . . ., $P(A/H_n)$, где $P(A/H_i)$ означает вероятность наступления события A в случае H_i, т. е. при условии реализации гипотезы H_i. Тогда вероятность наступления события A определится по формуле

$$P(A)=P(H_1)P(A/H_1)+P(H_2)P(A/H_2)+\ldots+P(H_n)P(A/H_n). \qquad (2)$$

Эта формула носит название *формулы полной вероятности*. Вероятности гипотез $P(H_1)$, $P(H_2)$, . . ., о которых шла речь выше, называют *априорными вероятностями* (вероятностями a priori, т. е. *до опыта*).

Чтобы лучше пояснить суть дела, сформулируем ту же задачу в конкретных терминах, нарочно в наиболее отвлеченной форме, в которой математическое содержание схемы выявляется особенно четко. Пусть дано несколько урн (ящиков), содержащих черные и белые шары в различных количествах. Мы выбираем какую-либо урну и извлекаем из нее один шар, который может быть или не быть белым. Событие A состоит в том, что вынутый шар окажется белым. Вероятность $P(H_i)$ каждой гипотезы есть вероятность выбора данной урны, а условная вероятность $P(A/H_i)$ — вероятность вынуть белый шар из урны с номером i. Эта вероятность определяется, очевидно, составом шаров в данной урне. Так, если в первой урне находится 7 белых и 3 черных шара, то $P(A/H_1)=0{,}7$.

Формула (2) определяет в этом случае вероятность того, что вынутый шар будет белым в сложном испытании, которое состоит в том, что сначала выбирается некоторая урна (*гипотеза*), а затем из нее вынимается шар.

Предположим теперь, что испытание произведено и событие A наступило, т. е. что вынутый шар оказался белым. Какова вероятность того, что он вынут из первой, второй, . . ., n-ой урны, т. е.

какова в этом случае *вероятность каждой из принятых ранее гипотез*?

Возьмем конкретные числовые данные. Пусть мы имеем три урны с составом шаров 9 белых и 1 черный в первой, 5 белых и 5 черных во второй, 10 черных в третьей, и каждая из этих трех урн может быть выбрана с одинаковой вероятностью. Тогда $P(H_1)=P(H_2)=P(H_3)=1/3$, $P(A/H_1)=0{,}9$, $P(A/H_2)=0{,}5$ и $P(A/H_3)=0$, а вероятность вынуть белый шар в силу формулы (2) будет равна

$$P(A)=0{,}9\times 1/3+0{,}5\times 1/3+0\times 1/3=7/15.$$

Если вынут белый шар, то вероятности выбора урн перестают быть одинаковыми: вероятность выбора третьей урны равна, очевидно, нулю, так как вынуть из нее белый шар не удастся: она содержит только черные шары. Интуитивно ясно, что вероятность того, что была выбрана первая урна, значительно больше, чем вероятность того, что была выбрана вторая.

Теперь возвратимся к общей постановке задачи. Предположим, что в результате произведенного испытания событие A наступило. Спрашивается, какова теперь вероятность реализации различных гипотез, т. е., например, какова вероятность того, что событие A наступило в результате реализации гипотезы H_1 или гипотезы H_2? ... Эти *условные вероятности гипотез* $P(H_1/A)$, $P(H_2/A)$, ..., $P(H_n/A)$, которыми мы сейчас интересуемся, в отличие от первоначальных априорных, носят название *апостериорных вероятностей* (вероятностей a posteriori, т. е. *после опыта*). Они находятся по формулам

$$P(H_i/A)=P(A/H_i)P(H_i)/P(A)=P(H_i)P(A/H_i)/\sum_{i=1}^{n}P(H_i)P(A/H_i) \quad (3)$$
$$(i=1,\ 2,\ \ldots,\ n),$$

которые и называются *формулами вероятностей гипотез* или *формулами Бейеса*.

Если вынут белый шар неизвестно из какой урны, то формула (3) даст, что вероятность гипотезы H_1 — выбора первой урны—равна при этом

$$P(H_1/A)=(0{,}9/3) : (7/15)=9/14,$$

вероятность H_2 — выбора второй урны

$$P(H_2/A)=(0{,}5/3) : (7/15)=5/14,$$

а вероятность выбора третьей урны равна нулю. Полученный результат соответствует интуитивным ожиданиям.

Таким образом, формула вероятностей гипотез, или формула Бейеса, позволяет определить вероятности различных гипотез после выполнения некоторого эксперимента, если известны вероятности получения данного результата эксперимента при различных гипотезах и предварительные вероятности различных гипотез до выполнения этого эксперимента.

Задачей диагностики является определение болезни по известным наблюдаемым симптомам. Трудность состоит в том, что одни и те же

симптомы возможны при различных болезнях, хотя и с разными вероятностями. Эти сведения мы можем получить из медицины. Можно получить также не только вероятность отдельного симптома, но и вероятность некоторого наблюдаемого у данного больного комплекса симптомов при каждой болезни, при которой такие симптомы могут наблюдаться. Например, затемнение на рентгеновском снимке легкого как вблизи от корня легкого, так и в его верхушке может наблюдаться и при раковой опухоли и при опухоли туберкулезного происхождения (туберкуломе), но при раке легкого — с большой вероятностью вблизи от корня и с малой в верхушке, а при туберкуломе — наоборот. Кроме того, из данных медицинской статистики мы можем считать известными вероятности различных болезней.

Назовем различные болезни различными гипотезами, а наблюдаемый комплекс симптомов — наступлением события. Известные из медицинской статистики вероятности отдельных болезней являются априорными вероятностями гипотез. По формулам Бейеса можно подсчитать апостериорные вероятности гипотез при условии наступления данного события, т. е. апостериорные вероятности различных болезней при условии наблюдения данного комплекса симптомов. Из подсчитанных вероятностей нетрудно выбрать наибольшую, которая и определит *наиболее вероятный диагноз*. Полученная вероятность покажет также, насколько вероятен этот диагноз.

Еще один подход к задачам диагностики связан с более общей постановкой задачи «распознавания образов», для которой диагностика, точнее, *дифференциальная диагностика*, является частным случаем. Этот подход будет рассмотрен в следующем параграфе.

§ 7. ЭВРИСТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Слова *эвристика, эвристические методы, эвристические соображения* имеют своим источником легендарное восклицание Архимеда «эврика!», настолько знаменитое, что вошло даже в энциклопедические словари. Эвристическими называют рассуждения, не являющиеся точными доказательствами, а основанные на наводящих соображениях, аналогиях, неполной индукции и т. п.

Программа для цифровой вычислительной машины воспроизводит некоторый алгоритм. Так как мы пока еще не знаем, *как* человек думает, то алгоритмы для решения «думательных» задач не могут быть вполне точными, а являются лишь эвристическими. Поэтому все применения вычислительных машин, в которых идет речь о «думательных» задачах, т. е. о моделировании функций человеческого мозга, принято относить к области, называемой эвристическим программированием. Эта область применений вычислительных машин особенно популярна у писателей-фантастов. Эвристическое программирование в свою очередь имеет несколько различных областей и направлений; на некоторых из них мы остановимся.

Можно полагать, что одним из первых применений вычислительных машин в задачах такого рода было их применение в задачах перевода с одного языка на другой. Дадим лишь приблизительное описание принципов работы такой программы. Но и по такому грубому приближению можно представить себе достигнутые результаты и имеющиеся трудности.

Представим себе два языка, состоящие из одинакового количества слов, которые не изменяются при склонении или спряжении и все являются значащими (в русском языке практически все слова являются значащими; напротив, в других европейских языках имеется ряд служебных слов, не имеющих самостоятельного значения, например артикли). Предположим, далее, что каждое слово одного языка имеет точно один перевод на другой язык, так что между словами этих двух гипотетических языков существует взаимно однозначное соответствие.

Если еще предположить, что порядок слов для обоих языков несуществен (примерно так и обстоит дело в русском языке; здесь слова в предложении можно переставлять достаточно произвольно в отличие от многих других языков, где порядок слов предписывается грамматической конструкцией и смысл написанной фразы от перестановки слов может измениться), то для таких двух языков написать программу перевода совсем просто.

Достаточно ввести в машину словарь. Это можно сделать, например, таким косвенным путем. Занумеруем слова одного языка каким-либо способом, например по алфавиту. Слова же второго языка занумеруем так, чтобы соответствующие друг другу слова двух языков имели один и тот же номер. Теперь остается сделать программу, с помощью которой машина могла бы по введенному слову находить его номер, затем по номеру — соответствующее слово второго языка и выдавать его на печать.

Фактически существующие языки, конечно, не предоставляют таких удобств для создания программы перевода. Здесь дело обстоит гораздо сложнее. Прежде всего, обычно слова являются изменяемыми и порядок слов в предложении не безразличен. Поэтому, кроме словаря программу необходимо снабдить грамматическими правилами обоих языков.

Работа программы перевода для реальных языков должна состоять из следующих этапов:

1) синтаксический анализ введенного предложения; необходимо определить подлежащее, сказуемое и второстепенные члены предложения и выяснить их характер;

2) морфологический анализ каждого слова: для каждого слова необходимо определить, к какой части речи оно относится, и для изменяемых слов найти их основную форму: неопределенную форму для глагола, именительный падеж единственного числа для существительного и т. п.;

3) перевод каждого слова на другой язык при помощи словаря;

4) расстановка слов другого языка в переведенном предложении в нужном порядке;

5) приведение каждого слова в нужную грамматическую форму (число, падеж или лицо и т. п., в зависимости от части речи).

Из этого перечисления видно, что для программы перевода необходимы довольно глубокие знания структуры языка, причем они должны быть изложены в форме точных алгоритмов, доступных для выполнения на вычислительной машине. Такими исследованиями занимается область языковедения, называющаяся *структурной лингвистикой*. Нужно отметить, что не все такие потребности переводящих программ структурная лингвистика может удовлетворить.

Впрочем, основные трудности машинного перевода находятся не здесь, а в словаре. До сих пор мы предполагали, что между словами двух языков существует взаимно однозначное соответствие. Как хорошо известно, на самом деле это далеко не так, и здесь заключены очень серьезные проблемы.

Прежде всего, в каждом языке имеются омонимы; так называются слова, имеющие несколько различных значений. Например, русское слово *коса* может иметь четыре различных значения: коса может быть острая, русая или песчаная или может быть кратким прилагательным (от *косая*). Ясно, что при встрече с омонимами перевод каждого слова в отдельности оказывается невозможным: при переводе каждого отдельного слова необходимо всякий раз обращаться к контексту. Да и словарь придется устраивать много сложнее, чем было описано выше.

Столь же большие трудности вызывает наличие синонимов, т. е. нескольких слов, имеющих одинаковое, или, точнее сказать, приблизительно одинаковое, значение, вроде «замечательно», «превосходно» и «великолепно». Чаще всего они взаимозаменяемы, но бывают случаи, когда из множества синонимов необходимо выбрать один, причем опять-таки этот выбор необходимо сообразовать с контекстом.

Описанные ситуации чрезвычайно затрудняют составление словаря и пользование им, так как каждому слову одного языка приходится сопоставлять не одно, а несколько слов другого. Свод грамматических правил, о которых мы говорили выше, нужно пополнять правилами выбора нужного перевода из нескольких, имеющихся в словаре, в зависимости от контекста. Такие правила трудно искать и еще труднее формулировать.

К этому можно добавить затруднения, связанные с переводом идиоматических выражений — некоторых устойчивых словосочетаний, где слова употребляются отнюдь не в прямом их смысле. Так, например, если по-русски о ком-то скажут, что он «собаку съел», то вряд ли в переводе этих слов на английский язык потребуется английское слово dog (собака). Точно так же английское red herring (буквально — *красная селедка*) в переводе на русский язык означает вовсе не селедку, а *утку* (в смысле — *ложь, ерунда*).

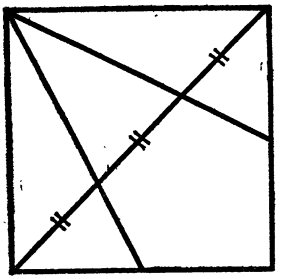


Рис. 109

Анализ слов по частям речи тоже может натолкнуться на неожиданные трудности. Например, никто из читателей, видимо, не усомнится в том, что «стекло» есть имя существительное среднего рода, стоящее в именительном падеже, . . . до тех пор, пока не встретится с фразой, где «стекло» является глаголом. Такой фразой может быть, например, следующая: «Масло разлилось по столу и ручейком стекло на пол». Конечно, в данном случае предварительный синтаксический анализ выяснит, что здесь «стекло» является сказуемым, но не во всех случаях так повезет.

Таким образом, проблем, требующих своего разрешения для создания полноценных программ перевода с одного языка на другой, еще достаточно много. Как метко выразился один из специалистов, работающих в этой области, «прежде всего необходимо научиться хорошо переводить с русского на русский». Однако это не значит, что в области перевода нет никаких достижений.

Одной из реальных идей упрощения задач перевода является сужение области рассматривающихся текстов. Если ограничиться текстами, относящимися к определенной узкой области, то для их перевода потребуется более простой словарь. В нем может быть меньше слов и меньше разнообразия для перевода каждого слова. Синтаксические конструкции также будут менее разнообразны, что облегчает работу и этого, и остальных блоков программы. Несколько таких «узких» программ перевода с французского и с английского языков на русский у нас имеется. Ведется работа по их усовершенствованию.

Другое направление работ в области эвристического программирования — машинное доказательство теорем, т. е. использование вычислительных машин для доказательства теорем, или, точнее говоря, вывода одних утверждений из других. При этом речь шла не только об уже известных утверждениях.

Из ряда имеющихся в этой области результатов отметим программу, составленную американским математиком Г. Гелернтером для доказательства геометрических теорем. Работая по этой программе, машина доказала теорему, справедливость которой ее автору ранее не была известна.

Эта теорема формулируется так. Из вершины квадрата, не лежащей на его диагонали, проведены два отрезка, соединяющие вершину с серединами противоположных сторон. Утверждается, что эти отрезки делят диагональ на три конгруэнтные части (рис. 109). Конечно, эта теорема не очень трудна, и, надо думать, читатели сумеют самостоятельно доказать ее. Но это доказательство уже не будет первым — впервые эту теорему доказала электронная вычислительная машина!

Аналогичные программы создавались и у нас. Например, в вычислительном центре Латвийского университета была создана программа для доказательства теорем арифметики, которая, в частности, умела пользоваться методом математической индукции.

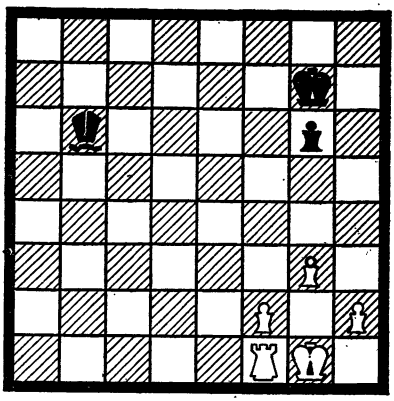


Рис. 110

Еще более интересны игровые программы, с помощью которых электронные вычислительные машины могут играть в карточные игры, домино, шашки и шахматы. Наиболее успешно справляются со своим делом программы игры в домино и в шашки. Программа игры в домино, составленная в Институте прикладной математики АН СССР, еще в конце пятидесятых годов, продемонстрировала очень успешную игру. Шашечная программа для 8-клеточных (русских) шашек, разработанная в 1966—1968 годах в Институте кибернетики АН УзССР в Ташкенте, выигрывала у очень сильных игроков и решила все предложенные ей этюды, в том числе и довольно сложные.

Наибольший интерес, конечно, представляют шахматные программы, на которых мы остановимся немного подробнее. О возможности создания программы для игры в шахматы говорил в свое время еще Чарльз Бэббедж. Реальные шахматные программы начали создаваться уже в конце пятидесятых годов, но играли они совсем слабо. В середине шестидесятых появились уже программы, могущие играть с человеком; сейчас лучшие шахматные программы можно, видимо, сравнить с игроками второго-третьего разряда и представляется возможным их дальнейшее улучшение.

Создание шахматной программы требует решения трех сложных и взаимосвязанных проблем:

1) как хранить в машине информацию о позиции и о ходе;

2) каким способом (по какому алгоритму) будет выбираться ход;

3) как реализовать выбранный алгоритм при данном способе записи информации.

Рассмотрим возможные решения этих проблем.

Шахматная доска имеет восемь горизонталей и восемь вертикалей, поэтому для нумерации полей доски удобно воспользоваться восьмеричной системой счисления. Поставим в соответствие каждому полю двузначное восьмеричное число, в котором первая цифра есть номер вертикали, а вторая — номер горизонтали, причем горизонтали и вертикали нумеруются восьмеричными цифрами от 0 до 7. При такой кодировке полю $a1$ будет соответствовать код 00, полю $b2$ — код 11 и полю $h8$ — код 77.

Занумеруем и все шахматные фигуры: дадим, например, пешке номер 1, коню — 2, слону — 3 и т. д.; кроме того, будем считать, что номера белых фигур положительны, а черных — отрицательны. Теперь понятно, как можно записать позицию. Отведем для ее записи ячейки, например, с номерами от 100 до 177 (так, чтобы две последние цифры адреса совпадали с номером поля) и в каждой ячейке

запишем номер фигуры, которая на этом поле стоит, или нуль, если поле свободно, например позиция (рис. 110):

Белые Кр *g*1, Л*f*1, п. п. *f*2, *g*3, *h*2, черные Кр*g*7, Ф*b*6, п. *g*6 изобразятся записями в ячейках:

115 : —5	162 : +1
150 : +4	165 : —1
151 : +1	166 : —6
160 : +6	171 : +1

В остальных ячейках будут записаны нули.

Ход представляется в машине тремя числами: номером фигуры и номерами начального и конечного полей ее движения. Например, ход белой пешки *e*2—*e*4 изобразится так:

+1 41 43,

а ход черного ферзя Ф*b*6—*d*4 так:

—5 15 33.

Легко «научить» машину делать ходы, проверяя их возможность, но очень трудно научить ее выбирать ход. Одним из возможных алгоритмов выбора может служить перебор вариантов. Однако полный перебор до «окончательной» позиции невозможен, ввиду необозримо большого числа возможных комбинаций. Поэтому приходится ограничивать и число рассматриваемых вариантов, и «глубину перебора».

Но если невозможно добираться до «окончательной» позиции, то нужно уметь оценивать промежуточные. В простейшем случае оценкой позиции (например, за белых) можно считать разность материальных ресурсов белых и черных на доске. Более тонкие оценки включают позиционные и даже тактические факторы (например, наличие сдвоенных пешек, возможность рокировки и т. п.).

Самая простая из стратегий перебора — перебор на фиксированную глубину, т. е. на определенное число ходов. После этого числа ходов полученную позицию следует оценить. Естественно, что программа тем лучше выбирает ход, чем больше глубина перебора, но увеличение глубины на 1 ход увеличивает время выбора хода приблизительно в 40 раз.

Мы хотим выбрать ход, приводящий к позиции с самой высокой оценкой, тогда как противник стремится прийти к позиции, в которой оценка будет возможно меньшей. Поэтому алгоритм выбора хода осуществляется процедурой, которую называют *минимаксной*. Рассмотрим эту процедуру на примере перебора вариантов с глубиной 1 ход.

Пусть из позиции A возможно несколько ходов, приводящих к позициям B_1, B_2, B_3, . . . (рис. 111). Если ходы противника из позиции B_1 приводят к позициям C_1, C_2, . . ., которые мы уже оцениваем, то надо считать, что противник выберет самый сильный, т. е. самый неблагоприятный для нас ход, и оценкой позиции B_1 следует считать м и н и м у м из оценок C_1, C_2, ... Так же определяются и оценки остальных позиций B_2, B_3, . . . Но теперь мы можем выбрать м а к с и м у м из полученных оценок позиций B_1, B_2, B_3, ... Это и будет наилучший ход, так как он обеспечит нам максимальную оценку при самом неблагоприятном для нас ответе противника.

В 1967 году состоялся первый шахматный матч машин СССР — США. От Соединенных Штатов Америки выступала программа, составленная математиками Стенфордского университета. Советский Союз представляла программа, разработанная в Институте теоретической и экспериментальной физики. Игрались четыре партии; в двух из них программа ИТЭФ играла с глубиной перебора в три полухода и в двух других — в пять полуходов. Обе первые

партии закончились вничью, обе вторые выиграла программа ИТЭФ, так что общий счет матча был 3 : 1. Модифицированная авторами в Институте проблем управления АН СССР, эта программа под именем *Каисса-72* играла в 1972 году две партии с читателями газеты «Комсомольская правда». Здесь противники оказались более серьезными. Счет матча 1,5 : 0,5 в пользу читателей: *Каисса* проиграла черными и свела вничью партию белыми.

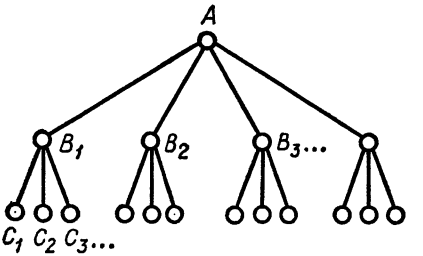


Рис. 111

Летом 1974 года *Каисса* выдержала еще одно испытание. В Стокгольме был организован первый международный турнир шахматных программ, в котором участвовали 13 программ из 8 стран мира. Программа *Каисса*, которую представлял в Стокгольме один из авторов М. В. Донской, выиграла все встречи и была провозглашена первым чемпионом мира по шахматам среди вычислительных машин.

Перейдем теперь еще к одной области эвристического программирования, о которой мы уже несколько раз упоминали,— к *задачам узнавания*. Здесь возможны различные постановки задачи и различные методы подхода к их решению; мы остановимся на некоторых.

Общую постановку задачи можно сформулировать, например, так. Имеется некоторое множество объектов, каждый из которых характеризуется определенным набором признаков. Каждый объект принадлежит к одному из нескольких классов, причем классы задаются только перечислением принадлежащих им объектов. Задается новый объект с известными признаками; задача программы состоит в том, чтобы определить, к какому классу относится этот объект.

Все дело в том, что нам (и программе) неизвестно, какие из имеющихся признаков информативны и насколько. Именно это в первую очередь и должна выяснить программа узнавания. Таким образом, работа программы узнавания состоит из двух частей — *обучения*, когда программа выясняет информативность имеющихся признаков при решении вопроса о том, к какому классу принадлежит объект, и *экзамена*, в котором решается, к какому классу отнести новый объект, и результат которого определяется качеством обучения.

Легко заметить, что частным случаем такой общей задачи являются и задача чтения рукописного текста, и задача дифференциальной диагностики, и задачи разведки полезных ископаемых, и многие другие. В терминах этой общей задачи узнавания можно сформулировать и некоторые игровые задачи. Например, шахматная

позиция «король с пешкой против короля» может оказаться выигрывающей или ничейной, в зависимости от расположения фигур на доске, и можно ставить задачу их узнавания по тем или иным признакам.

Как уже говорилось, возможны различные подходы к такой общей задаче. Одним из них является *алгоритм секущих плоскостей*, предложенный Э. М. Браверманом, который использует *гипотезу компактности*. Если каждый признак имеющихся объектов характеризуется своим числовым значением, то каждый объект изобразится точкой многомерного пространства с числом измерений, равным числу рассматриваемых признаков, причем каждую координату мы будем принимать равной числовому значению этого признака. При этом вовсе не обязательно считать, что признак принимает непрерывное множество значений. Это множество может быть дискретным либо вообще состоять только из 0 и 1, т. е. объект может о б л а д а т ь или н е о б л а д а т ь данным признаком.

Упоминавшаяся *гипотеза компактности* состоит в том, что объекты, принадлежащие одному и тому же классу, расположены «компактно» («кучно»), на «небольшом расстоянии» друг от друга в этом пространстве, а объекты разных классов — достаточно далеко. Иначе говоря, можно провести *секущую плоскость* таким образом, чтобы точки, изображающие объекты одного класса, находились по одну сторону от этой плоскости, а точки другого — по другую. Тогда для любого нового объекта вопрос о его принадлежности к тому или иному классу решается простой проверкой: по какую сторону секущей плоскости лежит соответствующая этому объекту точка (если классов больше, чем два, то может понадобиться рассмотрение нескольких секущих плоскостей).

К сожалению, во многих задачах алгоритм секущих плоскостей оказывается бессильным, так как гипотеза компактности в них не оправдывается. Точки, относящиеся к разным классам для очень многих задач, оказываются очень сильно «перемешанными» в пространстве. Грубо говоря, они расположены как частицы губки и воды в случае, когда вода пропитывает губку. Невозможно провести не только плоскость, но и любую другую гладкую поверхность так, чтобы по одну сторону от нее была только губка, а по другую — только вода.

На иных принципах построены алгоритмы узнавания, предложенные М. М. Бонгардом. В них используется *отбор полезных признаков*. Для удобства его описания будем предполагать, что признаки носят двоичный характер, т. е. что заданные объекты могут обладать или не обладать каждым из рассматриваемых признаков. Кроме того, будем предполагать, что речь идет только о д в у х различных классах.

Среди имеющихся признаков может найтись такой, который принимает значение 1 на всех объектах первого класса и 0 на всех объектах второго. В этом случае задачу узнавания можно считать ре-

шенной, так как о принадлежности нового объекта к тому или иному классу можно судить по этому признаку * .

Разумеется, такой признак удается найти редко и на это не следует рассчитывать. Как правило, встречаются признаки, которые часто принимают значение 1 на объектах одного класса и редко на объектах другого. Имея несколько таких признаков, дающих различные сечения множества объектов, можно попытаться построить такую логическую функцию их, которая будет разделять классы однозначно, т. е. будет принимать значение 1 на всех объектах одного класса и 0 на всех объектах другого, или по крайней мере достаточно часто.

Не вдаваясь в дальнейшие подробности и детали алгоритма, сообщим, что эти алгоритмы нашли широкое применение в решении различных задач узнавания, в частности задач диагностики. Так, на базе алгоритмов, отбирающих полезные признаки, в Онкологическом институте им. П. А. Герцена была разработана программа дифференциальной диагностики, весьма успешно отличающая центральный рак легкого от нераковых опухолей, в частности от туберкулемы, по признакам, полученным из рентгеновского снимка.

Эти же алгоритмы были положены в основу программ, отличающих нефтеносные пласты от водоносных по признакам, полученным на основе геофизических исследований (измерение электрического сопротивления пластов). Обе упомянутые группы программ уже в течение ряда лет находятся в «промышленной эксплуатации» и приносят немалую пользу.

Приведенные примеры далеко не исчерпывают работ в области эвристического программирования и даже направлений этих работ. Но и сказанное позволяет сделать вывод, что в эвристическом программировании имеются немалые достижения.

Тем не менее работа здесь продвигается значительно медленнее и трудности оказываются гораздо более серьезными, нежели представлялось вначале: для воспроизведения даже отдельных сторон умственной деятельности мозга требуется знать о способах этой деятельности намного больше, чем мы сейчас знаем.

Но и это обстоятельство играет положительную роль в развитии науки: попытки создать «думающие программы» ставят много новых задач. Благодаря таким попыткам в целом ряде наук возникли ситуации, аналогичные тем, которые так хорошо охарактеризованы К. Кребером для литературоведения (см. стр. 252). Влияние электронных вычислительных машин сказывается, таким образом, далеко за пределами их непосредственного применения.

---

* Это утверждение справедливо при выполнении постулата *достаточности материала для обучения*. В принципе возможны ситуации, когда это не так, но в таком случае программа вообще не обладает данными, достаточными для надежной классификации объектов.

§ 8. УПРАВЛЯЮЩИЕ МАШИНЫ И СИСТЕМЫ УПРАВЛЕНИЯ

Как мы уже упоминали в § 1, существует три уровня управления, на которых используются вычислительные машины. Мы начнем с применения вычислительных машин для управления процессами. Прежде всего, дадим требуемые общие определения.

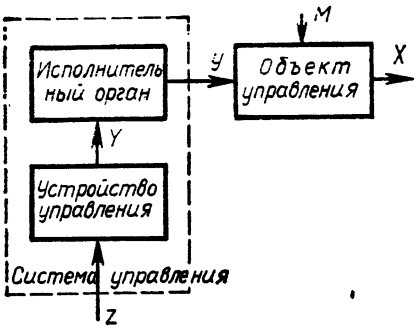


Рис. 112

Управлением принято называть *целенаправленное воздействие* на объект или процесс. Блок-схема (рис. 112) содержит *объект управления*, в котором протекает управляемый *процесс*, и *систему управления*, состоящую из *управляющего устройства* и *исполнительного органа*. Объект управления характеризуется *выходной величиной X* (она же *управляемая* величина) и имеет два входа, на которые поступают *управляющие воздействия Y* от системы управления и *внешние воздействия M* (или *возмущающие* воздействия).

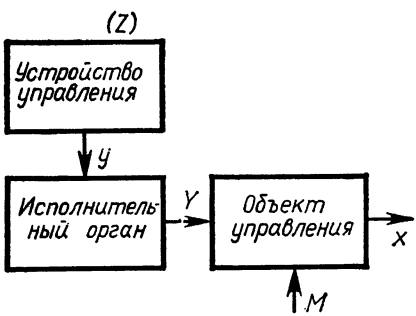


Рис. 113

Под влиянием внешней информации *Z* управляющее устройство вырабатывает сигнал управления *Y*, под воздействием которого исполнительный орган вырабатывает уже управляющее воздействие *y*, непосредственно воздействующее на процесс или объект управления.

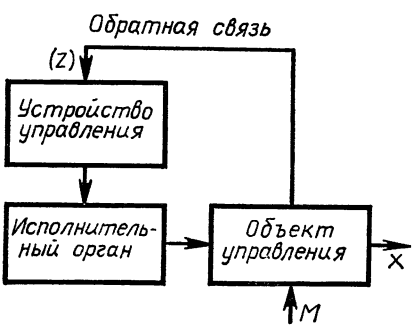


Рис. 114

В зависимости от природы внешней информации системы управления делятся на *разомкнутые системы* и *замкнутые системы* или системы без обратной связи и системы с обратной связью. В разомкнутых системах (рис. 113) внешняя информация должна содержать программу последовательного изменения управляющих воздействий или сведения о возмущающих воздействиях. Напротив, в замкнутых системах управления

(рис. 114) управляющее воздействие вырабатывается на основе сигналов *обратной связи*, содержащих информацию о состоянии управляемого объекта.

Замкнутые системы управления используются, когда имеется много различных возмущающих воздействий и трудно определить зависимость управляющих воздействий от возмущающих. Преимущество разомкнутых систем — в изменении управляющего воздействия до того, как возмущающее успеет существенно изменить значение управляемой величины. Для использования преимуществ тех и других систем их часто объединяют в *комбинированную систему управления*, в которой в формировании сигналов управления участвует как информация об основных внешних воздействиях, так и информация о значениях управляемых величин.

Системы управления решают одну из следующих основных задач: *стабилизация, выполнение программы, слежение, оптимизация.* Задачей *стабилизации* является поддержание управляемой величины X на некотором постоянном уровне (естественно, с заданной точностью) при наличии возмущающих воздействий, закономерных или (и) случайных. Так, в системах энергоснабжения необходимо стабилизировать напряжение и частоту тока в сети вне зависимости от изменения потребления энергии.

Задача *выполнения программы* возникает тогда, когда значения управляемых величин должны изменяться заранее известным образом. Например, при управлении положением трубы телескопа, направленной в определенную точку неба, трубу нужно перемещать определенным образом, чтобы компенсировать вращение Земли. Если изменение управляемой величины заранее неизвестно, то возникает задача *слежения*, состоящая в соблюдении в каждый момент t равенства выходной величины $X(t)$ заданной функции $X_0(t)$.

Не всегда задачу управления можно формулировать перечисленными выше способами. В ряде случаев сведения о требуемом состоянии управляемой величины не могут быть ни заранее введены в систему управления, ни получены в процессе работы. Такая ситуация возникает, например, при управлении электростанцией, работающей в сложных и изменяющихся условиях. Цель управления состоит в том, чтобы — в данном случае с электростанцией — обеспечить максимальное значение коэффициента полезного действия. В других случаях задача состоит в том, чтобы обеспечить максимальное (или минимальное) значение какого-либо другого критерия. Это и есть задача *оптимизации* — установления *оптимального режима* работы объекта управления. При этом должен быть необходимо задан *критерий оптимальности.*

Остановимся в нескольких словах на основных этапах развития систем управления. Первым из таких этапов является *ручное управление*, при котором оператор непосредственно участвует в управлении процессом, используя для получения информации о выходной величине свои органы чувств и сопоставляя их в уме с заданными или интуитивно установленными. Недостатками

ручного управления являются не только субъективность оценок и недостаточная скорость реакции на возмущающие воздействия, но и сложность одновременного управления и документирования основных характеристик процесса. Решение оптимальных задач в таких условиях практически невозможно.

Шагом в направлении улучшения управляющего процесса является *ручное управление с использованием измерительных и регистрирующих приборов*. Здесь функции оператора сводятся уже только к тому, чтобы замечать отклонения и вводить поправки, особенно при наличии автоматически регистрирующих приборов (самописцев).

Следующим шагом явились *локальные системы автоматического регулирования*, в которых удалось полностью исключить человека из цепи обратной связи, заменив оператора *автоматическим регулятором*. Элементарным примером такого регулятора является центробежный регулятор Уатта. В более сложных системах может участвовать оператор, роль которого сводится к вводу эталонных значений выходной величины. Такие регуляторы обычно устанавливались в непосредственной близости от исполнительных органов и назывались *местными* или *локальными*.

По мере того как число автоматически регулируемых переменных возрастало, наблюдение за многими удаленными друг от друга регуляторами становилось слишком сложным. Возникла необходимость в создании *централизованного автоматического регулирования*, при котором все (или большинство) контрольно-измерительных приборов и автоматических регуляторов собраны на *панели управления центрального пункта*. Оператору оставалось следить за приборами и регистрировать их показания, а также вводить эталонные значения для автоматических регуляторов.

Такой переход к централизованному регулированию устранил одни трудности, но породил другие — слежение за большим объемом информации. Для их преодоления были изобретены *машины централизованного контроля*, с помощью которых показания приборов считываются в определенной последовательности *обегающим устройством (коммутатором)*. Эти показания преобразуются в цифровую форму и регистрируются в виде таблиц.

Логические схемы могут сравнивать фактические значения регулируемых переменных с заданными и при отклонениях подавать звуковой или световой сигнал. Однако функции переработки потоков информации и выбор решений для согласования работы отдельных регуляторов остаются за оператором — за человеком.

Централизованное автоматическое регулирование с машинами централизованного контроля является весьма передовой формой автоматизации управления процессами, но сохраняет еще ряд недостатков. Прежде всего, так как управление в целом осуществляется человеком, то в связи с его медленной реакцией на отклонения всегда остается риск аварий или нежелательных режимов. Кроме того, возможны затруднения при очень большом объеме

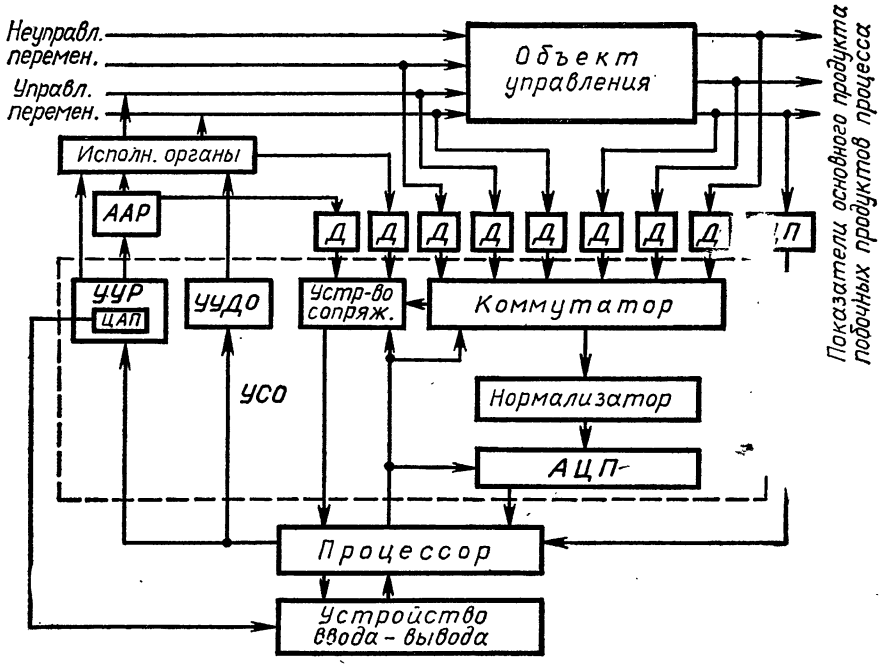


Рис. 115

информации. Наконец, по-прежнему остается практически неосуществимым оптимальное управление.

Все эти недостатки устраняются в системах управления, *использующих цифровые вычислительные машины*. Это может быть сделано различными способами. В одном из вариантов цифровая вычислительная машина используется в роли «советчика» оператора. Вся информация поступает параллельно на панель управления оператора и в вычислительную машину. Вычислительная машина по заранее составленной программе и в соответствии с заданным критерием оптимальности вычисляет требуемые управляющие воздействия, которые и выдаются оператору в качестве рекомендаций. Оператор вручную меняет установки автоматических регуляторов, следуя или не следуя указаниям вычислительной машины по собственному усмотрению.

Другой вариант предполагает непосредственное участие вычислительной машины в управлении процессом, передавая рассчитанные управляющие воздействия прямо на автоматические регуляторы. Как правило, результаты расчетов выводятся при этом и на пульт оператора, который может в случае необходимости воздействовать на регуляторы и вручную. Возможен и третий вариант, когда цифровая машина непосредственно воздействует на исполнительные органы,

минуя автоматические регуляторы. Такой режим работы принято называть *прямым цифровым управлением*. В этом режиме одна цифровая машина (в режиме разделения времени) может заменить несколько сотен регуляторов. Недостаток такого режима только в том, что при выходе из строя цифровой машины работа объекта управления прекращается.

Рассмотрим теперь коротко принципы действия цифровых вычислительных машин, предназначенных для использования в системах управления различными физическими (в частности, технологическими) процессами; такие машины называют *управляющими вычислительными машинами*. От универсальных цифровых вычислительных машин, которые мы рассматривали до сих пор, управляющие машины отличаются наличием дополнительного *устройства связи с объектом*.

Блок-схема системы управления показана на рисунке 115; на ней несколько более детализированы входные и выходные переменные управляемого процесса. Входная информация поступает в управляющую машину с *датчиков*, преобразующих входные и выходные величины в сигналы, удобные для дальнейшей работы. Эти сигналы могут иметь различную физическую природу (электрические, механические и пр.)и носят, как правило, непрерывный характер. Кроме того, они поступают непрерывно по времени.

Управляющая машина оперирует с цифровыми дискретными величинами, которые вводятся и выводятся в дискретные моменты времени. В то же время не только входные, но и выходные величины должны иметь непрерывный характер, так как будут использоваться для автоматического управления исполнительными механизмами. Таким образом, вся информация, связанная с управляющей вычислительной машиной, должна преобразовываться: входная из непрерывной (аналоговой) формы в цифровую, выходная — из цифровой в аналоговую.

Эти преобразования осуществляются специальными устройствами, которые называются *аналого-цифровыми* и *цифро-аналоговыми преобразователями*; как видно из блок-схемы рисунка 115, они входят в состав устройства связи с объектом. Кроме этих преобразователей, устройство связи с объектом содержит:

1) *коммутатор* — устройство, которое поочередно подключает к входу аналого-цифрового преобразователя выходы различных датчиков; это экономит оборудование, позволяя не иметь для каждого датчика свой преобразователь, который много дороже коммутатора;

2) *нормализатор*, осуществляющий унификацию сигналов, т. е. приведение их к единому виду, используемому в данной системе;

3) *устройство сопряжения*, обеспечивающее ввод в оперативную память управляющей машины дискретных сигналов датчиков, уже преобразованных к нужному виду;

4) *устройство управления двухпозиционными органами*. Оно воздействует на такие исполнительные механизмы, которые могут

находиться лишь в двух крайних фиксированных положениях, «открытом» или «закрытом». К ним относятся, например, клапаны, вентили, задержки и т. п.;

5) *устройство управления регуляторами.* Последнее задает входные данные локальным автоматическим регуляторам, работающим уже самостоятельно. Кроме того, это устройство может выдавать управляющие воздействия непосредственно на исполнительные органы, например в случае прямого цифрового управления.

Работой различных блоков устройства связи с объектом управляют сигналы процессора управляющей вычислительной машины. При *синхронной связи* с объектом процесс управления разбивается на циклы тактовыми импульсами датчика времени. В начале каждого цикла, определяющегося приходом тактового импульса в устройство прерывания, производится последовательный опрос датчиков, преобразование их сигналов в цифровую форму и ввод этой информации в оперативную память машины. Этот этап работы занимает небольшую часть цикла, и можно считать, что показания всех датчиков относятся к одному и тому же моменту времени.

Основное время цикла расходуется процессором управляющей машины на расчет необходимых величин управляющих воздействий. Затем следует передача цифрового кода в устройство управления двухпозиционными органами или (и) преобразование цифрового сигнала в аналоговую форму и передача в устройство управления регуляторами. После этого вычислительная машина останавливается либо может перейти к каким-нибудь вспомогательным расчетам до прихода очередного тактового импульса прерывания, начинающего новый цикл управления.

Когда необходима более тесная связь объекта с управляющей машиной, используется *асинхронная связь.* Вместо тактовых импульсов в управляющую машину поступают сигналы датчиков прерывания, непосредственно связанных с объектом; время выдачи сигналов прерывания определяется потребностями управляемого процесса. Вычислительная машина реагирует на импульсы прерывания с учетом заранее установленной системы приоритетов.

В некоторых системах применяется комбинированная связь: наряду с регулярной циклической работой, характерной для синхронной связи, имеются также отдельные, связанные с объектом датчики прерывания, имеющие возможность потребовать управляющего обслуживания в любой нужный момент времени. К таким датчикам относятся, например, датчики аварийных режимов.

Таким образом, управляющие вычислительные машины выполняют следующие обязанности:

1) Собирают, обрабатывают и приводят в удобный для восприятия вид информацию о ходе управляемого процесса и выдают ее оператору;

2) определяют характеристики оптимального режима, удовлетворяющего заданному критерию оптимальности, и обеспечивают фор-

мирование и реализацию управляющих воздействий для поддержания оптимального режима;

3) производят расчет, регистрацию и выдачу текущих и усредненных технологических показателей.

Характеристики вычислительных машин, применяемых для целей управления, меняются в довольно широких пределах. Весьма широко используются *специализированные машины*, предназначенные для использования в каком-либо конкретном технологическом процессе. Эти машины могут быть аналоговыми, цифровыми или гибридными. С примером одной из них — *интерполятором* — мы познакомимся в § 10. Отметим еще, что управляющие машины по сравнению с обычными вычислительными должны обладать повышенной надежностью. Причина этого требования достаточно ясна.

Заключительный вопрос этого параграфа, к рассмотрению которого мы сейчас переходим, является, пожалуй, самым важным и основным в использовании вычислительных машин для целей управления. Чтобы вычислительная машина участвовала в процессе управления некоторым технологическим процессом, необходимо: построить *математическую модель* управляемого процесса; сформулировать *критерий управления*; разработать *алгоритм управления*. На базе этих трех китов современного управления должны быть написаны и введены в память управляющей машины соответствующие программы. Без всего этого любая управляющая машина останется набором дорогих, но бесполезных устройств.

Математической моделью называют совокупность математических зависимостей и логических условий, описывающих поведение реальных объектов. *Алгоритм управления* позволяет вычислить необходимые управляющие воздействия при известных условиях течения процесса; *критерий управления*, он же *критерий оптимальности*, позволяет сравнить различные возможные условия течения процесса и выбрать те, которые считаются наиболее желательными.

Построение математической модели возможно одним из двух путей. Один из них, наиболее желательный, но не всегда возможный, основан на результатах теоретических исследований и логического анализа. Рассмотрим такое построение на простом и для ясности несколько огрубленном примере.

Допустим, что речь идет о плавке в электрометаллургической печи. В состав шихты входят хорошо известные первичные продукты, теплоемкости которых предполагаются заранее известными. Зная точки плавления и скрытую теплоту плавления для каждой составляющей и получив от датчиков сведения о весе каждой из них в шихте (или об общем весе, если процентный состав точно известен), легко подсчитать количество тепла, требуемое для того, чтобы расплавить шихту: если θ означает начальную температуру шихты, а T — самую высокую точку плавления из всех возможных для различных составляющих, то для i-ой составляющей с точкой плавления θ_i нужно затратить

$$Q_i=((\theta_i-\theta)c_{i1}+q_i+(T-\theta_i)c_{i2})m_i$$

калорий тепла, где c_{i1} и c_{i2} — ее удельные теплоемкости в твердом и жидком состояниях, q_i — удельная скрытая теплота плавления и m_i — масса. Суммирование по всем составляющим даст требуемое количество тепла.

С другой стороны, количество тепла, выделяемое электрическим током, подсчитывается по известной формуле Джоуля — Ленца

$$Q=AtRI^2,$$

где t — время, R — сопротивление, I — ток и A — термический эквивалент работы. Сопротивление R также можно считать заранее известным, и если задаться целью провести плавку за заданное время t, то по известному Q можно подсчитать требуемый ток I. Это значение и будет выдано в качестве уровня, который должен поддерживаться автоматическим стабилизатором тока.

Если процесс происходит с какими-либо отклонениями, то пересчет можно производить через определенные требуемые промежутки времени. Возможны и другие постановки задачи. Например, заданным критерием оптимальности может являться максимальная производительность при заданных условиях: наибольшая температура, допускаемая облицовкой печи; наибольший ток, допускаемый источником или электротехническими устройствами печи; ограничения по времени плавки (сверху или снизу) или ограничения по другим параметрам. Другими критериями оптимальности могут служить наибольший коэффициент полезного действия или минимальный расход электроэнергии на тонну готового металла или на тонну шихты.

Каждый из критериев оптимальности может быть выражен в виде некоторой функции от параметров процесса, и задача расчета оптимального режима сведется к нахождению максимума или минимума заданной функции (с учетом имеющихся ограничений). По полученным данным об оптимальном режиме и данным о фактическом состоянии процесса рассчитываются уже требуемые управляющие воздействия.

К сожалению, такое построение математической модели процесса не всегда возможно из-за недостаточной изученности многих процессов; приходится прибегать к статистическим методам.

Статистический метод построения математической модели состоит в использовании экспериментальных данных для подбора эмпирических формул, приближенно представляющих течение управляемых процессов.

Предположим, что критерием оптимальности является количество z выходного продукта, которое как-то зависит от двух регулируемых входных величин x, y. Мы можем произвести несколько одновременных измерений всех интересующих нас величин и получить несколько троек соответствующих друг другу значений (x_i, y_i, z_i).

Допустим, что мы решили изображать нужную нам зависимость простейшей формулой в виде многочлена второй степени

$$z=a+bx+cy+\alpha x^2+\beta xy+\gamma y^2,$$

в которой пока неизвестны коэффициенты a, b, c, α, β, γ. Для их нахождения следует использовать результаты произведенных измерений. Каждая из полученных троек (x_i, y_i, z_i) приводит нас к одному уравнению вида

$$z_i = a + bx_i + cy_i + \alpha x_i^2 + \beta x_i y_i + \gamma y_i^2,$$

где в противоположность привычным обозначениям неизвестными являются коэффициенты a, b, c, α, β, γ.

Так как здесь 6 неизвестных, то для их нахождения достаточно иметь 6 уравнений, т. е. 6 троек — произвести 6 измерений. Однако полученные из такой системы значения коэффициентов не будут пригодны для дальнейшего использования. Причиной этого являются, прежде всего, ошибки измерений. Даже если бы искомая зависимость действительно выражалась написанным многочленом, ошибки при измерении неизвестных привели бы к искажению значений коэффициентов. Еще одной причиной является то, что написанный многочлен выражает нужную нам зависимость лишь приближенно. Равенство

$$z_i - (a + bx_i + cy_i + \alpha x_i^2 + \beta x_i y_i + \gamma y_i^2) = 0$$

на самом деле является приближенным и выполняется с некоторой погрешностью

$$z_i - (a + bx_i + cy_i + \alpha x_i^2 + \beta x_i y_i + \gamma y_i^2) = \varepsilon_i.$$

Чтобы найти пригодные значения коэффициентов, поступают так. Производят большое число измерений $n \gg b$ и определяют значения коэффициентов так, чтобы сумма квадратов погрешностей ε_i была минимальной. Для этого надо искать такие значения a, b, c, α, β, γ, при которых минимальна сумма

$$\sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} (z_i - (a + bx_i + cy_i + \alpha x_i^2 + \beta x_i y_i + \gamma y_i^2))^2.$$

Это приводит к системе линейных уравнений относительно искомых коэффициентов.

После того как коэффициенты найдены, мы можем решать задачу оптимизации, находя такие значения x, y, которым соответствует максимальное количество z выходного продукта. Преимущество вычислительной машины здесь выражается, между прочим, еще и в том, что в процессе работы системы управления коэффициенты $a, \ldots, \gamma$ можно время от времени корректировать с учетом новых экспериментальных данных. Корректировка коэффициентов позволяет уменьшать погрешность, определяемую приближенным характером математической модели.

§ 9. АНАЛОГО-ЦИФРОВЫЕ И ЦИФРО-АНАЛОГОВЫЕ ПРЕОБРАЗОВАТЕЛИ

Роль аналого-цифровых и цифро-аналоговых преобразователей в системах управления была достаточно подробно выяснена в предыдущем параграфе. В настоящем параграфе разберем принципы осуществления таких преобразований и их схемную реализацию. Начнем с рассмотрения преобразования непрерывно изменяющейся во времени величины в последовательность цифровых кодов.

Преобразование непрерывной величины в дискретную состоит в измерении этой непрерывной величины через определенные интервалы времени *(квантование по времени)* и замене полученных значений приближенными, принадлежащими отметке дискретной шкалы *(квантование по уровню)*. Геометрически это эквивалентно аппроксимации непрерывной функции ступенчатой, с заданной заранее сеткой по обеим координатным осям (рис. 116). Как правило, используются равномерные сетки с постоянным шагом. Шаг по времени $\tau=\tau_{i+1}-\tau_i$ называется *периодом квантования*; обратная величина $f=1/\tau$ называется *частотой квантования*. Шаг по уровню называют *разрешающей способностью* преобразователя.

Рассмотрим один метод аналого-цифровых преобразований. Сущность его состоит в последовательном подсчете единичных приращений эталонной величины до совпадения с преобразуемой входной величиной. Поэтому его называют *методом последовательного суммирования и счета единиц приращений*. В процессе преобразования производится подсчет единичных приращений эталонной величины, возрастающая сумма которых сравнивается с преобразуемой величиной x. В момент совпадения величины x и $x_{эт}=\Sigma q=n_x q$ счет прекращается; значение n_x будет числовым эквивалентом величины x.

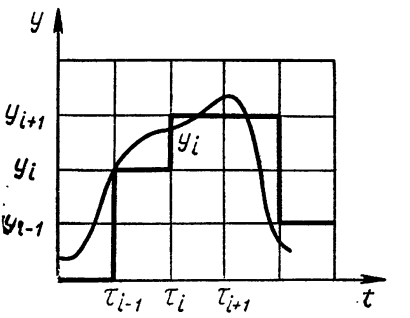


Рис. 116

Метод последовательного суммирования особенно просто реализуется, когда преобразуемой аналоговой величиной является *временная характеристика* сигнала, например длительность импульса. Упрощенная блок-схема такого преобразователя показана на рисунке 117, а временная диаграмма, иллюстрирующая его работу,— на рисунке 118. В начальный момент, соответствующий

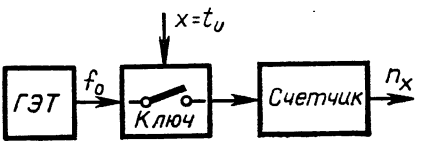


Рис. 117

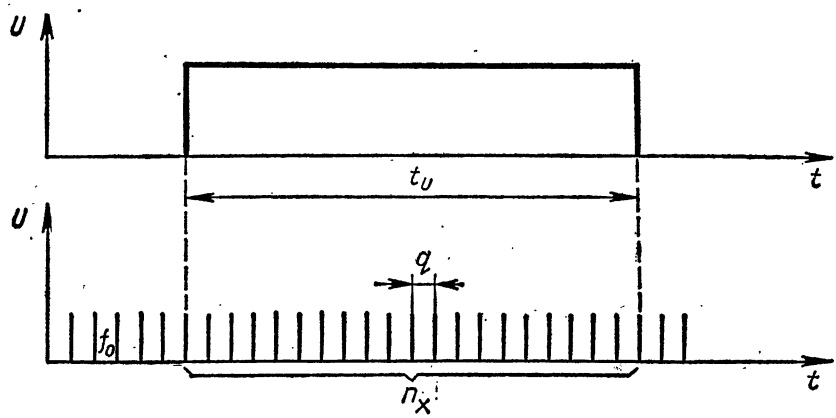


Рис. 118

переднему фронту импульса, счетчик с помощью электронного ключа подключается к генератору импульсов эталонной частоты. После окончания импульса ключ размыкается, прекращая доступ эталонных импульсов в счетчик. Цифровой код, образовавшийся в счетчике, является выходным эквивалентом входной величины t_u.

На практике входной величиной обычно бывает не временной интервал, а напряжение. Удобство описанной выше схемы приводит к мысли воспользоваться искусственным обходным путем: сначала преобразовать напряжение u_x во временной интервал t_u, который затем преобразовать в цифровой код по описанной схеме. Блок-схема такого преобразователя показана на рисунке 119, а временна́я диаграмма его работы — на рисунке 120. Преобразование $u_x \rightarrow t_u$ осуществляется с помощью генератора пилообразного напряжения и схемы совпадения, сравнивающей линейное эталонное напряжение $u_э = kt$ (рис. 120, *а*) с входным u_x. В момент их совпадения эта схема выдает импульс, перебрасывающий в нулевое положение триггер, который одновременно с включением генератора эталонного пилообразного напряжения был переведен в единичное (рис. 120, *б*).

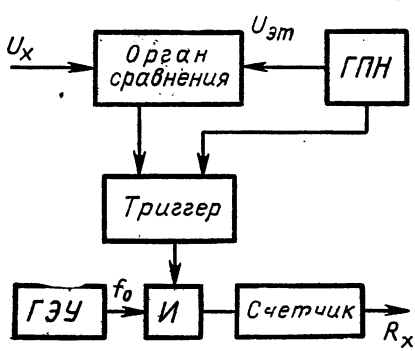


Рис. 119

Длительность пребывания триггера в единичном состоянии (рис. 120, *в*) t_u оказывается пропорциональной напряжению u_x. Триггер управляет схемой совпадения, на второй вход которой поступают тактовые эталонные импульсы (рис. 120, г), проходящие в счетчик только тогда, когда триггер находится в единичном состоянии (рис. 120, *д*). Следовательно, число

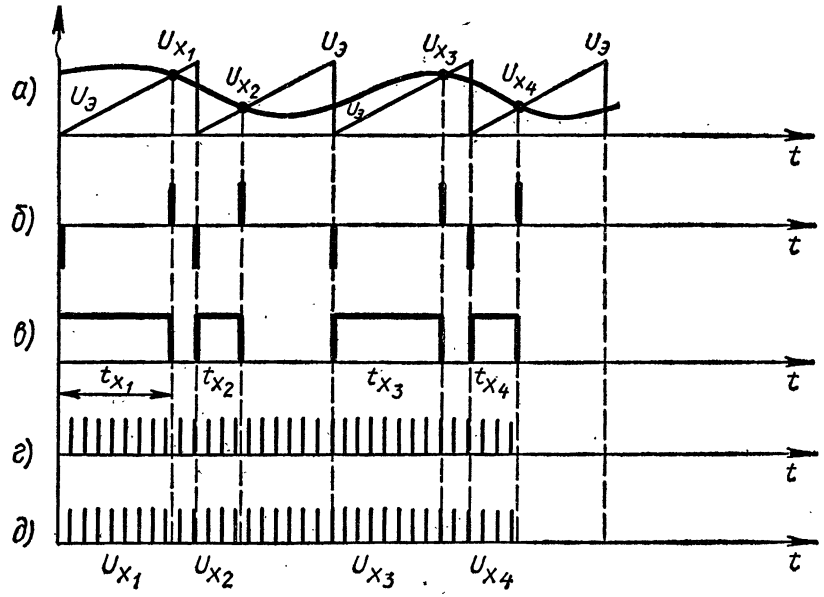


Рис. 120

импульсов на счетчике будет пропорционально входному напряжению. Остается подобрать частоту тактовых импульсов и крутизну k эталонного напряжения так, чтобы коэффициент пропорциональности равнялся единице.

Обратное преобразование дискретных величин в непрерывные осуществляется путем выдачи на выходе цифро-аналогового преобразователя физической величины, имеющей непрерывный характер. Как и в предыдущем случае, возможны различные методы цифро-аналоговых преобразований. Например, можно исходный цифровой код преобразовать сначала в соответствующее количество импульсов. Затем каждому импульсу поставить в соответствие постоянное единичное приращение выходной аналоговой величины. При помощи суммирующего устройства, обладающего цифровой или аналоговой памятью, выполняется суммирование единичных приращений q в нужном количестве, в результате чего величина x на выходе цифро-аналогового преобразователя будет пропорциональна входному числу. Скорость работы такого преобразователя невелика, так как единичные приращения должны суммироваться последовательно.

§ 10. ПРИМЕНЕНИЕ УПРАВЛЯЮЩИХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

Интенсивное применение управляющих вычислительных машин началось в шестидесятых годах с появлением машин второго поколения. Современный этап их развития связан с появлением

машин третьего поколения. Технология интегральных схем позволила уменьшить габариты, повысить быстродействие и поднять надежность, что дало возможность использовать эти машины в системах прямого цифрового управления. Применения управляющих вычислительных машин развиваются по двум направлениям: с одной стороны, это создание одномашинных автономных систем с ограниченными задачами, с другой стороны, централизованные системы для комплексного управления сложными или несколькими технологическими процессами.

В настоящем параграфе мы рассмотрим несколько конкретных примеров применения вычислительных машин. Первый пример, на котором мы остановимся, относится к металлообрабатывающей промышленности. Речь идет об управлении станками, процесс работы которых состоит в перемещении режущего инструмента относительно заготовки. Перемещение это осуществляют рабочие органы станка, которыми управляет вручную или полуавтоматически человек.

Для исключения участия человека в процессе обработки детали было предложено много способов автоматизировать перемещение инструмента (или заготовки) по определенной траектории. Рассмотрим в общих чертах один из них, наиболее распространенный в настоящее время и основанный на применении современной цифровой вычислительной техники,— *цифровое программное управление.*

Главная особенность этого способа задания программы состоит в том, что траектория движения инструмента относительно обрабатываемой заготовки представляется в виде ряда последовательных положений инструмента. Каждое из этих положений определяется группой чисел — координат инструмента в некоторой системе, связанной с неподвижной заготовкой. Совокупность координат, определяющих ряд последовательных положений инструмента, представляет программу работы станка.

Однако задания одних только опорных точек недостаточно для управления перемещением инструмента в промежутках между ними. Для этого используется особое устройство, получившее название *интерполятора*, которое представляет собой небольшую специализированную вычислительную машину. Выходные сигналы интерполятора поступают в устройство управления станка (непосредственно или с предварительной записью на магнитную ленту), представляя собой *унитарный код*, т. е. попросту последовательность единичных импульсов. Каждому импульсу соответствует небольшое «единичное» перемещение рабочего органа, скажем, на 10 *мкм*. Количество таких импульсов в определенный промежуток времени, за который инструмент проходит расстояние между опорными точками, определяет *траекторию движения* инструмента.

Программа для обработки детали на станке с программным управлением составлялась вначале вручную инженером-программистом по чертежу. Эта работа была громоздкой и трудоемкой. Первым шагом в облегчении этой работы было создание языков программирования, специально предназначенных для более удобной для челове-

ка формулировки и записи таких задач. Еще более серьезно облегчает процесс подготовки таких программ использование средств графического взаимодействия, типа дисплея и светового карандаша.

Следующим шагом применения цифровой вычислительной техники в металлообрабатывающей промышленности было создание автоматизированных систем управления целым комплексом станков. При этом цифровая управляющая вычислительная машина не только осуществляет прямое управление станками, но и управляет подачей заготовок и транспортировкой готовых деталей, а также выполняет и у ч е т н ы е функции, собирая и обобщая сведения о движении материальных потоков в производстве.

Еще более широкое применение нашли управляющие вычислительные машины для управления технологическими процессами химической и нефтехимической промышленности, металлургии и энергетики.

Интересным примером применения автоматизированного управления в энергетике является управление энергоблоками. Современный энергоблок включает котельный агрегат, паровую турбину и электрический генератор. В качестве топлива используется газ, жидкое топливо или уголь, который сжигается в пылевидном состоянии. Котельный агрегат состоит из парового котла, в котором за счет энергии сгорающего топлива образуется пар высокой температуры и давления, вращающий паровую турбину. Последняя обычно состоит из трех последовательных частей высокого, среднего и низкого давлений. Паровая турбина служит двигателем для элект-

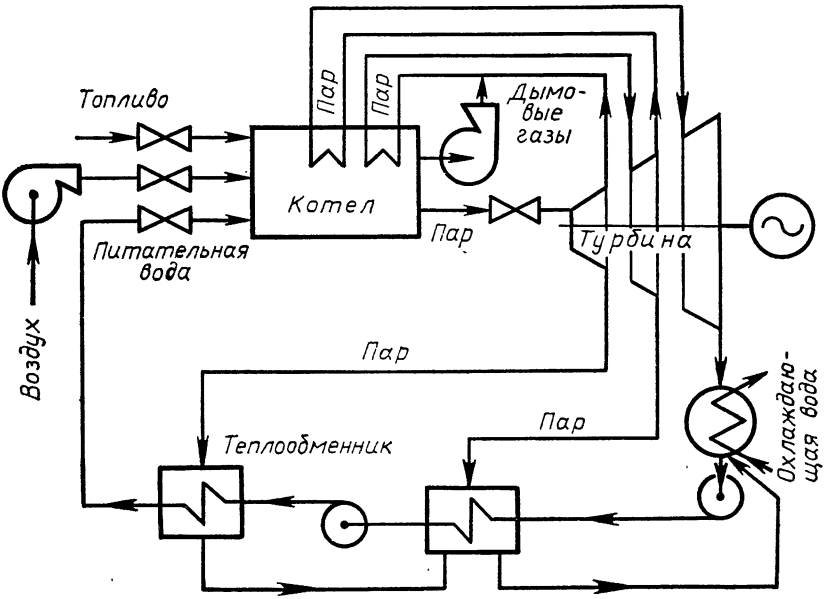


Рис. 121

рического генератора, вырабатывающего электроэнергию (см. схему на рис. 121).

Для обеспечения нормальной работы энергоблока необходимо контролировать около 1000 параметров. Наиболее важными из них являются число оборотов вала турбины, давление и температура пара перед турбиной, давление в конденсаторе турбины, давление и коэффициент избытка воздуха в топке. Математическая модель работы энергоблока представляет собой сложную систему дифференциальных уравнений с частными производными.

В Центральном научно-исследовательском институте комплексной автоматизации (ЦНИИКА) под руководством М. П. Симою была разработана управляющая система «Комплекс» для автоматического управления энергоблоками мощностью от 200 до 800 *Мвт*. Система управляет двумя энергоблоками и является двухступенчатой: в нее входят информационные подсистемы, индивидуальные для отдельных блоков, и общая вычислительная подсистема. Информационные подсистемы обеспечивают сбор и переработку первичной информации от датчиков, сигнализацию и регистрацию параметров, вышедших за допустимые границы. Основные операции по управлению блоком выполняются вычислительной подсистемой. В сложных случаях она работает в роли «советчика» оператора. Система «Комплекс» уже ряд лет находится в промышленной эксплуатации.

Можно привести еще целый ряд примеров создания такого рода автоматических и автоматизированных систем управления в самых различных областях промышленности и транспорта, эксплуатируемых или разрабатываемых и в СССР и за рубежом.

§ 11. УПРАВЛЕНИЕ ПРЕДПРИЯТИЯМИ И АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ УПРАВЛЕНИЯ

В предыдущих параграфах речь шла, главным образом, о применении вычислительных машин в первом из трех этапов управления, о которых мы говорили в § 1 настоящей главы,— автоматическом управлении технологическими процессами. В настоящем параграфе мы хотим еще более коротко осветить две другие ступени — управление производством (АСУП), где речь идет об управлении на уровне цехов или сравнимых с ними подразделений, и управление предприятиями или объединениями (АСУ), т. е. более высокими организационными единицами, до целой отрасли включительно.

Задачи управления производством по своему содержанию существенно отличаются от задач управления технологическими процессами. Здесь первое место занимают задачи планирования и распределения заданий между различными агрегатами и установками, плановая и учетная деятельность, расчет и регистрация технико-экономических показателей и т. п. Ввиду обилия информационного материала обойтись без применения вычислительной техники при его обработке оказывается невозможным. Однако алгоритмы управления на этом уровне пока разработаны еще совершенно недостаточ-

но. Поэтому управление производством осуществляется обычно разомкнутыми системами, которые действуют в роли «советчика», только человек исполняет здесь уже роль не оператора, а роль *диспетчера*.

Ряд планово-экономических задач, возникающих в этой области, требует для своего решения нового математического аппарата и большой вычислительной работы. Поэтому вычислительные машины используются здесь «по своему прямому назначению». Именно благодаря такого рода задачам возник ряд новых областей математики, как «линейное программирование», «динамическое программирование» и т. п.

Познакомимся на простых примерах с постановкой и методами задач линейного программирования. Рассмотрим сначала простейший случай так называемой *распределительной задачи*. Пусть имеется два различных станка, на которых можно производить одно и то же изделие, причем стоимость изготовления одного изделия на первом станке равна α_1, а на втором — α_2. Производительность первого станка равна β_1 изделий за единицу времени (например, за час), второго — β_2. Требуется распределить задание между станками так, чтобы было произведено не менее m изделий и чтобы стоимость их изготовления была минимальной.

Обозначим через x_1 время работы первого станка и через x_2 — время работы второго. За это время будет изготовлено $\beta_1 x_1 + \beta_2 x_2$ изделий, причем по условию $\beta_1 x_1 + \beta_2 x_2 \geqslant m$. Кроме того, очевидно, что $x_1 \geqslant 0$, $x_2 \geqslant 0$ и $x_1 \leqslant a_1$, $x_2 \leqslant a_2$, где a_1 и a_2 — фонды времени для станков, т. е. общее число часов (скажем, за месяц) минус время, отведенное на профилактику, планово-предупредительный ремонт и т. п. Стоимость изготовления всех изделий равна

$$S = \alpha_1 \beta_1 x_1 + \alpha_2 \beta_2 x_2, \tag{1}$$

и требуется найти минимум функции S при условии, что x_1 и x_2 удовлетворяют указанным ограничениям

$$\begin{gathered} 0 \leqslant x_1 \leqslant a_1, \quad 0 \leqslant x_2 \leqslant a_2, \\ \beta_1 x_1 + \beta_2 x_2 \geqslant m. \end{gathered} \tag{2}$$

Так как у нас всего два переменных, то задачу можно иллюстрировать геометрически на плоскости. Ограничения (2) показывают, что допустимыми значениями x_1, x_2 являются в данном случае (см. рис. 122) точки внутри треугольника BEF (в других случаях прямая EF может пересекать прямоугольник $OABC$ по другим сторонам). Легко сообразить, что минимум функции (1) может достигаться либо в точке E, либо в точке F (подумайте почему?).

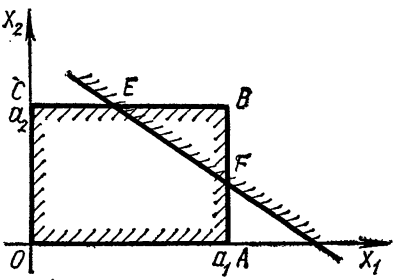


Рис. 122

Как мы видим, речь идет об отыскании экстремума линейной функции (1) при наличии линейных ограничений (2). Такая задача и есть *общая задача линейного программирования*. В самой общей постановке распределительная задача формулируется так: имеется p станков и q видов продукции, причем заданы α_{ij} — стоимость изготовления единицы j-ой продукции на i-ом станке и β_{ij} — производительность (штук в час) i-го станка при изготовлении j-ой продукции. Кроме того, известны ресурсы времени $a_1, \ldots, a_p$ по каждому станку и плановые задания $b_1, \ldots, b_q$ по каждому виду продукции. Требуется распределить задания по станкам так, чтобы общая стоимость производства была минимальной.

Если x_{ij} означает время работы i-ого станка на изготовление j-ой продукции, то задача сводится к отысканию минимума функции

$$S=\sum_{i=1}^{p}\sum_{j=1}^{q}\alpha_{ij}\beta_{ij}x_{ij}, \tag{3}$$

при условиях

$$x_{ij}\geqslant 0 \ (i=1, 2, \ldots, p;\ j=1, 2, \ldots, q),$$

$$\sum_{j=1}^{q}x_{ij}\leqslant a_i \quad (i=1, 2, \ldots, p),$$

$$\sum_{i=1}^{p}\beta_{ij}x_{ij}\geqslant b_j (j=1, 2, \ldots, q).$$

Геометрическая интерпретация этой задачи аналогична рассмотренной выше, однако здесь приходится рассматривать $n=pq$ переменных. Минимум по-прежнему достигается на одной из вершин многогранника, но их оказывается слишком много для того, чтобы найти точку минимума простым перебором, особенно когда n достаточно велико. На практике число перебираемых вершин может достигать порядка 10^6—10^7, поэтому приходится пользоваться специальными методами отыскания минимума. Одним из наиболее распространенных является *симплекс-метод*, легко программируемый и потому широко используемый для решения задач линейного программирования на цифровых вычислительных машинах.

Автоматизированные системы управления предприятиями или объединениями должны содержать в себе как подсистемы рассмотренные выше системы управления производством и, следовательно, решать такие же задачи, но они имеют также и ряд специфических задач. В частности, вышестоящая система должна вырабатывать и выдавать своим подсистемам управления производством критерии оптимальности.

Некоторые задачи, возникающие перед такой системой, можно сформулировать на языке программирования, и тогда они решаются теми же методами. К таким задачам относится, например, *транспорт-*

ная задача. Вместе с тем имеется и целый ряд более сложных задач, в которых функция или ограничения не имеют линейного характера. Их часто называют задачами *нелинейного программирования*.

Сформулируем общую постановку транспортной задачи. Пусть имеется m поставщиков и n потребителей некоторого продукта и заданы мощности $a_1, a_2, \ldots, a_m$ каждого поставщика и потребности $b_1, b_2, \ldots, b_n$ каждого потребителя, причем $\sum_{i=1}^{m} a_i \geqslant \sum_{j=1}^{n} b_j$. Кроме того, задана матрица $\| c_{ij} \|$ размером $m \times n$, элементы которой c_{ij} означают стоимость перевозки единицы продукта от i-ого поставщика j-ому потребителю. Требуется составить план перевозок таким образом, чтобы стоимость перевозок была минимальной.

Математически задача сводится к отысканию минимума функции

$$S = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

с условиями $x_{ij} \geqslant 0$ ($i=1, 2, \ldots, m$; $j=1, 2, \ldots, n$) и

$$\sum_{j=1}^{n} x_{ij} \leqslant a_i \quad (i=1, 2, \ldots, m),$$

$$\sum_{i=1}^{m} x_{ij} = b_j \quad (j=1, 2, \ldots, n).$$

Легко видеть, что это обычная задача линейного программирования.

Решение транспортной задачи широко используется при планировании перевозок в самых крупных масштабах, например при планировании перевозок угля в масштабе всего Советского Союза.

ОГЛАВЛЕНИЕ

Глава V. ЭЛЕМЕНТЫ ПРОГРАММИРОВАНИЯ

Глава VI. АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ

Глава VII. СЕГОДНЯ И ЗАВТРА

Глава VIII. ПРИМЕНЕНИЯ ЭЛЕКТРОННЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

Рафаил Самойлович Гутер,
Юрий Леонович Полунов

МАТЕМАТИЧЕСКИЕ
МАШИНЫ

Очерки вычислительной техники

Редактор *Л. М. Котова*

Художник переплета *В. Ф. Громов*

Художественный редактор *Е. Н. Карасик*

Технический редактор *В. Ф. Коскина*

Корректоры *К. П. Лосева, В. Г. Соловьева*

Сдано в набор 27/VI 1975 г.
Подписано к печати 11/XI 1975 г. 60×90¹/₁₆. Бумага тип. № 3. Печ. л. 18,0. Уч.-изд. л. 18,15.
Тираж 106 тыс. экз. А 05580. Заказ № 3078.

Ордена Трудового Красного Знамени издательство «Просвещение» Государственного комитета Совета Министров РСФСР по делам издательств, полиграфии и книжной торговли. Москва, 3-й проезд Марьиной рощи, 41.

Ордена Трудового Красного Знамени Первая Образцовая типография имени А. А. Жданова Союзполиграфпрома при Государственном комитете Совета Министров СССР по делам издательств, полиграфии и книжной торговли. Москва, М-54, Валовая, 28

Цена 50 коп.

50 коп.