ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА В ИНЖЕНЕРНЫХ И ЭКОНОМИЧЕСКИХ РАСЧЕТАХ

Под редакцией докт. техн. наук,
проф. Б. В. Анисимова

Допущено Министерством
высшего и среднего
специального образования СССР
в качестве учебника
для студентов высших
учебных заведений

МОСКВА «ВЫСШАЯ ШКОЛА» 1975

6Ф7
В92
УДК 681.3(075.8)

А. В. Петров, В. Е. Алексеев, М. А. Титов, В. И. Суровцев, П. Н. Шкатов

Рецензенты: кафедра «Вычислительная техника» Ленинградского института точной механики и оптики и докт. техн. наук, проф. В. И. Дракин.

В92 **Вычислительная** техника в инженерных и экономических расчетах. Учебник для вузов. Под ред. Б. В. Анисимова. М., «Высш. школа», 1975.

302 с. с ил.

На обороте тит. л. авт.: А. В. Петров, В. Е. Алексеев, М. А. Титов [и др.].

В учебнике рассмотрены принципы действия и информационные основы построения ЭВМ; приведена методика подготовки инженерных и экономических задач для ЭВМ, численные методы и алгоритмы их решения; алгоритмические языки «АЛГОЛ-60», «ФОРТРАН», «КОБОЛ» и автокод «ИНЖЕНЕР».

Предназначается для студентов технических и инженерно-экономических вузов. Может быть полезен инженерно-техническим работникам, занимающимся решениями инженерных и экономических задач на ЭВМ.

В $\frac{30502-178}{001(01)-75}$ 146—75 6Ф7

ВВЕДЕНИЕ

Основой современной научно-технической революции является бурное развитие электронной вычислительной техники, широко используемой во всех областях научной, производственной и хозяйственной деятельности. В настоящее время вычислительная техника располагает различными техническими средствами и почти неограниченными возможностями для выполнения работ, связанных с логической обработкой информации и вычислениями. Так, без вычислительных машин немыслимо решение трудоемких задач, выбор оптимальных вариантов проектов, рациональной структуры устройств, эффективного использования комплексов машин и систем различного назначения.

Вычислительная машина становится незаменимым инструментом в руках инженера-исследователя, открывает большие возможности в новых разработках. До применения быстродействующих вычислительных машин искусство исследователя заключалось в основном в максимальном упрощении задачи, выявлении и отбрасывании тех факторов, которые не оказывают существенного влияния на изучаемый объект. Расчетная часть при разработке новых образцов техники составляла небольшую долю и сводилась к приближенным решениям. Задача решалась в основном путем проведения эксперимента, что требовало больших затрат труда, времени и материальных средств. В настоящее время искусство исследователя состоит в том, чтобы наиболее полно и точно охарактеризовать изучаемое явление и дать его строгое математическое описание.

Сократить путь от сложных математических уравнений до получения конкретных результатов, ускорить выполнение простейших вычислительных операций и освободить человека от утомительной работы помогает вычислительная машина.

Необходимость в своевременной и качественной обработке всевозможной информации приводит в настоящее время к широкому использованию вычислительных машин для управления процессами и объектами в различных областях промышленности, транспорта, в военном деле. Автоматические и автоматизированные системы управления осуществляют сбор, хранение, передачу и переработку информации, отражающей состояние регулируемых объектов. Информация, выработанная системой, используется для оперативного воздействия на управляемый объект (или процесс) с целью поддержания нужного

состояния. Основу подобных систем управления составляют вычислительные машины.

Несколько обособленно следует выделить область использования вычислительных машин для выполнения экономических расчетов, научно обоснованного экономического анализа деятельности промышленных предприятий и различных отраслей народного хозяйства.

Ведение хозяйства при социалистическом способе производства обязывает специалистов учитывать разнообразные факторы (например, затраты труда на различные виды продукции), сравнивать прошлые затраты с настоящими и будущими, оценивать имеющиеся ресурсы и обеспечивать наиболее выгодные условия для развития народного хозяйства. Поэтому использование вычислительных машин для оптимального планирования, расчета всех трудовых затрат, себестоимости выпускаемой продукции и фондов заработной платы, определения расходов основных и вспомогательных материалов и т. д. позволяет улучшать и совершенствовать административно-хозяйственную деятельность производства.

Вычислительные машины в зависимости от того, какими величинами представлена в них информация, делят на аналоговые (АВМ) и цифровые (ЦВМ). Используются и комбинированные аналого-цифровые вычислительные машины, сочетающие в себе как аналоговый, так и дискретный принцип действия.

АВМ строят из элементов, в которых изменение тока, напряжения или других величин удовлетворяет тем же уравнениям, что и параметры изучаемого на них явления; поэтому их называют аналоговыми, или моделирующими. Исходные данные, вводимые в АВМ, представляют собой непрерывно изменяющиеся во времени значения каких-либо физических величин (напряжений, токов, углов поворота, длин и т. п.). Решение задачи в машине производится параллельно на всех ее элементах, соединенных между собой в соответствии со структурной схемой решаемой задачи. Результаты решения выводятся на экран электронно-лучевой трубки или фиксируются на измерительных приборах (например, вольтметре).

ЦВМ оперируют цифровыми величинами, принимающими ряд отдельных (дискретных) численных значений, представленными в виде электрических импульсов или перепадов напряжений. Процесс решения задачи на такой машине состоит из отдельных последовательных элементарных арифметических операций: сложения, вычитания, умножения, деления (последовательность выполнения операций задается в программе вычислений, заранее составленной). Результаты решения выводятся из машины в виде, удобном для восприятия человеком.

При использовании аналоговой или дискретной машины в той или иной области следует иметь в виду их сравнительную характеристику.

Достоинство АВМ — высокое быстродействие, определяемое длительностью переходных процессов в деталях элементов, простота конструкции, наглядность и простота ввода и вывода данных, возможность эксплуатации машины техническими работниками без большой предварительной подготовки, простота в изменении параметров исследуе-

мой задачи, возможность работы в различных масштабах времени. Высокое быстродействие позволяет использовать АВМ в качестве управляющих машин во многих системах автоматического регулирования, где необходимо получать решения в том масштабе времени, в каком протекают сами управляющие процессы.

Недостаток АВМ — невысокая точность получаемых результатов решения (0,5—10%) и узкоспециализированность (предназначены в основном для решения линейных и нелинейных систем дифференциальных уравнений различного порядка).

Достоинство ЦВМ — универсальность (на них может быть решена практически любая задача, если разработан численный метод решения); большая точность вычислений, зависящая от величины разрядной сетки машины.

Элементарные операции ЦВМ выполняют со скоростью, достигающей сотен тысяч и даже миллионов операций в секунду, что обеспечивается высоким быстродействием элементов и совмещением во времени многих вспомогательных операций.

Недостаток ЦВМ — довольно трудоемкий и длительный процесс подготовки задачи для решения.

Возможности ЦВМ определяются уровнем развития теории и техники программирования, а также быстродействием процессора и емкостью памяти машины. При наличии современных высокопроизводительных вычислительных средств проблема их рационального использования может быть решена только подготовкой квалифицированных программистов.

В основу книги положены лекции, читаемые на протяжении многих лет авторами на кафедре «Электронные вычислительные машины» МВТУ им. Баумана.

Материал в учебнике представлен таким образом, что может быть хорошо усвоен при различном количестве часов, отводимых в учебных планах на изучение курса «Вычислительная техника в инженерных и экономических расчетах». Изложение некоторых разделов, в частности алгоритмических языков и принципов устройства некоторых современных ЦВМ, предполагает наличие особенностей преподавания этого курса в различных вузах.

Авторы признательны и благодарны коллективу кафедры «Вычислительная техника» Ленинградского института точной механики и оптики и докт. техн. наук, проф. В. И. Дракину за ценные указания, сделанные в процессе рецензирования данного учебника.

Все замечания по книге просим направлять по адресу: *Москва, К-51, Неглинная ул., 29/14, издательство «Высшая школа».*

Авторы

ГЛАВА 1

ЦИФРОВЫЕ ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ

§ 1.1. ПРИНЦИП ДЕЙСТВИЯ И СТРУКТУРНАЯ СХЕМА ЦВМ

ЦВМ предназначены для обработки информации, представленной в дискретной форме. При решении инженерных и экономических задач ЦВМ оперирует с числами и алфавитно-цифровыми словами (операндами), которые в виде исходных данных подаются на вход машины и в виде результата получаются на ее выходе. ЦВМ может выполнять некоторый ограниченный набор арифметических и логических действий (операций), причем в каждый момент времени машина в состоянии выполнить лишь одно из них. Поэтому всякий вычислительный процесс должен быть представлен в виде последовательности инструкций, записанных в том порядке, в котором необходимо выполнять действия. Так, например, вычисление значения функции $Z = (ax^2 + b)/c$ сводится к выполнению действий умножения, сложения и деления в порядке, указанном в табл. 1.1.

Таблица 1.1

Наименование действия	Участвующие числа (операнды)		Результат
	I	II	
Умножение	a	x	$P_1 = ax$
Умножение	P_1	x	$P_2 = ax^2$
Сложение	P_2	b	$P_3 = ax^2 + b$
Деление	P_3	c	$Z = (ax^2 + b)/c$

В таблице через P_1, P_2 и P_3 обозначены числа, равные результату соответствующих действий, а через Z — число, равное значению искомой функции.

Арифметические и логические операции в ЦВМ выполняются арифметическим устройством (АУ). Информация о том, какие именно действия должно выполнить АУ в настоящий момент, над какими операндами и куда следует поместить результат, задается с помощью команд. Кроме того, если вычислительный процесс состоит из нескольких действий, то необходимо указывать, какое действие должно выполняться после данного.

Таким образом, для выполнения вычислений машине надо задавать не только исходные данные, но и последовательность команд, определяющих, какие действия, над какими числами и в какой последовательности надо проводить. Эту последовательность команд называют *программой решения задачи.*

Исходные данные, промежуточные и окончательные результаты, а также команды программы хранятся в памяти машины. При этом каждое число, слово или команда в виде числового кода хранится в отдельной ячейке, имеющей свой номер или адрес. Другими словами, машина может производить действия не только над числами, но и над алфавитно-цифровой информацией, что используется для преобразования последней и формирования машиной новых команд. Информация о том, какие операнды участвуют в выполнении данного действия, задается в виде адресов или номеров ячеек, в которых они хранятся.

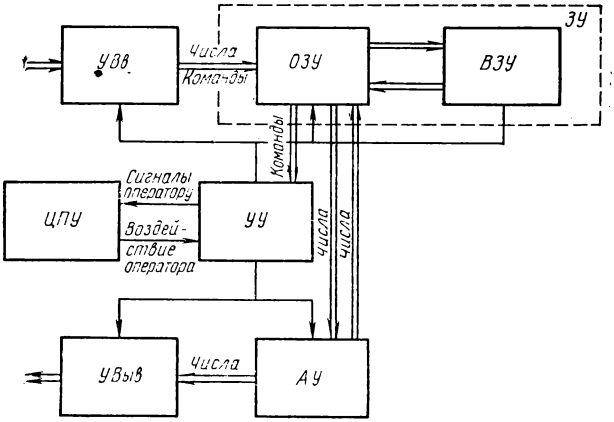


Рис. 1.1. Структурная схема ЦВМ

В процессе решения задач необходимо также выбирать из памяти команду, подлежащую выполнению на данном этапе, и расшифровывать по информации в ней заключенной, какое действие и над какими операндами должно выполнить АУ по этой команде. Эти функции реализует устройство управления.

Поскольку для решения задачи в машину вводятся исходные данные и программа решения, а результаты решения выдаются из машины, то ЦВМ должна быть укомплектована соответствующими устройствами ввода и вывода.

Для контроля вычислительного процесса, вмешательства в его ход в случае надобности, а также для пуска ЦВМ последняя снабжается пультом управления.

Следовательно, в состав ЦВМ должны входить следующие основные устройства: арифметическое (АУ); запоминающее (ЗУ); управления (УУ); ввода (УВв); вывода (УВыв); центральный пульт управления (ЦПУ).

Структурная схема ЦВМ с наиболее важными связями между устройствами показана на рис. 1.1, где *УВв* — устройство ввода, *ЗУ* — запоминающее устройство; *ОЗУ* — оперативное запоминающее устройство; *ВЗУ* — внешнее запоминающее устройство; *ЦПУ* — центральный пульт управления; *УУ* — устройство управления; *УВыв* — устройство вывода; *АУ* — арифметическое устройство.

Кроме этих основных устройств, ЦВМ укомплектовывается внешними устройствами, предназначенными для записи исходных данных и программ решения на внешние носители (перфоленты и перфокарты).

Из рис. 1.1 видно, что устройства ЦВМ обмениваются между собой информацией двух видов — кодами (числа, слова, команды) и управляющими сигналами. Связи, по которым осуществляется передача кодов, на рисунке представлены двойными стрелками, а связи, по которым осуществляется передача управляющих сигналов,— одинарными. Центральный пульт управления *ЦПУ* через устройство управления *УУ* и информационные связи обоих видов связан со всеми устройствами. Эти связи на рисунке не показаны.

Рассмотрим принцип действия ЦВМ.

Устройство ввода *УВв* по сигналу из устройства управления *УУ* считывает входную информацию (исходные данные и программу) с внешнего носителя и записывает ее в ячейки оперативной памяти. По сигналу с устройства управления *УУ* из соответствующей ячейки памяти машины выбирается подлежащая выполнению команда, которая запоминается в специальном регистре устройства управления *УУ*. Затем *УУ* с помощью дешифратора команд определяет, какое действие надо выполнить по этой команде, вырабатывает и передает в арифметическое устройство *АУ* соответствующий сигнал. Кроме того, *УУ* определяет адреса (номера) ячеек, хранящих участвующие в данном действии операнды, и вырабатывает сигналы, по которым содержимое этих ячеек считывается и посылается в *АУ*. Арифметическое устройство *АУ* выполняет указанное действие над операндами и результат снова посылает в определенную устройством управления ячейку памяти машины.

После окончания выполнения одной команды устройство управления *УУ* выбирает из памяти следующую команду и цикл снова повторяется. Если очередная команда является командой выдачи результата, то *УУ* вырабатывает сигнал, с помощью которого результат из арифметического устройства поступает в устройство вывода.

§ 1.2. ПРИНЦИП ДЕЙСТВИЯ ОСНОВНЫХ УСТРОЙСТВ ЦВМ

ЦВМ оперирует с числами, которые представляются в машине в виде последовательности цифр. Для изображения одноразрядного числа необходим элемент, имеющий столько устойчивых состояний равновесия, сколько цифр в данной системе счисления. Тогда каждому устойчивому состоянию элемента будет соответствовать одна из цифр этой системы. Например, если ЦВМ считает в десятичной системе счисления, то элемент должен иметь десять устойчивых состояний, которым присваиваются значения 0, 1, ..., 9. С помощью одного такого элемента можно изображать одноразрядное число. Для изображения числа, состоящего из N разрядов, необходимо иметь N таких элементов, тогда один элемент будет изображать количество

единиц, другой — количество десятков, третий — количество сотен и т. д. Набор элементов для хранения одного числа называют *ячейкой*.

Большинство существующих электронных элементов имеют лишь два устойчивых состояния, например, лампа или полупроводниковый транзистор могут либо пропускать ток, либо не пропускать, контакты реле могут быть либо замкнуты, либо разомкнуты, сердечник намагничен в состояние с остаточной индукцией $+B_r$ или $-B_r$ и т. д., поэтому в большинстве современных машин используют двоичную систему счисления.

Арифметическое устройство. Арифметическое устройство (АУ) предназначено для выполнения всех арифметических и логических операций. В большинстве современных машин участвующие в операциях числа поступают в АУ последовательно в разные временные такты. Первое число поступает в специальный регистр и там запоминается, второе число поступает в сумматор.

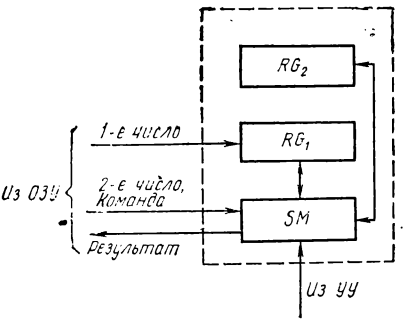


Рис. 1.2. Структурная схема АУ

Регистр, как и ячейка, состоит из элементов с двумя устойчивыми состояниями равновесия и используется для запоминания числа.

Сумматор состоит из набора элементов с двумя устойчивыми состояниями, соединенных друг с другом связями, предназначенными для переносов из младшего разряда в соседний старший. Принцип действия сумматора основан на счете поступающих на его вход импульсов. Всякий раз, как только в данном разряде сумматора получается результат бóльший или равный основанию системы счисления, происходит перенос единицы в соседний старший разряд.

При выполнении некоторых действий, например умножения, необходимо до конца операции хранить и множитель и младшие разряды результата. Для этих целей используют второй регистр.

Структурная схема АУ приведена на рис. 1.2.

Регистр RG_1 предназначен для хранения первого числа, участвующего в операции. Регистр RG_2 играет вспомогательную роль при выполнении операций над числами (хранит результат, множитель и младшие разряды произведения при выполнении операций деления и умножения и т. д.). Сумматор *SM* предназначен для хранения второго, участвующего в операции числа, а также для выполнения операций над числами и командами. Регистр RG_1 и сумматор *SM* связаны с памятью машины, сумматор *SM*, кроме того, связан с обоими регистрами.

Работа арифметического устройства *АУ* осуществляется следующим образом. Из памяти машины в регистр RG_1 поступает первое число, затем в сумматор *SM* — второе число. По сигналу из устройства управления *УУ* все блоки *АУ* настраиваются на выполнение данного действия. В сумматоре *SM* производится соответствующее дей-

ствие и получается результат. Затем результат записывается в память машины. На этом заканчивается выполнение одного действия (команды). Для следующей команды указанный цикл повторяется снова.

Запоминающие устройства. Память машины должна обладать большой емкостью и малым временем обращения, т. е. временем поиска и считывания информации из нее. Поскольку в настоящее время нет ЗУ, которые удовлетворяли бы обоим этим требованиям в полной мере, в современных ЦВМ используют оперативную и внешнюю память.

Оперативная память при выполнении операций обменивается числами и командами с другими устройствами машины, поэтому время обращения к ней должно быть малым, так как в противном случае уменьшается быстродействие ЦВМ. Для изображения каждого разряда двоичного числа в оперативной памяти используют элемент с двумя устойчивыми состояниями. Ячейка, предназначенная для запоминания одного числа, состоит из набора таких элементов. Попытка создания ЗУ большой емкости на таких элементах приводит к значительному увеличению его стоимости и габаритов. В современных ЦВМ емкость оперативной памяти колеблется от 4000 до 32 000 двоичных чисел, а время обращения к ней составляет 1 мкс.

Внешние ЗУ используют иные принципы запоминания информации, например, запись на магнитную ленту, магнитный барабан или диск. Емкость памяти таких накопителей зависит от площади магнитного носителя, причем чем больше эта площадь, тем большее время затрачивается на поиск и считывание информации.

Внешняя память не имеет непосредственной связи с другими устройствами машины, кроме оперативной памяти. Иформация из внешней памяти записывается в оперативную память и только после этого может быть использована при решении задачи.

Оперативные запоминающие устройства. В настоящее время оперативные ЗУ строятся, как правило, с использованием магнитных сердечников с обмотками. В зависимости от направления тока в обмотках сердечник намагничивается в состояние с остаточной магнитной индукцией $+B_r$ или $-B_r$, что соответствует 0 или 1.

Запись и считывание информации в избранном сердечнике производится при подаче импульсов тока по адресным шинам, на пересечении которых находится этот сердечник.

Запись и считывание информации из одной ячейки происходит параллельно, т. е. одновременно по всем разрядам.

Для сохранения информации в ячейках при считывании оперативное ЗУ имеет схемы регенерации или восстановления исходных состояний сердечника. Такие ЗУ позволяют получить время обращения около 1 мкс.

Запоминающие устройства на ферритовых сердечниках. Структурная схема оперативного ОЗУ на ферритовых сердечниках представлена на рис. 1.3.

По кодовым шинам адреса *КША* в регистр адреса RG_A поступает адрес числа. С помощью адресного коммутатора *АК* выбирается нуж-

пая ячейка магнитного куба *МК* и происходит запись или считывание числа в соответствующую ячейку магнитного куба *МК*. Число, подлежащее записи в *МК*, или считанное из него, запоминается в регистре числа $RG_{Ч}$, который связан с машиной кодовыми шинами числа *КШЧ*. Работой всех блоков запоминающего устройства управляет специальный блок управления, который на схеме не показан.

Внешние запоминающие устройства. Накопитель на магнитной ленте. Запись информации на магнитную ленту осуществляется с помощью магнитных головок записи, а считывание информации — с помощью магнитных головок воспроизведения. Го-

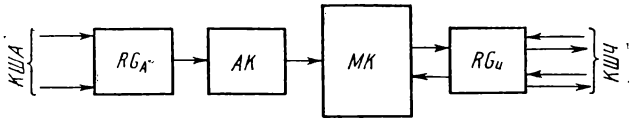


Рис. 1.3. Структурная схема оперативного ЗУ на ферритах

ловка — это кольцевой сердечник с обмоткой, обладающий высокой магнитной проницаемостью. В передней части кольцевого сердечника имеется рабочий зазор.

При записи информации импульсы тока, представляющие собой выражение соответствующего числа, поступают на головку записи $Г_з$ (рис. 1.4). Под воздействием импульса тока в головке $Г_з$ создается магнитное поле. Поскольку зазор в головке $Г_з$ представляет большое сопротивление для магнитного потока, то он замыкается через ферромагнитный слой ленты *МЛ*, намагничивая ее до состояния с остаточной индукцией $+B_r$ или $-B_r$. Магнитная лента *МЛ* перемещается относительно головок, таким образом на ней будут оставаться участки *А*, намагниченные в момент действия импульса тока записи.

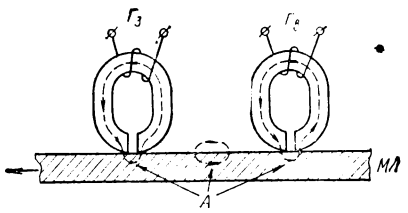


Рис. 1.4. Схема магнитной записи и воспроизведения

При считывании информации поле намагниченного участка замыкается через головку воспроизведения ($Г_в$), так как материал сердечника представляет для него малое магнитное сопротивление. Под воздействием этого поля в обмотке головки $Г_в$ индуцируется напряжение.

Несколько магнитных головок собирают в блок, который позволяет осуществлять запись на магнитную ленту одновременно нескольких разрядов числа или всего числа.

Емкость накопителя на магнитной ленте зависит от длины последней и составляет от нескольких сот тысяч до нескольких миллионов чисел. Но необходимо помнить, что чем больше длина ленты, тем большее время затрачивается на поиск и считывание нужной информации.

Накопитель на магнитном барабане. Магнитный барабан представляет собой цилиндр из диамагнитного материала, на поверхность которого нанесен ферромагнитный слой, являющийся носителем информации. Запись и считывание информации осуществляется с помощью магнитных головок, расположенных вдоль образующей барабана. Барабан приводится во вращение с большой угловой скоростью специальным двигателем. Запись информации осуществляется на кольцевые дорожки параллельным или последовательно-параллельным способом.

Запись или считывание информации происходит путем подачи сигнала на соответствующие головки в тот момент, когда под головками окажется нужная ячейка. Выбор нужной ячейки осуществляется сравнением кода заданной ячейки с кодом, считываемым со специальной дорожки барабана.

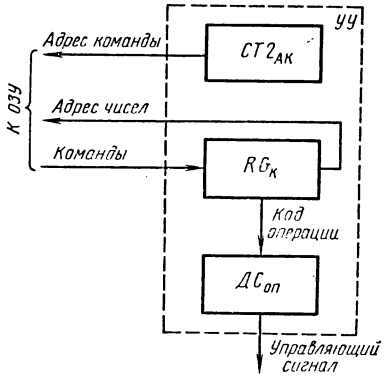


Рис. 1.5. Структурная схема УУ

Время обращения к накопителю на магнитном барабане значительно меньше, чем время обращения к накопителю на магнитной ленте. Однако следует учитывать, что при этом емкость его также значительно меньшая.

Накопитель на магнитных дисках. В этом накопителе в качестве носителя информации используют набор дисков. Диски покрыты ферромагнитным слоем, а магнитные дорожки на них располагаются в виде концентрических окружностей с обеих сторон диска. Принцип записи и считывания тот же, что и у накопителя на магнитном барабане. Емкость накопителя на магнитных дисках значительно больше, чем накопителя на магнитном барабане, и составляет до миллиона чисел.

Устройство управления. Устройство управления (УУ) предназначено для управления всеми устройствами вычислительной машины в соответствии с программой. УУ включает в себя: счетчик адреса команд $CT2_{АК}$, запоминающий адрес команды, подлежащей выполнению; регистр команд $RG_К$, в который записывается команда, подлежащая выполнению; дешифратор операции $ДС_{оп}$, расшифровывающий информацию о том, какую операцию надо выполнять по данной команде (рис. 1.5).

В ЦВМ принят, как правило, порядок выполнения команд естественный, т. е. после команды из ячейки N выполняется команда из ячейки $N + 1$. Перед выполнением программы в счетчик $CT2_{АК}$ заносится адрес первой команды программы (после выполнения каждой команды содержимое этого счетчика увеличивается на единицу). Содержимое счетчика $CT2_{АК}$ (адрес команды) поступает в *ОЗУ*. Далее из соответствующей ячейки *ОЗУ* извлекается команда, подлежащая выполнению. Эта команда записывается в $RG_К$, где из нее выделяется

информация, показывающая, какую операцию нужно выполнить (код операции), и подается на $ДС_О$, который вырабатывает соответствующий управляющий сигнал. Из команды выделяются также адреса участвующих в операции чисел и подаются в *ОЗУ* для выборки из него этих чисел.

Устройства ввода. С помощью устройств ввода (УВв) в ЦВМ вводят исходные данные и программы, которые предварительно кодируются с помощью системы отверстий на перфолентах или перфокартах. При этом наличие отверстия соответствует 1, а отсутствие — 0.

Наибольшее распространение получили устройства ввода, использующие фотооптический принцип считывания. При этом принципе считывания между осветителем и фотоэлементами (фотодиодами, фототранзисторами, кремниевыми фотоэлементами) протягивается перфолента или перфокарта. Когда мимо фотоэлемента проходит отверстие, свет попадает на него и в результате вырабатывается фотоэлектрический импульс, который затем усиливается, формируется и поступает в машину.

Устройства вывода. Устройства вывода результатов (УВыв) строятся чаще всего по принципу электромеханических печатающих устройств, достоинство которых заключается в том, что результаты непосредственно печатаются на бумажную ленту и обеспечивают хорошую четкость.

Принцип работы печатающего устройства состоит в следующем. Между печатающим колесом, на котором нанесены символы, и молоточками помещается бумажная лента. В момент прохождения нужного знака молоточек ударяет по бумаге. Печатающее устройство имеет несколько таких колес, поэтому за один оборот осуществляется печать всего числа (слова) или даже целой строки.

Наибольшее распространение из печатающих устройств получили быстродействующий печатающий механизм (БПМ), предназначенный для печати чисел со скоростью 20 чисел/с и алфавитно-цифровое печатающее устройство (АЦПУ), предназначенное для печати алфавитно-цифровой информации со скоростью 10—20 строк/с. На БПМ на узкую бумажную ленту печатается, как правило, лишь числовая информация. При этом на печать выводятся не более 11 символов с учетом знаков.

На АЦПУ на широкую бумажную ленту печатается как числовая, так и алфавитная информация, причем за один такт печатается целая строка, насчитывающая до 128 символов (в зависимости от конструкции устройства набор используемых символов может быть от 64 до 128).

Результаты решения печатаются также на телетайпе. Но ввиду малой скорости работы этого устройства информация из машины выводится предварительно на перфоратор, где пробивается в виде системы отверстий на бумажной ленте, а затем только автономно распечатывается на телетайпе. Количество символов, которое можно отпечатать на одной строке телетайпа, не может превышать 68. Однако все символы строки печатаются последовательно один за одним. Этим и объясняется малая скорость такого устройства.

§ 1.3. ПОДГОТОВКА ДАННЫХ ДЛЯ ВВОДА В ЦВМ

Исходные данные и программы перед вводом в ЦВМ наносятся на носители информации. В качестве носителей информации в настоящее время наиболее широко используют перфокарты и перфоленты. Информация на перфокартах и перфолентах представляется в виде системы отверстий, причем каждому символу соответствует своя комбинация отверстий.

Информация на носители записывается с помощью специальных устройств подготовки данных последовательно — символ за символом. В зависимости от способов расположения информации на носителе и ввода ее в ЦВМ различают:

а) перфокарты с параллельной построчной записью информации;

б) перфоленты с последовательной записью информации;

в) перфоленты с последовательно-параллельной записью информации.

При последовательной записи информации на носителе в каждый момент времени в ЦВМ вводится код одного символа (для ввода слова, состоящего из n кодов, потребуется n тактов). При параллельной записи информации на носителе — все слово, а при последовательно-параллельной — часть информации слова. Под *словом* понимают информацию, хранящуюся в одной ячейке, т. е. слово может представлять собой число, команду или какую-либо буквенно-цифровую информацию.

Перфокарта. Перфокарта представляет собой картонный прямоугольник со срезанным верхним левым углом. По ширине перфокарта разбивается на 80 колонок. По высоте перфокарты может располагаться до 12 строк, в каждой из которых записывается одно число или слово.

Рассмотрим способы нанесения информации на перфокарты на примере перфокарт для ЦВМ М-222.

Первые 18 колонок перфокарты используются для служебной информации. В них пробивается номер перфокарты, номер задачи и другая вспомогательная информация. В зависимости от того, что представляет из себя слово (число, команда или буквенно-цифровая информация), распределение колонок для нанесения одного символа различное. Для кодирования десятичной цифры необходимо четыре колонки, так как каждой цифре должна соответствовать своя система отверстий. После каждых четырех колонок, предназначенных для записи одной цифры, оставляется одна свободная колонка. Десятичное число располагается на перфокарте так, как показано на рис. 1.6. В 22-ю и 24-ю колонки записывается знак числа и знак порядка, в колонках с 26-й по 31-ю — порядок числа, а в колонках с 34-й по 77-ю — мантисса числа.

Вертикальными линиями на рисунке отделены колонки, предназначенные для записи одной цифры мантиссы и порядка или знака.

Команды записываются в восьмеричной системе счисления. Для изображения каждой цифры восьмеричной системы достаточно трех колонок. Аналогично изображаются восьмеричные константы. Пример

записи команды и восьмеричной константы на перфокарте показан на рис. 1.7. Код операции *КОп* представляется двухразрядным числом в восьмеричной системе счисления, а адреса A_1, A_2, A_3 — четырехразрядными числами. Для восьмеричной константы отводится 14 разрядов.

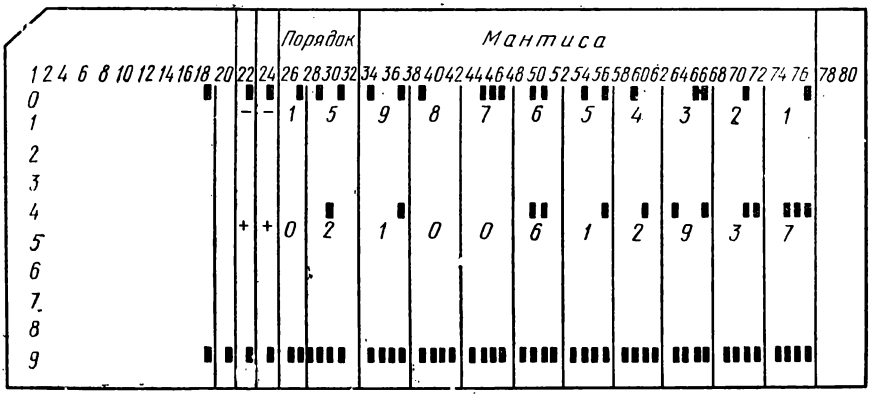


Рис. 1.6. Пример записи чисел на перфокарте

При записи буквенно-цифровой информации количество колонок, необходимых для записи одного символа, зависит от общего количества различных используемых символов входного языка. На рис. 1.8 представлен пример записи буквенно-цифровой информации на языке А Л Г О Л.

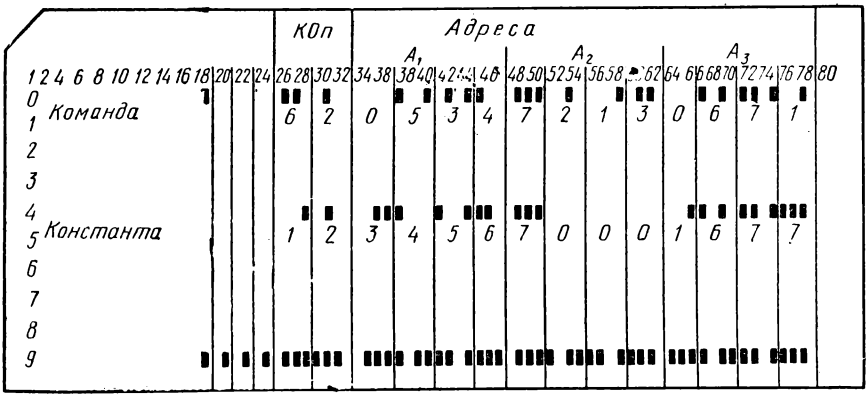


Рис. 1.7. Пример записи команд на перфокарте

Перфолента. Перфолента — это плотная бумажная или целлулоидная лента, которая может быть различной ширины. По ширине перфоленты выделяют несколько дорожек для перфорации отверстий.

Информацию на перфоленте можно представить либо в служебном коде машины, либо в коде входного языка. Служебный код предназна-

чен для ввода десятичных чисел и команд, а также служебной информации, необходимой для обеспечения различных режимов ввода. Код входного языка используют для ввода программ, записанных на том или ином алгоритмическом языке.

Ввод информации, записанной в служебном коде, может быть адресным, групповым или адресно-групповым. При адресном вводе перед каждым числом или командой записывается номер (адрес) ячейки, в которую должно быть помещено следующее за ним число или команда. При групповом вводе числа или команды записываются без адресов и располагаются в памяти машины в последовательных ячейках в том порядке, в котором они нанесены на перфоленту. Адрес

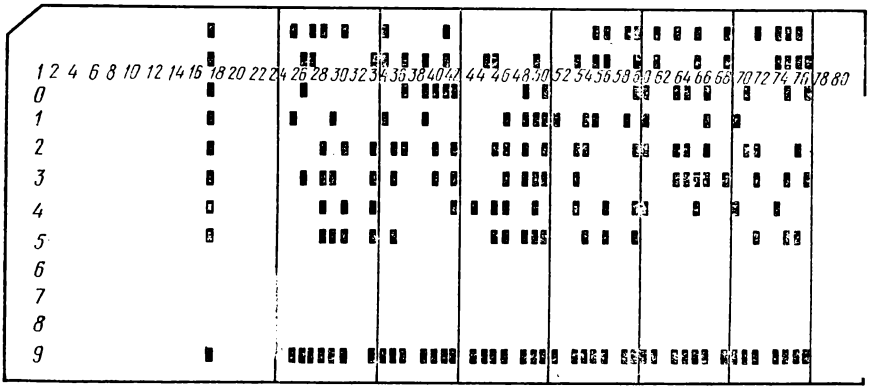


Рис. 1.8. Запись алфавитно-цифровой информации на перфокарту

первой ячейки массива указывается в команде ввода. При адресно-групповом методе ввода допускается произвольное чередование адресного и группового ввода, но перед каждым массивом указывается адрес первой ячейки массива.

На ленте шириной 17,5 мм имеется пять кодовых дорожек и дорожка синхронизации. По дорожке синхронизации перфорируются отверстия меньшего диаметра.

Примеры записи информации на перфоленте показаны на рис. 1.9, *а* — запись десятичного числа — 0,978650234 (длина слова 10 строк); на рис. 1.9, *б* — запись команды в восьмеричном коде +120073456012 (длина слова 13 строк); на рис. 1.9, *в* — запись различных буквенно-цифровых символов во втором международном телеграфном коде (длина слова не ограничена).

Устройства подготовки данных. В качестве устройств подготовки данных используют, как правило, клавишные устройства, предназначенные для автоматического преобразования исходной информации (исходных данных и программ) в систему отверстий на носителях. Такие устройства осуществляют также одновременно с перфорацией носителя печать записываемой информации на бумажную ленту, используемую для контроля правильности. Набор символов на клавиатуре этого устройства определяется алфавитом входного языка.

Рассмотрим устройства подготовки данных, используемые для перфорации лент шириной 17,5 мм.

Для нанесения служебного кода исходных данных и программ на ленту используется устройство подготовки данных, созданное на базе телеграфного аппарата СТА-2М. Клавиатура этого устройства представлена на рис. 1.10. Здесь, кроме цифр, имеются клавиши для записи знаков как восьмеричного, так и десятичного чисел, а также набор вспомогательных клавиш: «ПЕР» — передача адреса — код, который ставится после адреса (номера ячейки); «ЗАП» — запись

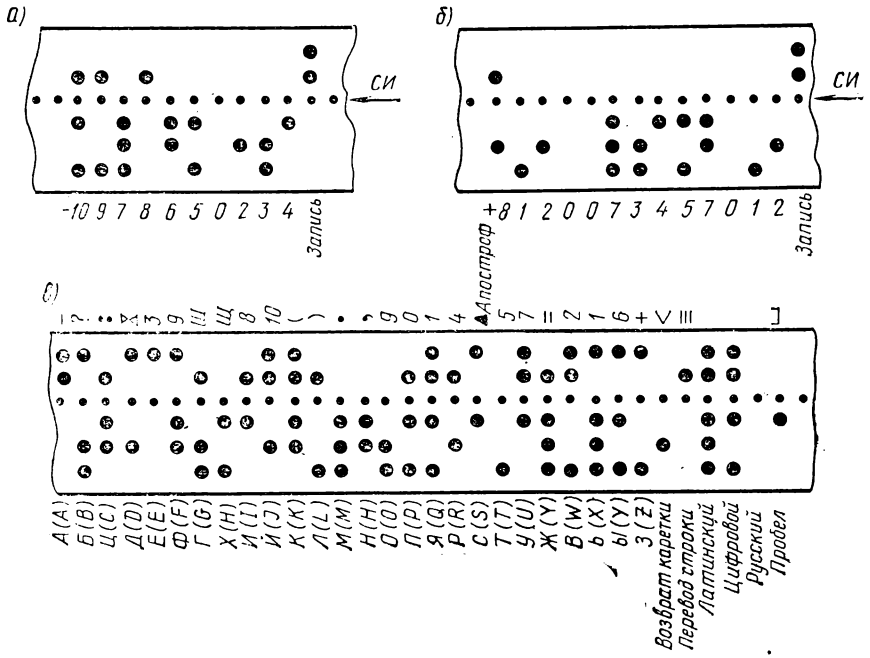


Рис. 1.9. Запись алфавитно-цифровой информации на перфоленту

(этим кодом заканчивается каждое слово); «,» — десятичная запятая; «граница», с помощью которой ограничиваются массивы информации. Длинная клавиша используется для перфорации кода «пробел».

Для нанесения алфавитно-цифровой информации на ленту применяют стандартный телеграфный аппарат СТА-2М, клавиатура которого представлена на рис. 1.11. Это устройство позволяет вводить все символы второго международного телеграфного кода. Поскольку на пятидорожечной ленте можно записать лишь 32 различные комбинации отверстий, а количество символов этого кода 78, не считая служебных, то используют трехрегистровый способ записи, при котором каждой комбинации отверстий на ленте соответствуют три различных символа. Для отличия этих символов перед каждым ставится код соответствующего регистра.

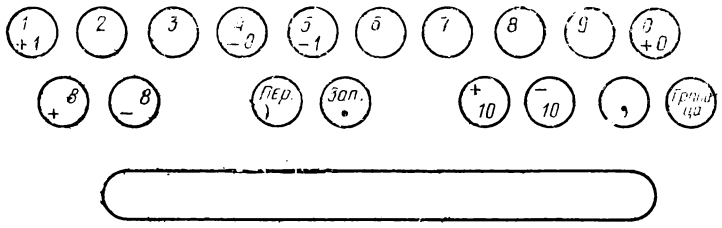


Рис. 1.10. Клавиатура устройства подготовки цифровых данных

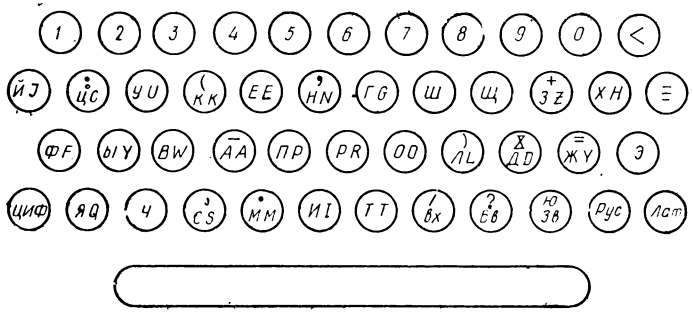


Рис. 1.11. Клавиатура устройства подготовки алфавитно-цифровой информации

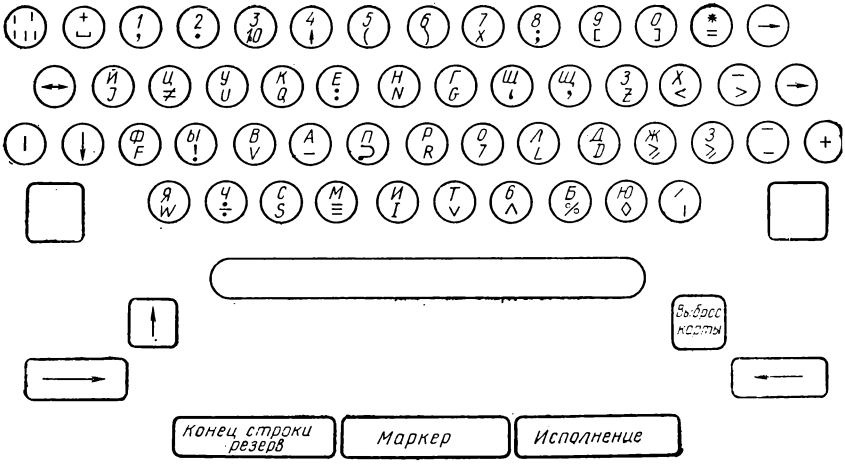


Рис. 1.12. Клавиатура устройства подготовки данных на перфокартах

На латинском регистре записываются буквы латинского алфавита, на русском — 26 букв русского алфавита, буквы Ч, Ш, Щ, Э, Ю, цифры и остальные символы телеграфного кода записываются на цифровом регистре.

Устройство подготовки данных машины М-222 осуществляет запись информации на перфокарты. В состав этого устройства входит электрическая пишущая машинка и перфоратор. При нажатии на клавишу пишущей машинки вырабатывается электрический сигнал, который затем преобразуется и поступает в перфоратор, в результате приводится в движение тот или иной набор пуансонов, пробивающих соответствующую комбинацию отверстий на перфокарте. Здесь в отличие от предыдущего устройства имеется лишь два регистра (как в пишущей машинке). На верхнем регистре набиваются цифры, русские буквы и некоторые специальные символы, на нижнем регистре — латинские буквы и все остальные символы (рис. 1.12). Кроме того, клавиатура имеет набор клавиш, используемых для нанесения служебной информации.

§ 1.4. ЦИФРОВЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

В связи с бурным развитием вычислительной техники все более расширяются области ее применения. Если первые вычислительные машины использовали, как правило, лишь для проведения научных и технических расчетов, то теперь сфера их применения распространилась буквально на все области деятельности человека, связанные с обработкой больших объемов информации.

Расширение сферы использования вычислительной техники предъявило новые требования, как к отдельным ее устройствам, так и к организации работы в целом. Это, в свою очередь, привело к созданию вычислительных систем (ВС) в виде ряда обладающих различными характеристиками машин (процессоров) с унифицированными каналами обмена информацией, позволяющими образовать вычислительные комплексы с различным составом оборудования.

В зависимости от области применения предъявляют различные требования к составу периферийных устройств, объему оперативной и внешней памяти, быстродействию процессора и т. д. Поэтому отдельные функциональные устройства строят в виде агрегатов, которые в нужном составе и количестве объединяются в ВС. Таким образом, ВС создаются из ряда моделей процессоров, обладающих различными характеристиками. К процессорам через унифицированные каналы подключают любые периферийные устройства, общие для всего ряда моделей.

В связи с появлением ВС остро встала проблема информационной и программной совместимости между отдельными моделями ряда, что потребовало единого для всех моделей способа кодирования информации и одинаковых или кратных длин машинных слов, а также возможность выполнения одной и той же программы решения задачи на любой модели ряда.

С учетом сказанного ВС должна состоять из процессора, внутренней памяти и каналов, к которым подключаются периферийные устройства (накопители на магнитных лентах, считывающие устройства ввода, перфораторы и печатающие устройства). Типичная схема ВС представлена на рис. 1.13, где *Пр* — процессор, *К* — канал, *П* — перфоратор, *ПУ* — печатающее устройство, *ВП* — внутренняя память, *НМЛ* — накопитель на магнитной ленте.

Принципы построения и действия памяти и периферийных устройств аналогичны рассмотренным ранее. Поэтому рассмотрим лишь вопросы, связанные с принципом действия и структурой процессора и каналов ввода — вывода.

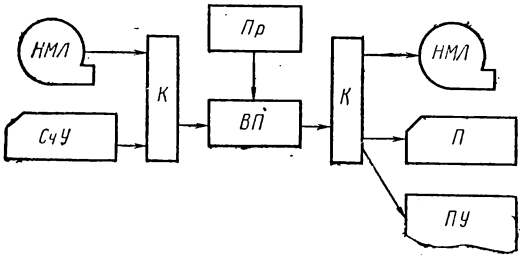


Рис. 1.13. Схема ВС

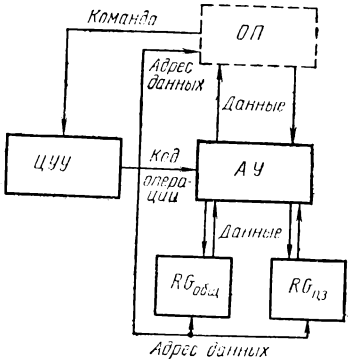


Рис. 1.14. Структурная схема процессора

Процессор. Процессор представляет собой устройство, выполняющее арифметические и логические преобразования информации (арифметическое устройство) и управляющее выполнением команд программы (устройство управления). Структурная схема процессора и основных информационных связей между его устройствами и памятью представлена на рис. 1.14.

Принцип действия процессора следующий. Центральное устройство управления *ЦУУ* системы определяет порядок выполнения команд и по адресам осуществляет их выборку из оперативной памяти *ОП*. Затем расшифровывает команды, вырабатывает управляющие сигналы, соответствующие коду операции, выделяет адреса участвующих в операции данных, которые из памяти *ОП* подаются в арифметическое устройство. *АУ* выполняет над этими данными соответствующие действия и результаты помещает в память *ОП* или в регистры общие $RG_{общ}$ или с плавающей запятой $RG_{п.з}$.

Часть функций, которые в вычислительных машинах первых образцов относились к устройствам управления, в современных машинах переданы отдельным устройствам, что позволяет совмещать во времени их работу (например, работу процессора и периферийных устройств). Это несколько увеличило объем используемого оборудования, но зато повысило производительность ВС.

Арифметическое устройство *АУ* может выполнять операции над числами как с фиксированной, так и с плавающей запятой, а также

оперировать с данными переменной длины, т. е. может выполнять операции не только над словами фиксированной длины (содержимым одной ячейки), но и над словами переменной длины (содержимым нескольких последовательных ячеек).

Общие регистры $RG_{общ}$ используются для хранения участвующих в операции чисел и результатов, представленных в форме с фиксированной запятой, а также для других целей, например, в качестве регистров индекса, базовых регистров, счетчика команд и т. д. В каждом регистре может храниться информация длиной в одно слово. В случае, если для выполнения операции требуется регистр бо́льшей длины, например при выполнении деления с фиксированной запятой, то используются два соседних регистра, начиная с четного.

Регистры с плавающей запятой $RG_{п.з}$ предназначены для хранения лишь чисел с плавающей запятой. В каждом таком регистре может храниться информация длиной в два слова.

В регистрах с плавающей запятой могут храниться оба участвующих в операции числа, т. е. при выполнении данной операции обращения к памяти не происходит. Это существенно позволяет увеличить скорость вычислений.

В случае выполнения операций над полями переменной длины обмен информацией осуществляется, как правило, лишь с памятью.

Каналы ввода — вывода. Стремление совместить работу процессора с работой периферийных устройств ввода — вывода в ВС привело к выделению функций управления работой периферийных устройств из центрального устройства управления и передачи их каналу, что придает периферийным устройствам автономность.

Периферийные устройства различных типов требуют различных устройств управления, учитывающих специфику каждого из них. Но обеспечение всякого периферийного устройства собственным устройством управления приводит к росту объема оборудования, а возможность подключения к ВС того или иного набора периферийного оборудования требует стандартизации форматов информации и команд.

Многие функции управления периферийными устройствами не зависят от типа последних и их выполняет канал, являющийся посредником между памятью и периферийными устройствами.

Таким образом, если требуется осуществить обмен информацией между периферийными устройствами и процессором или памятью (иначе говоря надо выполнить команды ввода — вывода), то процессор посылает в канал соответствующую команду. После этого он продолжает свою работу, не дожидаясь обмена информацией.

Канал автономно от процессора выполняет те функции по вводу — выводу информации, которые являются общими для всех периферийных устройств, независимо от принципа их действия и формы представления в них информации. В зависимости от скорости работы периферийных устройств используют каналы селекторные и мультиплексные.

Селекторный канал подключается к одному из периферийных устройств и осуществляет передачу информации из этого устройства (или

к этому устройству) до тех пор, пока не будет передана вся информация. Только после этого к селекторному каналу может быть подключено другое периферийное устройство. Селекторные каналы используют для работы с быстродействующими устройствами ввода — вывода.

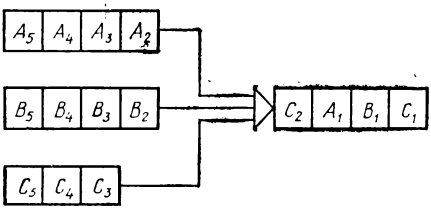


Рис. 1.15. Принцип работы мультиплексного канала

Мультиплексный канал предназначен для одновременной работы с несколькими периферийными устройствами по принципу побайтной (послоговой) передачи информации. Другими словами, после передачи одного байта информации из одного периферийного устройства канал подключается к другому периферийному устройству и т. д.

Принцип работы мультиплексного канала показан на рис. 1.15, где A_i, B_i, C_i — байты информации, передаваемой с устройств типа A, B, C.

Периферийные устройства, выполняющие специфические для них функции, снабжаются промежуточными управляющими блоками, которые могут обслуживать одно устройство или целую группу однотипных периферийных устройств. Промежуточные блоки управления выполняют специфические для данного периферийного устройства функции преобразования информации и управления его механизмами.

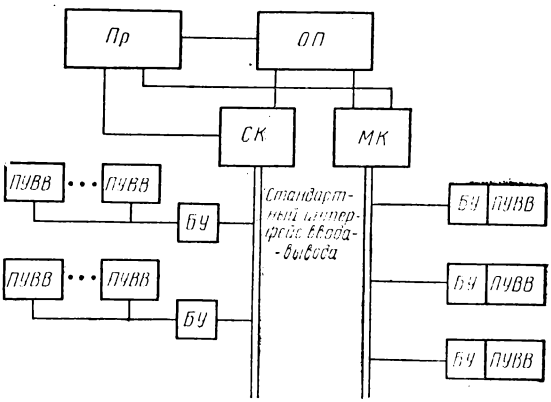


Рис. 1.16. Логическая схема ВС

Периферийные устройства и их блоки управления связываются с каналом посредством стандартной системы сопряжения. Физически сопряжение представляет собой набор шин, проходящих через все периферийные устройства, и электронных схем, формирующих сигналы, проходящие через эти шины. Шины, предназначенные для передачи по ним информации, называют *информационной магистралью*, а шины, предназначенные для передачи управляющих сигналов — *служебными*.

На рис. 1.16 приведена схема логической структуры ВС, где *ОП* — оперативная память; *Пр* — процессор; *МК* — мультиплексный канал; *СК* — селекторный канал; *БУ* — блоки управления внешними устройствами; *ПУВВ* — периферийное устройство ввода — вывода.

ГЛАВА 2

АРИФМЕТИЧЕСКИЕ ОСНОВЫ ЦВМ

§ 2.1. СИСТЕМЫ СЧИСЛЕНИЯ

Системой счисления называют способ представления чисел посредством числовых знаков (цифр).

Всякая система счисления характеризуется *основанием* — количеством цифр, принятых для записи чисел. Так, десятичная система счисления использует для записи чисел 10 цифр 0; 1; 2; 3; 4; 5; 6; 7; 8; 9. Эта система является позиционной, т. е. значение цифры (ее вес) зависит от ее положения (позиции) в числе. Например, число 33,3 состоит из трех цифр 3, вес каждой из которых различен. Первая цифра 3 указывает, сколько десятков в числе, вторая — количество единиц и третья — количество десятых долей единицы, т. е. это число можно представить в виде степенного ряда: $33{,}3 = 3\cdot 10^1 + 3\cdot 10^0 + 3\cdot 10^{-1}$.

В степенной ряд можно разложить число, записанное в любой позиционной системе счисления, т. е. в общем виде число $\alpha_n\alpha_{n-1}\ \dots\ \alpha_0,\ \alpha_{-1}\alpha_{-2}\ \dots\ \alpha_{-m}$, записанное в системе счисления с любым основанием q, можно разложить в ряд по степеням основания этой системы:

$$\alpha_n\,\alpha_{n-1}\ \dots\ \alpha_0,\ \alpha_{-1}\,\alpha_{-2}\ \dots\ \alpha_{-m} = \alpha_n\,q^n + \alpha_{n-1}\,q^{n-1} + \dots +$$
$$+ \alpha_0\,q^0 + \alpha_{-1}\,q^{-1} + \alpha_{-2}\,q^{-2} + \dots + \alpha_{-m}\,q^{-m},$$

где α_i — цифры этой системы счисления.

Существует бесконечное множество систем счисления; рассмотрим лишь те из них, которые имеют отношение к ЦВМ.

Двоичная система счисления. В двоичной системе счисления для изображения чисел используют цифры 0 и 1.

В десятичной системе счисления с помощью одного разряда записываются десять различных чисел от 0 до 9, а число 10 (основание системы) и бóльшие числа изображаются с помощью двух и более разрядов. При этом число $N = q^m$ (где q — основание системы; m — целое число) изображается в виде единицы в старшем разряде с последующими нулями в количестве m. Аналогично этому в двоичной системе счисления с помощью одного разряда записываются лишь числа 0 и 1, а число 2 и бóльшие числа записываются с помощью двух и более разрядов. Так, число $2 = 2^1$ записывается в виде 10, а число $4 = 2^2$ — как 100 и т. д. Каждое число, большее данного на единицу, получается путем прибавления 1 к младшему разряду.

Несколько десятичных чисел и соответствующие им двоичные, восьмеричные и шестнадцатиричные числа представлены в табл. 2.1.

Большинство элементов, на которых строятся ЦВМ, имеют лишь два устойчивых состояния равновесия; одному из этих состояний равновесия присваивается значение цифры 1, а другому — 0. По этой

Таблица 2.1

Числа							
десятичные	двоичные	восьмеричные	шестнадцатиричные	десятичные	двоичные	восьмеричные	шестнадцатиричные
0	0	0	0	9	1001	11	9
1	1	1	1	10	1010	12	A
2	10	2	2	11	1011	13	B
3	11	3	3	12	1100	14	C
4	100	4	4	13	1101	15	D
5	101	5	5	14	1110	16	E
6	110	6	6	15	1111	17	F
7	111	7	7	16	10000	20	10
8	1000	10	8				

причине большинство современных ЦВМ используют двоичную систему счисления. Правила выполнения арифметических действий в этой системе счисления чрезвычайно просты (табл. 2.2), а следовательно, и просто реализуются в вычислительных машинах.

Таблица 2.2

Сложение	Вычитание	Умножение
0+0=0	0—0=0	0·0=0
1+0=1	1—0=1	1·0=0
0+1=1	10—1=1	0·1=0
1+1=10	1—1=0	1·1=1

При сложении в двоичной системе счисления двух чисел, равных единице, в данном разряде получается 0 и осуществляется перенос единицы в старший разряд, а при вычитании из нуля единицы осуществляется заем единицы из старшего разряда, отличного от нуля. Если в старшем разряде 0, то заем осуществляется из первого разряда. При этом единица, занятая в этом разряде, дает две единицы в младшем разряде и единицы во всех разрядах между данными и младшим.

Примеры.

Сложение	Вычитание	Умножение	Деление
110110	111001	1011	110010 | 1010
+ 101110	− 100111	× 1001	− 1010 |----
1100100	010010	1011	001010 101
		0000	1010
		0000	0000
		1011	
		1100011	

Из примеров видно, что умножение сводится к многократному сложению и сдвигам (если в данном разряде множимого записана 1, то осуществляется прибавление к промежуточной сумме множимого, сдвинутого на один разряд влево, если — 0, то нуля), а при выполнении деления используют правила умножения и вычитания.

Восьмеричная и шестнадцатиричная системы счисления. При подготовке задач для решения на ЦВМ часто используют восьмеричную и шестнадцатиричную системы счисления. Эти системы вспомогательные и применяются для записи команд программы решения задачи и некоторых констант при программировании в коде машины. Они удобны тем, что требуют соответственно в три и четыре раза меньше разрядов для записи команд и констант, чем двоичная система счисления.

В восьмеричной системе счисления для записи чисел используется восемь цифр: 0, 1, 2, 3, 4, 5, 6, 7 (см. табл. 2.1).

В шестнадцатиричной системе счисления для обозначения цифр используется шестнадцать различных символов. Поскольку в десятичной системе счисления имеется лишь десять символов, то для обозначения остальных шести символов применяют латинские заглавные буквы A, B, C, D, E и F (см. табл. 2.1).

Двоично-десятичный код. Двоично-десятичный код является вспомогательным и вычисления в нем, как правило, не ведутся. Для изображения десятичных цифр в этом коде используется четыре двоичных разряда (тетрада). Перевод в двоично-десятичный код очень прост и выполняется чисто механически на устройствах подготовки данных. При этом каждая десятичная цифра заменяется соответствующей ей двоичной тетрадой (см. табл. 2.1).

Например, десятичное число 49,8125 в двоично-десятичном коде будет иметь вид:

4	9,	8	1	2	5
0100	1001,	1000	0001	0010	0101

Как видно из этого примера, число, записанное в двоично-десятичном коде, отличается от соответствующего двоичного числа, несмотря на то, что представляется также с помощью нулей и единиц.

Ввиду того, что с помощью четырех двоичных разрядов можно представить шестнадцать различных чисел, правила выполнения арифметических действий в двоично-десятичном коде отличаются от правил выполнения действий в двоичной системе счисления и являются значительно более сложными.

Перевод чисел из двоично-десятичного кода в десятичную систему счисления происходит следующим образом. Двоично-десятичное число разбивается на тетрады, начиная от запятой влево и вправо, и каждая тетрада заменяется эквивалентной ей десятичной цифрой. Неполные крайние слева и справа тетрады дополняются до полных нулями. Например,

0001	0100	0111,	0100
1	4	7,	4,

что соответствует десятичному числу 147,4.

Итак, двоично-десятичный код используется для ввода исходных данных в машину. Перевод десятичных чисел в двоично-десятичный код в виде системы отверстий на носителе осуществляет устройство подготовки данных. Затем двоично-десятичный код самой машиной

по специальной программе переводится в двоичную систему. Результаты решения по специальной программе переводятся из двоичной системы в двоично-десятичный код и поступают на устройства выдачи результатов, где осуществляется переход от двоично-десятичного кода к десятичной системе счисления.

§ 2.2. ФОРМЫ ЗАПИСИ ЧИСЕЛ

В современных ЦВМ применяют естественную (с фиксированной запятой) или нормальную (с плавающей запятой) формы представления чисел.

В соответствии с этим вычислительные машины, использующие ту или другую форму представления чисел, и называют *машинами с фиксированной* или *плавающей запятой*. В некоторых современных машинах можно представлять числа в любой из этих форм.

При решении ряда задач экономико-статистического и счетно-финансового характера приходится иметь дело с величинами, имеющими лишь целочисленные значения. В этих случаях такие числа изображают в форме условно целых чисел, являющейся частным случаем формы с фиксированной запятой.

Естественная форма представления чисел. При естественной форме число представляется в виде целой части числа и отделенной от нее запятой дробной части. Место запятой постоянно фиксировано, т. е. для целой и дробной частей числа отводится вполне определенное количество разрядов.

Например, если для целой и дробной частей числа отводится по 3 десятичных разряда, то число 125,3 будет представляться в виде: +125,300, а число —3,721 — в виде: —003,721. Из примеров видно, что запятая, отделяющая целую часть от дробной, зафиксирована после третьего разряда. Очевидно, что в этом случае можно представлять числа по абсолютной величине, расположенные в диапазоне от 000,001 до 999,999. Все числа, меньшие 0,001, представляются в виде нуля, который называют *машинным нулем*. Наименьшее число 0,001 определяет точность представления чисел.

При выполнении арифметических действий над числами может получиться результат, больший наибольшего числа 999,999. Например,

$$\begin{array}{r} 905{,}374 \\ +\ 094{,}627 \\ \hline 1000{,}001 \end{array}$$

Так как для целой части числа отводится лишь три разряда, то в них будет записано 000. Следовательно, в этом случае результат оказывается неправильным. Такое явление называют *переполнением разрядной сетки*. Чтобы избежать указанного искажения результатов, прибегают к *масштабированию*.

Сущность масштабирования заключается в том, что перед решением задачи исходные данные умножают на соответствующие *масштаб-*

ные коэффициенты, подобранные таким образом, чтобы все промежуточные и окончательные результаты не превосходили по абсолютной величине максимально допустимого числа. Подбор масштабных коэффициентов — сложный и трудоемкий процесс, требующий большого опыта от программиста, подготавливающего задачу к решению. Необходимость выбора масштабных коэффициентов является наиболее существенным недостатком естественной формы представления чисел.

Обычно в ЦВМ запятая фиксируется перед старшим цифровым разрядом, т. е. используются числа, меньшие единицы. Это облегчает выбор масштабных коэффициентов, поскольку при выполнении операции умножения (наиболее опасной с точки зрения переполнения разрядной сетки) получают результат, всегда меньший единицы. Числа с фиксированной запятой представляются в виде группы цифр, следующих после запятой. Например, число +0,0275 будет представляться в виде $+\underbrace{027500\ldots0}_{n}$, где n — количество разрядов для представления числа без знака.

Нормальная форма представления чисел. Всякое число можно представить в виде произведения двух сомножителей:

$$N = mq^p,$$

где m — мантисса числа (дробное число, меньшее единицы); p — порядок числа (целое число); q — основание системы счисления (целое число).

Число N считается представленным в нормальной форме, если оно записано в виде произведения мантиссы m, для которой выполняется условие $|m| < 1$, и основания в степени p.

Например, число 3,25 в нормальной форме можно записать следующим образом:

$$3{,}25 = 0{,}325 \cdot 10^1;$$
$$3{,}25 = 0{,}0325 \cdot 10^2;$$
$$3{,}25 = 0{,}00325 \cdot 10^3 \text{ и т. д.}$$

Порядок числа может быть отрицательный, например, $0{,}00325 = 0{,}325 \cdot 10^{-2}$.

Порядок показывает положение запятой в числе. Величина порядка говорит, на сколько разрядов влево (в случае отрицательного порядка) или вправо (в случае положительного порядка) необходимо сдвинуть запятую в мантиссе числа, чтобы получить число в естественной форме.

Число 3,25 в нормальной форме может быть записано с различными мантиссами и порядками, равными 1, 2, 3 и т. д. Первое из этих чисел называется нормализованным числом, а остальные ненормализованными числами.

Условие нормализации числа N можно записать следующим образом:

$$(1/q) \leqslant |m| < 1,$$

где q — основание системы счисления.

У нормализованного числа первая цифра мантиссы отлична от нуля. У чисел ненормализованных одна или несколько первых цифр мантиссы являются нулями. Число в памяти машины желательно хранить в нормализованном виде, так как у ненормализованных чисел могут теряться младшие разряды мантиссы из-за конечного количества разрядов для ее записи. Для нормализации числа необходимо сдвинуть число влево на столько разрядов, чтобы старшая цифра мантиссы стала отличной от нуля, при этом на столько же единиц надо уменьшить порядок числа.

При выполнении арифметических действий над числами, представленными в нормальной форме, можно получить результат, мантисса которого больше единицы. Такое число можно нормализовать, сдвинув мантиссу вправо и увеличив порядок числа.

Таким образом, при нормальной форме представления чисел не происходит переполнение разрядной сетки машины, искажающее результат, а имеет место лишь нарушение нормализации, которое легко устраняется.

Нормализация чисел в машине производится как самой машиной автоматически после выполнения арифметических действий, так и по специальным командам.

В ЦВМ числа в нормальной форме записывают с помощью двух групп цифр, характеризующих мантиссу и порядок числа с соответствующими знаками. При этом мантисса записывается с помощью цифр, стоящих после запятой, т. е. нуль целых и запятая опускаются. Например, если для записи мантиссы отводится четыре разряда, а для записи порядка два разряда, то числа $+0{,}325 \cdot 10^{1}$ и $-0{,}325 \cdot 10^{-2}$ будут представляться в виде $+3250+01$; $-3250-02$ (первая группа цифр указывает цифры мантиссы после запятой, а вторая — порядок).

Диапазон представления чисел в нормальной форме шире, чем диапазон представления чисел в естественной форме, что является первым преимуществом нормальной формы представления чисел. Вторым ее преимуществом является то, что нет необходимости выбирать масштабные коэффициенты.

Недостатком нормальной формы по сравнению с естественной является необходимость дополнительных разрядов для хранения порядка. Кроме того, алгоритм выполнения арифметических действий над числами с плавающей запятой более сложный, что приводит к увеличению времени его выполнения и усложнению реализующей его аппаратуры.

Условно целые числа. Условно целые числа используют при решении экономико-статистических и счетно-финансовых задач. Эти числа изображаются в виде количества единиц самого младшего разряда. Если для изображения чисел с фиксированной запятой используется n разрядов, то условно целой единицей является единица младшего n-го разряда, т. е. вес условно целой единицы будет q^{-n}. Тогда целое число N равно количеству таких единиц, т. е.

$$x = Nq^{-n},$$

где x — условно целое число.

Условно целые числа записываются по тем же правилам, что и числа с фиксированной запятой. Например, целое число 17 в условно целой форме представляется как $x = 0{,}\underbrace{00\ldots}_{n}017$, где n — количество разрядов для записи числа с фиксированной запятой без знака. Таким образом можно представить любое целое число в диапазоне от 0 до $q^n - 1$.

Как и для случая с фиксированной запятой, при записи условно целого числа нуль целых и запятая не пишутся.

Рассмотренные примеры записи чисел в обеих формах представляли собой примеры записи десятичных чисел (в таком виде числа записываются на бланках исходных данных). По этим же правилам записи могут быть записаны числа в любой системе счисления.

§ 2.3. КОДИРОВАНИЕ АЛФАВИТНО-ЦИФРОВОЙ ИНФОРМАЦИИ

С помощью двоичных кодов можно закодировать и хранить в памяти машины любую алфавитно-цифровую информацию. Для этого каждому символу необходимо присвоить свой код.

Если кодировать информацию двоичными кодами, состоящими из n разрядов, то можно закодировать с их помощью 2^n различных символов. Следовательно, количество двоичных разрядов, необходимых для кодирования информации, зависит от количества различных кодируемых символов.

Необходимость в кодировании алфавитно-цифровой информации возникла в связи с решением на ЦВМ задач производственного и планово-экономического характера, программирование которых ведется на алгоритмических языках, использующих для записи программ не только цифры, но и буквы и некоторые специальные символы.

Рассмотрим принципы кодирования алфавитно-цифровой информации на примерах кодирования символов языков автокод «Инженер» (АКИ) и А Л Г О Л-60.

В языке АКИ в качестве символов входного языка используют символы второго международного телеграфного кода МТК-2, а в качестве устройства подготовки данных — стандартный телетайп СТА-2М, который переносит коды на стандартную пятидорожечную перфоленту.

Информация с помощью телетайпа СТА-2М кодируется на трех регистрах (цифровом, русском и латинском).

На пятидорожечной ленте записывается пятиразрядный двоичный код, с помощью которого можно закодировать лишь 32 различных символа. Поскольку общее количество символов значительно больше, то они разбиты на три группы (регистра). Каждому коду соответствует какой-либо символ в каждой группе, а принадлежность к той или иной группе задается с помощью кодов регистров.

В табл. 2.3 показано кодирование символов кода МТК-2.

Таблица 2.3

№ пп	Регистры			Комбинация перфораций	Двоичный код
	Латинский	Русский	Цифровой	1 2 СИ 3 4 5	
1	A	А	—	●●•	11000
2	B	Б	?	●•••●	10011
3	C	Ц	:	•●•●●	01110
4	D	Д	X	●•••●	10010
5	E	Е	З	●•••	10000
6	F	Ф	Э	●•●●●	10110
7	G	Г	Ш	•••●●	01011
8	H	Х	Щ	••●•●	00101
9	I	И	8	•●•●•	01100
10	J	Й	Ю	●●••●	11010
11	K	К	(	●●•●●	11110
12	L	Л	)	•●•••	01001
13	M	М	.	••●●●	00111
14	N	Н	,	••●●•	00110
15	O	О	9	••••●	00011
16	P	П	0	●•●•●	01101
17	Q	Я	1	●●•●●	11101
18	R	Р	4	●•••●	01010
19	S	С	Апостроф	●••●•	10100
20	T	Т	5	••••●	00001
21	U	У	7	●●•●•	11100
22	V	Ж	=	•●•●●●	01111
23	W	В	2	●●••●	11001
24	X	Ь	/	●•●●●	10111
25	Y	Ы	6	●•●•●	10101
26	Z	З	+	●•••●	10001
27	Возврат каретки		<	•••●•	00010
28	Перевод строки		)	••••••	01000
29	Латинский регистр			●●•●●●	11111
30	Цифровой регистр			●●••●●	11011
31	Пробел		⊔	••●••	00100
32	Русский регистр			••••••	00000

Многорегистровый способ кодирования требует малого количества разрядов для кодирования каждого символа, но усложняет работу по подготовке информации для ввода в машину и ее расшифровку.

Другим примером кодирования, когда каждому символу присваивается свой код, может служить кодирование символов языка АЛГОЛ-60 (табл. 2.4).

Таблица 2.4

Символ	Код	Символ	Код	Символ	Код
0	0000000	А	0100000	G	1000001
1	0000001	Б	0100001	I	1000010
2	0000010	В	0100010	J	1000011
3	0000011	Г	0100011	L	1000100
4	0000100	Д	0100100	N	1000101
5	0000101	Е	0100101	Q	1000110
6	0000110	Ж	0100110	R	1000111
7	0000111	З	0100111	S	1001000
8	0001000	И	0101000	U	1001001
9	0001001	Й	0101001	V	1001010
+	0001010	К	0101010	W	1001011
—	0001011	Л	0101011	Z	1001100
/	0001100	М	0101100	Надчерк	1001101
,	0001101	Н	0101101	⩽	1001110
.	0001110	О	0101110	⩾	1001111
␣	0001111	П	0101111	∨	1010000
10	0010000	Р	0110000	∧	1010001
↑	0010001	С	0110001	⊃	1010010
(	0010010	Т	0110010	¬	1010011
)	0010011	У	0110011	÷	1010100
X	0010100	Ф	0110100	≡	1010101
=	0010101	Х	0110101	%	1010110
;	0010110	Ц	0110110	◇	1010111
[	0010111	Ч	0110111	|	1011000
]	0011000	Ш	0111000	Гор. черта	1011001
*	0011001	Щ	0111001	Подчерк	1011010
'	0011010	Ы	0111010	!	1011011
,	0011011	Ь	0111011	Резерв.	1111010
≠	0011100	Э	0111100	Выброс карты	1111011
<	0011101	Ю	0111101	Конец строки	1111100
>	0011110	Я	0111110	Интерв. строк	1111101
:	0011111	D	0111111	Интерв. слов	1111110
		F	1000000	Выделение	1111111

На бланке алфавитно-цифровая информация записывается с помощью символов соответствующего языка, а в ячейке памяти — в виде двоичных кодов. В каждую ячейку записывается несколько кодов в последовательности, в которой они записаны на перфоленте или перфокарте.

§ 2.4. ОСОБЕННОСТИ КОДИРОВАНИЯ ИНФОРМАЦИИ В ВЫЧИСЛИТЕЛЬНЫХ МАШИНАХ ЕС ЭВМ

Вычислительные машины ЕС ЭВМ обрабатывают цифровую и алфавитно-цифровую информацию, содержащую, кроме цифр, буквы, знаки препинания, математические и другие символы.

Для кодирования такой информации в машинах первых поколений применялся шестиразрядный двоичный код (слог), с помощью которого можно было закодировать 64 различных символа, что в настоящее время уже недостаточно.

В последнее время в вычислительных машинах для представления алфавитно-цифровой информации используется восьмиразрядный двоичный код, называемый *байтом*. В табл. 2.5 приведен наиболее употребляемый в вычислительных машинах расширенный двоично-десятичный код EBCDIC.

Таблица 2.5

Номера разрядов 4567 \ 01 \ 23	0 00	0 01	0 10	0 11	01 00	01 01	01 10	01 11	10 00	10 01	10 10	10 11	11 00	11 01	11 10	11 11
0000					б пусто	bc	—						>	<	≠	0
0001							/		a	j			A	J		1
0010									b	k	s		B	K	S	2
0011									c	l	t		C	L	T	3
0100	PF	RES	BYP	PN					d	m	u		D	M	U	4
0101	HT	NL	LF	RS					e	n	v		E	N	V	5
0110	LS	BS	EDB	US					f	o	w		F	O	W	6
0111	DFL	IDL	PRE	EOT					g	p	x		G	P	X	7
1000									h	q	y		H	Q	Y	8
1001					•		,	"	i	r	z		I	R	Z	9
1010					?	!		:								
1011					.	$	,	#								
1100					←	*	%	@								
1101					(	)	m	'								
1110					+	;	_	=								
1111					‡	q	±	∨								

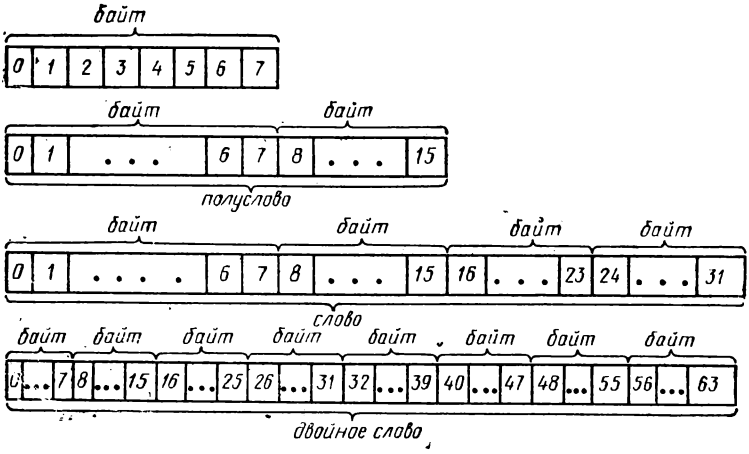
Байт является наименьшим адресуемым элементом памяти. Он состоит из восьми информационных разрядов (битов) и контрольного разряда. Контрольный разряд используется для проверки правильности передачи информации, например, по четности или нечетности.

Поскольку вычислительные машины ЕС ЭВМ являются машинами широкого назначения, то они должны оперировать с данными не толь-

ко фиксированной длины (как в машинах первых поколений), но и с данными переменной длины (полями переменной длины).

В зависимости от назначения, с учетом экономии памяти и требуемой точности вычислений, вычислительная машина может оперировать с данными фиксированной длины, состоящими из двух, четырех или восьми байтов. В зависимости от этого поля фиксированной длины называют соответственно *полусловом*, *словом* и *двойным словом*.

Информационные разряды нумеруют, начиная от нуля, от крайнего левого разряда:



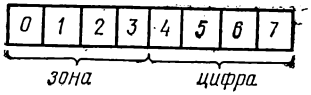
Каждый формат имеет адрес, совпадающий с адресом крайнего левого байта. Команда, использующая поля фиксированной длины, должна указывать не только адрес первого байта, но и определять длину используемых данных (фиксированную или переменную). Если данные имеют фиксированную длину, то необходимо также указывать, являются ли они словом, полусловом или двойным словом.

Следует помнить, что адрес каждого полуслова должен начинаться с четного байта, т. е. делиться на два, адрес каждого слова должен делиться на четыре, а двойного слова — на восемь, так как каждое полуслово составляется из двух смежных байтов, начиная с четного, слово — из двух смежных полуслов, а двойное слово — из двух смежных слов:

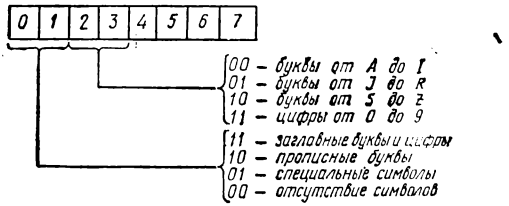
байт 0000	байт 0001	байт 0002	байт 0003	байт 0004	байт 0005	байт 0006	байт 0007
полуслово		полуслово		полуслово		полуслово	
слово				слово			
двойное слово							

Поля переменной длины могут иметь любое количество байтов и начинаться и кончаться любым адресом.

Для кодирования различных символов используется расширенный двоично-десятичный код (EBCDIC), в котором разряды байта распределяются следующим образом:

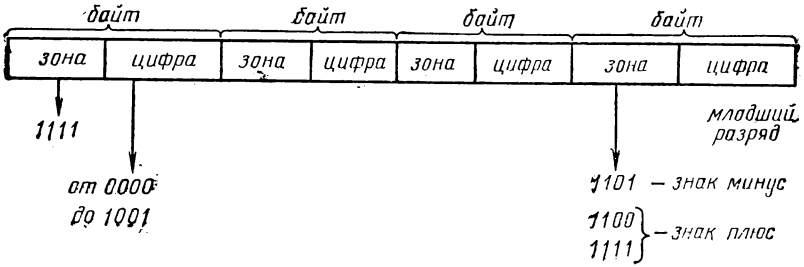


Код, записываемый в разрядах зоны, имеет служебный характер и указывает на тип символа следующим образом:

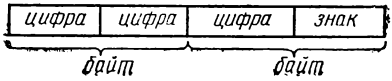


На основании табл. 2.3 нетрудно определить коды различных символов. Например: *A* — 1100 0001; *a* — 1000 0001; 7 — 1111 0111; = — 0111 1110.

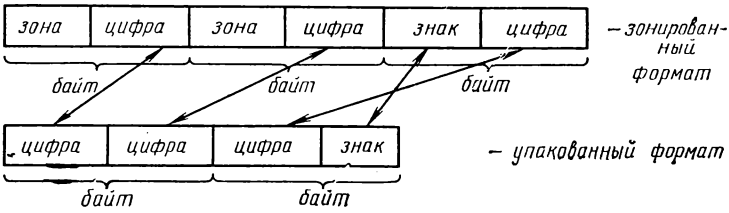
Числа в двоично-десятичном коде могут быть представлены в этом коде в зонированном или распакованном формате, который выглядит следующим образом:



Однако при таком формате не экономно расходуется память и уменьшается скорость обработки данных, поэтому в вычислительных машинах ЕС ЭВМ используется упакованный формат, который содержит по две цифры в каждом байте, а в младшем байте одну цифру (разряды с нулевого по третий) и знак числа:



Вычислительные машины ЕС ЭВМ могут преобразовывать числа из одного формата в другой по следующей схеме:

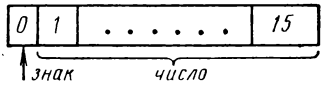


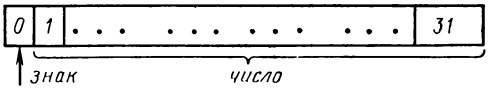
Если действия производятся над десятичными числами, то последние должны быть представлены в упакованном формате.

Десятичные данные и данные в коде EBCDIC имеют переменную длину, зависящую от количества символов. Перед выполнением действий над ними переход к двоичной системе не происходит.

Числа в двоичной системе счисления (двоичные данные) имеют фиксированную длину.

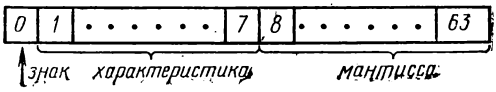
В естественной форме их длина — полуслово или слово. В обоих случаях знак записывается в крайнем левом разряде, а в остальные разряды записывается само число:





Числа в этой форме используют, как правило, для представления условно целых чисел.

Длина двоичных данных в нормальной форме составляет слово или двойное слово. В крайнем левом разряде записывается знак числа, в разрядах с первого по седьмой — характеристика (порядок), в остальных — мантисса числа:



Характеристика числа представляется числами от 0 до 127 и определяется с помощью выражения $x = 64 + p$, где p — порядок числа. Например, если порядок числа равен — 17, то характеристика такого числа будет $x = 64 + p = 64 - 17 = 47$.

ГЛАВА 3

ПОДГОТОВКА ЗАДАЧ ДЛЯ ПРОГРАММИРОВАНИЯ

Процессу решения задачи на ЦВМ предшествуют следующие этапы подготовки программы.

1. Математическая формулировка задачи.
2. Разработка алгоритма решения задачи: а) выбор метода вычислений для решения задачи; б) разработка схемы алгоритма.
3. Составление программы на алгоритмическом языке.

При решении конкретных задач некоторые из этих этапов могут быть исключены постановкой задачи, поскольку приведенная последовательность этапов носит условный характер, хотя смысл перечисленных действий в процессе подготовки задачи сохраняется.

§ 3.1. МАТЕМАТИЧЕСКАЯ ФОРМУЛИРОВКА ЗАДАЧИ

На этапе математической формулировки задачи математически описывается ее условие, определяются аналитические выражения и формулы.

Рассмотрим несколько примеров.

Задача 3.1. Определить высоту треугольника x по заданной площади c, если известно, что основание больше высоты на величину b.

Известно, что площадь треугольника равна половине произведения высоты на основание, т. е.

$$c = x\,(x+b)/2.$$

Отсюда x определяется как корень уравнения

$$x^2 + bx - 2c = 0. \tag{3.1}$$

Таким образом, математическая формулировка задачи сводится к отысканию действительного положительного корня квадратного уравнения (3.1).

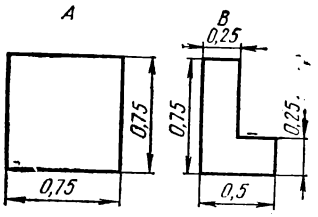


Рис. 3.1. Размеры деталей

Задача 3.2. На завод поступают прямоугольные листы размером 1 × 3 м, из которых надо изготовить 1000 заготовок типа A и 6000 заготовок типа B (рис. 3.1). Необходимо таким образом раскроить листы, чтобы получить минимальный отход материала.

Листы такого размера можно раскроить несколькими способами, основные из которых представлены на рис. 3.2. Введем следующие обозначения: x, y, z, u, v — количество листов, раскроенных соответственно по схемам, показанным на рис. 3.2. *а*, *б*, *в*, *г*, *д*.

Для каждого из этих вариантов раскроя из одного листа можно изготовить различное количество деталей типа A и B, при этом получим различную величину отхода (табл. 3.1).

Обозначим через a_{1i} и a_{2i} соответственно количество заготовок типов A и B, получаемых из одного листа по i-му варианту раскроя.

Тогда

$$1000 = a_{11}x + a_{12}y + a_{13}z + a_{14}u + a_{15}v,$$
$$6000 = a_{21}x + a_{22}y + a_{23}z + a_{24}u + a_{25}v.$$

Подставив значения этих коэффициентов, получим:

$$0x+1y+2z+3u+4v=1000,$$
$$12x+9y+7z+4u+0v=6000.$$

Откуда

$$v=250-y/4-z/2-u/4,$$
$$x=500-3y/4-7z/12-u/3.$$

При этом надо так выбирать x, y, z, u и v, чтобы отход материала был минимальным

$$c=\min(c_1x+c_2y+c_3z+c_4u+c_5v),$$

где c_i — отход материала от одного листа, раскроенного по i-му варианту.

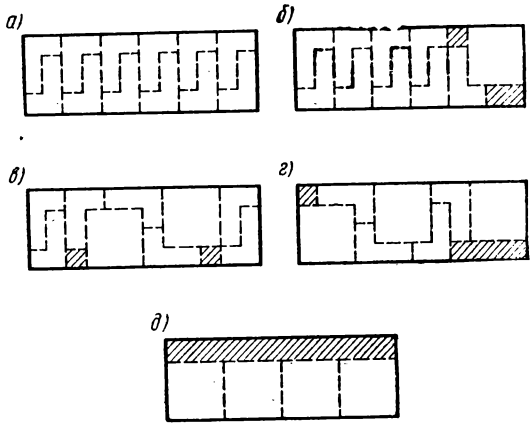


Рис. 3.2. Схемы раскроя материала

Таким образом, если не считать в качестве отхода часть листа, из которой может быть изготовлена одна или несколько заготовок того или другого типа, то величина отхода

$$c=0x+3y/16+z/8+5u/16+3v/4.$$

При решении этой задачи надо учитывать, что все переменные x, y, z, u и v больше или равны нулю.

Таблица 3.1

Вариант раскроя	Количество заготовок из одного листа, шт.		Отход, м²
	A	*B*	
а	0	12	0
б	1	9	3/16
в	2	7	1/8
г	3	4	5/16
д	4	0	3/4

Математическая формулировка подобных задач такова: минимизировать функцию

$$c=3y/16+z/8+5u/16+3v/4,$$

при следующих ограничениях, наложенных на аргументы:

$$\begin{cases} y+2z+3u+4v=1000, \\ 12x+9y+7z+4u=6000, \end{cases}$$
$$x\geqslant 0;\quad y\geqslant 0;\quad z\geqslant 0;\quad u\geqslant 0;\quad v\geqslant 0.$$

§ 3.2. РАЗРАБОТКА АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ

Алгоритмом называют систему формальных правил, однозначно приводящую к решению данной задачи. Применительно к решению задач на ЦВМ алгоритм представляет собой последовательность арифметических и логических действий над числовыми значениями

переменных, приводящую к вычислению решения задачи при изменениях исходных данных в достаточно широких пределах.

Алгоритм должен обладать:

детерминированностью (в силу полной однозначности правил применение алгоритма к одним и тем же исходным данным должно приводить к одному и тому же результату);

массовостью (алгоритм представляет собой последовательность действий для получения результата при различных исходных данных для некоторого класса задач);

результативностью (при применении алгоритма реализуется конечный процесс вычислений, который заканчивается либо получением некоторого искомого результата, либо сигналом о том, что данный алгоритм неприменим к имеющимся исходным данным).

Для составления алгоритма и программы решения задач на ЦВМ математическая формулировка задачи, включающая символы математического анализа (символы интеграла, производной, дифференциальных операторов, конечных разностей и т. д.), должна быть преобразована непосредственно в процедуру решения задачи, представляющую собой последовательность арифметических действий и логических связей между ними, т. е. должен быть выбран метод решения задачи.

Например, при решении квадратного уравнения $ax^2 + bx + c = 0$ вычисление корней производится по формуле

$$x_{1,2} = \left(-b \pm \sqrt{b^2 - 4ac}\right) / (2a),$$

а значение определенного интеграла $y = \int_a^b (x^3 + v)dx$ может быть вычислено по формуле

$$y = b^4/4 - a^4/4 + v\,(b - a).$$

В этих формулах показаны способы решения поставленных задач на основе последовательности арифметических действий над переменными a, b, c, v, поэтому они могут составить основу алгоритмов решения подобных задач.

Недостаток формул заключается в том, что они не обладают достаточной массовостью. Действительно, для алгебраических уравнений степени $n \geqslant 5$ не существует общих формул, выражающих связь между коэффициентами уравнений и их корнями в радикалах; интеграл $y = \int_a^b f(x)\,dx$ вычисляется по приведенной формуле только для $f(x) = x^3 + v$, для большинства же $f(x)$, встречающихся на практике, либо формулы для первообразной функции весьма громоздки, либо первообразная вообще не может быть выражена через элементарные функции.

В большинстве случаев при решении нелинейных дифференциальных уравнений с переменными коэффициентами применяют приближенные методы, поскольку неизвестны точные формулы замкнутых решений для уравнений подобного рода.

Перечень задач, для которых вычислительные формулы для получения точных решений не могут быть получены обычными математическими приемами, охватывает практически все задачи математического анализа. Разработка методов арифметизации задач, называемых *численными методами*, относится к вычислительной математике.

Численные методы в большинстве своем приближенные, приспособлены для решения широких классов задач и обеспечивают достаточно высокую точность решения последних. Применение приближенных численных методов оказывается более предпочтительным даже в тех случаях, когда известен точный способ решения задачи, поскольку достаточная точность и небольшие затраты времени (при применении ЦВМ) позволяют получить практически ценные результаты, не прибегая к громоздким выкладкам.

Примером численного метода является, например, метод прямоугольников для приближенного интегрирования, не требующий вычисления первообразной (неопределенного интеграла) для подынтегральной функции:

$$\int_a^b f(x)\,dx \cong \sum_{i=1}^{n} f(x_i)\,\Delta x_i, \qquad (3.2)$$

где $x_1 = a$ (нижний предел интегрирования); $x_{n+1} = b$ (верхний предел интегрирования); n — число отрезков, на которые разбит интервал интегрирования (a, b); Δx_i — длина элементарного отрезка; $f(x_i)$ — значения подынтегральной функции $f(x)$ на концах элементарных отрезков интегрирования x_i.

Вычисление определенного интеграла сводится, таким образом, к приближенному определению площади, ограниченной осью абсцисс, кривой подынтегральной функции и ординатами, восстановленными из концов интервала интегрирования.

Выбор метода вычисления. Практически для любой математической задачи в инженерной практике или в экономических расчетах разработаны численные методы решения. Более того, многие задачи могут быть решены с помощью различных численных методов и вычислительных методик. Выбор того или иного численного метода для решения задачи на ЦВМ связан, с одной стороны, с требованиями, предъявляемыми постановкой задачи (точность решения, быстрота получения результата, стоимость подготовки программы решения задачи), и, с другой стороны, с требованиями, предъявляемыми ЦВМ и программой к численному методу для его реализации на машине.

Алгоритм, на основе которого составляется программа решения задачи, представляет собой последовательность действий, которые может выполнить (или может выполнить наиболее рационально) ЦВМ. В нем отражаются не только арифметические действия, необходимые для реализации выбранного численного метода, но и логические связи, которые численный метод налагает на исходные данные. Эти логические связи должны быть заданы в форме, воспринимаемой вычислительными машинами — в форме проверки тех или иных соот-

ношений, допускающих (или не допускающих) автоматическое выполнение действий, предписываемых машине программой.

К логическим условиям, не допускающим автоматическую работу ЦВМ, относят: 1) невозможность деления на нуль или на разность близких чисел (при делении на нуль получается результат, не имеющий численного выражения (∞); при делении на разность близких чисел возможно возникновение чисел, превышающих диапазон представления их в машине); 2) невозможность представления $\ln 0 (-\infty)$ и логарифмов отрицательных чисел; 3) отсутствие смысла при вычислениях $\arcsin x$, $\arccos x$ для $|x| > 1$; 4) отсутствие в машине внутреннего способа представления мнимых и комплексных чисел.

Алгоритм вычислений на ЦВМ может строиться не как развернутая, а как циклическая последовательность действий с многократным повторением отдельных его участков. При этом уменьшаются трудоемкость составления программы и используемый объем памяти. А это означает, что численный метод предпочтителен, если связанные с ним формулы вычислений позволяют максимально использовать цикличность при вычислениях. Численному методу, обеспечивающему многократное использование тех или иных участков алгоритмов для различных целей, может быть отдано предпочтение перед другими численными методами, поскольку такие многократно используемые участки алгоритма могут быть выделены в подпрограммы и однократно включены в описание алгоритма.

Необходимо учитывать, что при вычислениях на ЦВМ используемые формулы должны обеспечить рациональное время решения задачи за счет отыскания оптимального соотношения между затратами времени на вычисления и подготовку программы и заданной точностью. Так, для вычисления определенного интеграла по формуле (3.2) точность вычисления пропорциональна шагу интегрирования $h = \Delta x$, следовательно, к увеличению точности в два раза приводит уменьшение в два раза шага $h^* = h/2$ и увеличение в два раза числа элементарных отрезков n и времени интегрирования.

Меньшую зависимость точности от шага интегрирования имеет метод парабол, для которого увеличение точности в два раза достигается уже при уменьшении h в $\sqrt[4]{2}$ раз за счет некоторого усложнения расчетных формул. Поэтому метод парабол более распространен в вычислительной практике.

§ 3.3. СПОСОБЫ ОПИСАНИЯ СХЕМ АЛГОРИТМОВ

Из существующих способов описания алгоритмов наиболее часто используют описания алгоритмов в виде блок-схемы и с помощью символов-операторов.

Описание алгоритмов в виде блок-схем. Алгоритм в этом случае представляется графически в виде последовательности блоков, выполняющих определенные функции. Блоки соединяются стрелками, показывающими связи между ними. Внутри блоков указывается информация, характеризующая выполняемые ими функции, которые

записываются словесно или с помощью формул. Все блоки блок-схемы имеют сквозную нумерацию.

Рассмотрим несколько примеров составления блок-схем для вычислительных процессов различного характера.

Линейный вычислительный процесс. Линейный вычислительный процесс — это процесс, блоки которого выполняются последовательно один за другим (порядок выполнения блоков естественный).

Задача 3.3. Составить блок-схему для линейного вычислительного процесса.

Пусть необходимо определить

$$Z = \frac{\sin(Ax^2 - Bx - C) + \cos(-Ax^2 + Bx + C)}{2(Ax^2 - Bx - C)}.$$

Вычисление функции Z можно описать следующим образом. Сначала надо найти значение числителя. Затем, после вычисления значения знаменателя, разделить на него значение числителя. Однако такой алгоритм не является единственным и оптимальным, поскольку нет надобности вычислять три раза значение выражения, заключенного в скобки. Целесообразнее сначала вычислить значение аргумента синуса (обозначим его через y), взять от него синус, затем косинус от y с обратным знаком и, наконец, получив сумму, разделить ее на $2y$. Затраты машинного времени на реализацию такого алгоритма будут меньшими.

Блок-схема алгоритма решения этой задачи представлена на рис. 3.3. Поскольку исходные данные вводятся в машину в двоично-десятичном коде, а машина решает задачу в двоичной системе счисления, то перед решением задачи необходимо осуществить перевод исходных данных в двоичную систему счисления.

Обратный перевод результата вычисления в двоично-десятичный код осуществляется перед выводом на печать. На рис. 3.3 блоки, выполняющие эти функции, не показаны.

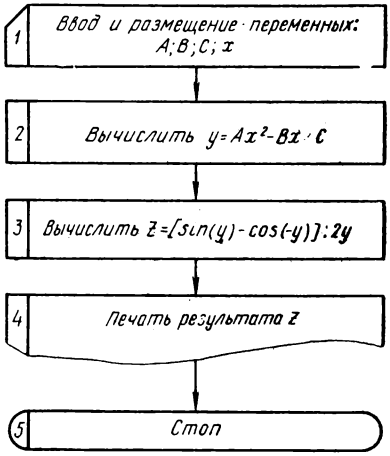


Рис. 3.3. Блок-схема алгоритма линейного вычислительного процесса

Разветвляющийся вычислительный процесс. На практике часто возникает необходимость в зависимости от полученных промежуточных результатов осуществлять вычисление по одним или другим формулам, т. е. в зависимости от выполнения какого-то логического условия вычислительный процесс должен идти по одной или другой ветви. Такой вычислительный процесс называют разветвляющимся.

Задача 3.4. Составить блок-схему разветвляющегося вычислительного процесса для решения квадратного уравнения.

Корни уравнения $ax^2 + bx + c = 0$ находят из выражения

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Квадратный корень из отрицательного числа ЦВМ вычислять не может, поэтому для комплексных корней $x_{1,2} = \alpha \pm i\beta$ отдельно вычисляют действительную часть α и коэффициент при мнимой единице β ($i = \sqrt{-1}$).

Тогда алгоритм формулируется следующим образом.

Вычислить

$$\begin{cases} x_{1,2} = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}, & \text{если } D = b^2 - 4ac \geqslant 0; \\ \alpha = \dfrac{-b}{2a}, \quad \beta = \dfrac{\sqrt{|b^2 - 4ac|}}{2a}, & \text{если } D = b^2 - 4ac < 0. \end{cases}$$

Такой вычислительный процесс имеет две ветви. В первой ветви, если выполняется условие $D \geqslant 0$, вычисляются x_1 и x_2, во второй ветви, если $D < 0$, — действительная часть α и коэффициент при мнимой единице β. После выполнения любой из этих ветвей снова возвращаются в общую последовательность блоков, т. е. печатают результаты и останавливают машину.

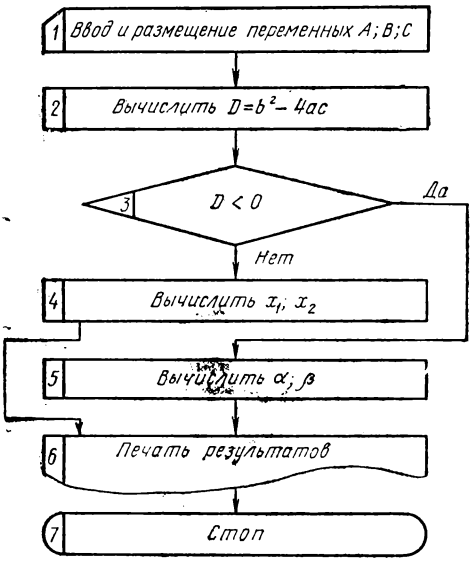


Рис. 3.4. Блок-схема алгоритма разветвляющегося вычислительного процесса

Блок-схема алгоритма такого вычислительного процесса представлена на рис. 3.4. В этом алгоритме естественный порядок выполнения блоков нарушается дважды:

1. После выполнения блока *3*, если выполняется условие $D < 0$, переходят сразу к блоку *5*, в противном случае — к блоку *4*. Такой переход называют *условным*, так как он осуществляется только при выполнении какого-то логического условия.

2. После выполнения блока *4* (вычисление действительных корней) нет смысла вычислять действительную часть и коэффициент при мнимой единице (выполнять блок *5*), поэтому всегда (безусловно) надо обходить блок *5* и переходить к следующему блоку общей последовательности, т. е. к блоку *6*.

Циклический вычислительный процесс. Такие процессы часто встречаются на практике, когда решение задачи сводится к многократному вычислению по одним и тем же математическим зависимостям при различных значениях входящих в них величин. Многократно повторяющиеся участки этого вычислительного процесса называют *циклами*. Циклический алгоритм позволяет существенно сократить объем программы за счет многократного выполнения ее циклического участка.

Задача 3.5. Составить блок-схему циклического вычислительного процесса.

Пусть надо вычислить сто значений функции

$$y_i = \frac{\sin(x_i \alpha)}{x_i} \qquad (x_i = 1, 2, \ldots, 100).$$

Очевидно, что для вычисления всех значений функции y_i необходимо сто раз вычислять по этой формуле значения y, изменяя каждый раз аргумент x_i на единицу. Цикл будет повторяться до тех пор, пока $x_i \leqslant 100$. Если x_i станет больше 100, то будет осуществлен выход из цикла, т. е. переход к следующему по порядку блоку.

Блок-схема алгоритма решения этой задачи представлена на рис. 3.5. В этой блок-схеме требуют пояснения лишь блоки *2*, *5* и *6*.

Блок *2* подготавливает цикл, т. е. задает начальное значение аргумента. В этом примере можно обойтись без блока *2*, если ввести начальное значение $x_1 = 1$ и шаг изменения аргумента также равный единице в исходных данных, но во многих случаях целесообразно поступать так, как показано на блок-схеме.

Блок *5* осуществляет изменение аргумента x_i на единицу после каждого выполнения цикла, что требуется по условию задачи.

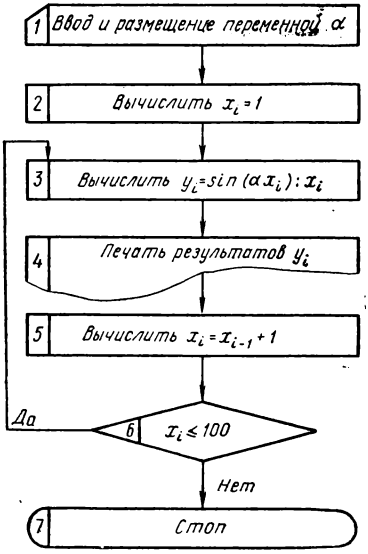


Рис. 3.5. Блок-схема алгоритма циклического вычислительного процесса

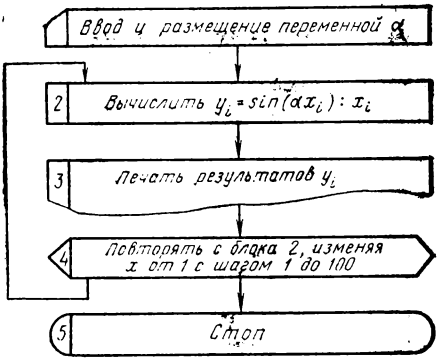


Рис. 3.6. Блок-схема алгоритма циклического вычислительного процесса

Блок *6* управляет циклом, для чего проверяется условие повторения цикла $x_i \leqslant 100$. При выполнении этого условия цикл должен повторяться, а при невыполнении осуществляется выход из цикла, т. е. переход к следующему по порядку блоку.

Для организации любого цикла необходимы блоки, выполняющие следующие функции: 1) задание начального значения переменной, изменяющейся в цикле (подготовка цикла); 2) изменение переменной перед каждым новым повторением цикла; 3) проверка условия окончания цикла и выход из него, если цикл закончен; 4) переход к началу цикла (управление циклом), если цикл не закончен.

В рассмотренном примере задание начального значения переменной осуществляет блок *2*, ее изменение — блок *5*, а функции проверки условия окончания цикла и управления циклом — блок *6*. Следовательно, отдельные блоки могут выполнять несколько функций, из перечисленных выше. Если один блок выполняет все указанные функции, то блок-схему алгоритма удобно представлять так, как показано на рис. 3.6.

Задача 3.6. Пусть необходимо вычислить значения функции

$$z_{ij} = x_i y_j,$$

где $x_i = x_1, x_2, x_3, \ldots, x_{10}$; $y_j = y_1, y_2, y_3, \ldots y_{20}$.

Результатом решения этой задачи будут числа, которые можно представить следующей матрицей:

$$\begin{Vmatrix} x_1 y_1 & x_1 y_2 & x_1 y_3 \dots x_1 y_{20} \\ x_2 y_1 & x_2 y_2 & x_2 y_3 \dots x_2 y_{20} \\ \dots & \dots & \dots \dots \dots \\ x_{10} y_1 & x_{10} y_2 & x_{10} y_3 \dots x_{10} y_{20} \end{Vmatrix}$$

Если в одном цикле вычислить элементы одной строки при одном значении x_i и при всех возможных зачениях y_j, а в другом цикле изменить значение i и повторить снова цикл, вычисляющий элементы строки, то в результате можно получить алгоритм, включающий в себя два вложенных один в другой цикла. Такой цикл называют *сложным циклом*.

Блок-схема программы со сложным циклом приведена на рис. 3.7.

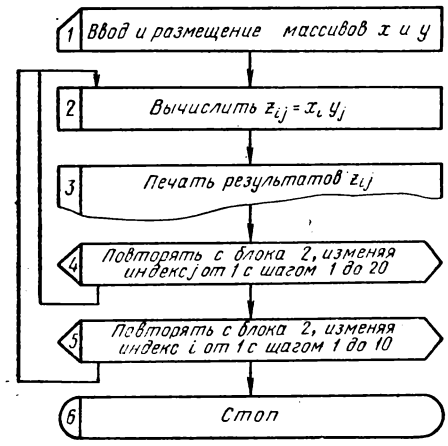


Рис. 3.7. Блок-схема алгоритма двойного цикла

Операторная запись алгоритмов. При такой записи алгоритмов каждый блок, выполняющий определенную функцию, обозначается с помощью специального символа-оператора. Операторы различных типов изображаются буквами, например, арифметические операторы буквой А, операторы ввода — буквой В, оператор печати — буквой П и т. д. Каждый оператор имеет индекс, соответствующий его порядковому номеру (нумерация операторов сквозная для всего алгоритма).

Правила записи алгоритмов в операторном виде следующие:

1. Операторы записываются в строку слева направо в той последовательности, в которой они должны выполняться.

2. Если после данного оператора управление никогда не передается к следующему за ним оператору, то между ними ставится точка с запятой.

3. При нарушении естественного порядка выполнения операторов передача управления указывается стрелкой.

Запись в операторной форме алгоритма решения квадратного уравнения (см. задачу 3.4) следующая:

$$B_1 A_2 A_3 P_4 A_5; \quad A_6 A_7 П_8 Я_9,$$

где B_1 — оператор ввода программы и исходных данных; A_2 — оператор перевода исходных данных из двоично-десятичной системы в двоичную; A_3 — оператор, вычисляющий $D = b^2 - 4ac$; P_4 — логический оператор, проверяющий выполнение условия $D < 0$; A_5 — оператор, вычисляющий действительные корни x_1 и x_2; A_6 — оператор, вычисляющий действительную часть α и коэффициент при

мнимой единице β комплексно-сопряженных корней; A_7 — оператор перевода результатов из двоичной системы в двоично-десятичный код; $П_8$ — оператор печати результатов; $Я_9$ — оператор останова машины.

Операторный способ записи алгоритма более компактен, чем с помощью блок-схем, но менее нагляден, поэтому используется реже.

§ 3.4. ОСОБЕННОСТИ ПРОГРАММИРОВАНИЯ ДЛЯ ЦВМ

Специфика работы ЦВМ вызывает ряд особенностей в способах описания вычислительных процессов как алгоритмическими, так и рабочими программами. Рассмотрим основные из этих особенностей.

Представление чисел при программировании. В ЦВМ числа представимы в двух качественно различных формах. Определены в ЦВМ числа действительные и целые, которые в математическом смысле принадлежат к множеству действительных чисел и могут принимать положительные, отрицательные и нулевые значения. Комплексные и мнимые числа в ЦВМ не определены. Возможность их возникновения в процессе вычислений, как результата извлечения корней четных степеней из отрицательных действительных чисел, вызывает прекращение автоматических вычислений и поэтому должна быть предусмотрена для сохранения непрерывности счета уже в процессе составления программного алгоритма (см. задачу 3.4). Для представления мнимых и комплексных чисел используют действительные числа. Так, комплексное число $\alpha + i\beta$ должно быть представлено в ЦВМ парой действительных чисел (α, β).

Действительные числа представляют в ЦВМ с указанием мантиссы и порядка. Например, в нормальной форме число $+27{,}3$ представляют в виде $+0{,}273 \cdot 10^{+2}$, где $0{,}273$ — мантисса числа, $+2$ — порядок числа. В ЦВМ это число представляют в форме $+0{,}273+02$.

Форма представления чисел с переменными порядками позволяет производить арифметические действия над числами с абсолютной величиной от $1 \cdot 10^{-19}$ до $1 \cdot 10^{+19}$. Поскольку действия над мантиссами и порядками ЦВМ в этом случае производят автоматически, то такие вспомогательные действия, как выравнивание порядков слагаемых при сложении, сложение порядков при умножении, вычитание порядков действительных чисел при делении не нужно учитывать в программе.

Целые числа — форма представления только целых чисел. Представление в ЦВМ целых чисел обеспечивает следующие преимущества при организации вычислений.

1. Скорость выполнения арифметических операций над целыми числами выше, чем скорость выполнения операций над действительными числами, требующими выполнения вспомогательных операций. Это свойство целых чисел используют при организации экономических расчетов, где единицами измерений являются рубли, тонны, километры и дробные части чисел не имеют существенного значения.

2. Перевод целых чисел из десятичной системы счисления в двоичную и обратно происходит без погрешностей, тогда как перевод действительных чисел, как правило, осуществляется неточно, с недостатком. Это свойство целых чисел используют при организации в программах «счетчиков» целых количеств, причем целые числа позволяют организовать анализ содержимого таких счетчиков на основе проверки на «равно» или «неравно», что нельзя осуществить для «счетчиков» с действительными числами, поскольку неучитываемые погрешности перевода чисел из одной системы счисления в другую и ошибки вычислений исключают соблюдение точных равенств.

3. При выводе результатов из ЦВМ целые числа печатают на бланках в форме, отличной от формы представления действительных чисел. Это свойство целых чисел используют для наглядного разделения групп числовых результатов, облегчающего их интерпретацию и обработку. При этом целые числа являются «признаками», характеризующими связанную с ними группу результатов счета, представленных в форме действительных чисел.

4. В целых числах возможно представление мантисс и порядков действительных чисел. При этом и мантисса и порядок действительного числа рассматриваются как пара отдельных чисел, что предполагает при выполнении арифметических операций составление особой программы выполнения каждого арифметического действия, включающего вспомогательные действия.

Например, при сложении двух чисел A и B, представленных в виде пар целых чисел a, p_a и b, p_b, где a, b — мантиссы; p_a и p_b — порядки, задается следующая последовательность действий:

1) сравнить порядки p_a и p_b; если они одинаковы, то сложить мантиссы и получить мантиссу результата $c = a + b$; порядок результата $p_c = p_a$ (сумма C при этом получится в виде пары чисел (c, p_c)), иначе выполнить пункт 2;

2) определить, какой из двух порядков p_a или p_b больше и на сколько; мантиссу числа с меньшим порядком умножить на $10^{-|p_a - p_b|}$ (это позволяет выравнить порядки слагаемых); мантиссы a^* и l^* (одна из них сдвинута) сложить, результату приписать порядок бо́льшего числа p_c ($p_c = p_a$, если $p_a > p_b$ или $p_c = p_b$, если $p_a < p_b$).

Такой подход применяют в случаях, когда числа, с которыми должна оперировать программа, превышают по абсолютной величине конечный диапазон представимых в ЦВМ действительных чисел.

Для большинства инженерных расчетов вполне достаточно при вычислениях использовать действительные числа, в силу чего целые числа применяют чаще всего для целей, указанных в пп. 1—3.

Представление переменных в ЦВМ. ЦВМ оперирует с числами, каждое из которых хранится в отдельной ячейке.

При математической формулировке задачи оперируют с буквенными переменными и буквенными или числовыми коэффициентами и константами.

Поскольку коэффициенты и константы имеют одно числовое значение, то для их хранения достаточно одной ячейки. Переменные

в ЦВМ представляются конечным количеством чисел, определяющих их значения в некоторые моменты времени, так как ЦВМ является машиной дискретного типа. Другими словами, переменная в ЦВМ представляется некоторой последовательностью ее числовых значений. Например, если переменная x изменяется в интервале от 0 до 1, то она может быть представлена следующей последовательностью чисел:

0; 0,1; 0,2; 0,3; 0,4; 0,5; 0,6; 0,7; 0,8; 0,9; 1.

Здесь значения переменной взяты с постоянным шагом изменения, равным 0,1.

Иногда необходимо представить переменные в виде последовательности, при которой в определенном месте интервала числа задаются более часто. Например, если надо определить значения переменной около нуля, то такая переменная может быть представлена в виде следующей последовательности чисел:

0; 0,005; 0,01; 0,02; 0,05; 0,1; 0,2; 0,3; 0,5; 0,7; 1.

Первую последовательность можно хранить в памяти машины двумя способами.

Способ 1. В памяти машины отводится 11 ячеек, в каждой из которых можно хранить одно число. При этом способе хранения не экономно используется память машины. Особенно сильно сказывается недостаток этого способа хранения тогда, когда имеется много переменных, каждая из которых задается большим количеством своих числовых значений.

Способ 2. Переменную задают пределами изменения и законом изменения ее величины. Тогда для хранения всех значений переменной достаточно одной ячейки, в которую заносят сначала ее начальное значение, а каждое новое значение получают путем прибавления к содержимому этой ячейки шага изменения (здесь шаг изменения равен 0,1), т. е. все значения переменной вычисляются самой машиной по формуле $x_{i+1} = x_i + 0{,}1$ и хранятся последовательно в одной и той же ячейке памяти машины.

Такой способ хранения числовых значений переменных более экономичен с точки зрения требуемого объема памяти ЦВМ, но он имеет тот недостаток, что в каждый момент времени в памяти машины хранится лишь одно числовое значение переменной. Поэтому этот способ хранения не всегда удается применить в тех случаях, когда для решения задачи требуется одновременно использовать несколько числовых значений одной и той же переменной. Также не всегда есть возможность найти закон изменения последовательностей числовых значений переменной, в общем случае описываемый любой функциональной зависимостью (например, вторая последовательность числовых значений переменной, а также последовательность, состоящая из каких-либо случайных чисел).

Таким образом, значения переменных в ЦВМ могут храниться либо в массиве последовательных ячеек (для каждого значения своя ячейка), либо по очереди в одной и той же ячейке.

В зависимости от способа хранения различают простые переменные и переменные с индексами.

Простые переменные — переменные, все значения которых вычисляются самой машиной по заданным ей граничным значениям и закону изменения и хранятся по очереди в одной и той же ячейке памяти машины.

Переменные с индексами — переменные, которые заданы всеми своими значениями и хранятся в массиве последовательных ячеек памяти машины (при переходе от одного значения переменной к другому необходимо изменять индекс переменной). Каждое значение переменной с индексом является элементом массива. Переменные с индексами используют для обозначения элементов не только одномерных массивов, но и многомерных, в последнем случае эти переменные должны иметь несколько индексов.

В языках программирования применяют различные способы записи как переменных с индексами, так и самих массивов.

Таким образом, чтобы избежать ошибок при программировании, необходимо всегда помнить, что понятия простых переменных и переменной с индексами, используемые при программировании, имеют несколько иной смысл, чем в математике. Например, пусть требуется вычислить значения функции $y = f(x)$ при различных значениях аргумента $x_1, x_2, \ldots, x_n$. В математике для обозначения как аргумента x, так и функции y не используют индексы. Если же все значения аргумента x заданы массивом значений, для которых не удалось найти закона их изменения, то при программировании надо считать x переменной с индексом, который указывает номер значения этой переменной в массиве. Переменную y определяют как простую переменную, если не требуется запомнить все ее значения в массиве ячеек, или как переменную с индексами в противном случае.

Пусть требуется определить значения функции $y_i = f(x_i, x_{i+1})$, т. е. функции, зависящей от двух последовательных значений аргумента x. В математике в этом случае и функция y_i, и аргументы x_i и x_{i+1} обозначаются как переменные с индексами. Если все значения x заданы пределами изменения от 0 до 1 и шагом изменения 0,1, то при программировании переменную x_i целесообразно изображать простой переменной, а функцию, как и в предыдущем примере, как простой переменной, так и переменной с индексами. Поскольку переменная x_i отличается от переменной x_{i+1} на величину шага 0,1, то вместо переменной x_{i+1} следует записывать $x + 0{,}1$.

Из этих примеров видно, что понятие простой переменной и переменной с индексами в математике и при программировании могут и не совпадать.

Запоминание результатов в памяти машины. В рассмотренных выше примерах все численные значения результата вычисляли, как значения простых переменных, т. е. поочередно записывали их в одну и ту же ответную ячейку. Если же необходимо сохранить в памяти машины все значения результата, то для каждого из них надо выделить отдельную ячейку и записать в нее соответствующее числовое значение переменной. Для реализации этого необходимо: 1) выделить в памяти

машины массив последовательных ячеек для запоминания всех значений функции (результата); 2) вычислить функцию как переменную с индексами, так как в массивы записываются лишь переменные с индексами.

Задача 3.7. Вычислить и запомнить значения функции $y_i = e^{x_i}/\sqrt{1 + x_i}$, где x_i — переменная с индексом, заданная своими значениями x_1, x_2, ..., x_{100}.

Решение этой задачи аналогично решению задачи 3.5 и описывается циклическим вычислительным процессом, особенностью которого является то, что необходимо запомнить все значения вычисляемой функции (результата). Поэтому необходимо ввести дополнительный блок, который выделит для значений функции массив, состоящий из 100 последовательных ячеек, а саму функцию вычислить как переменную с индексом i (рис. 3.8). Индекс i изменяется при каждом новом повторении цикла на единицу с помощью блока *5*, благодаря чему каждое новое значение функции y_i будет записываться в отдельную ячейку массива y. Поскольку индекс i принимает значения 1, 2, 3, ..., 100, то результаты будут записываться соответственно в 1, 2, 3, ..., 100-ю ячейки этого массива. Здесь имеется два массива: массив исходных данных (переменной с индексом x_i), который вводится в машину с помощью блока *1*, массив результатов, выделяемый с помощью блока *2*, в который записываются соответствующие числовые значения по мере их вычисления.

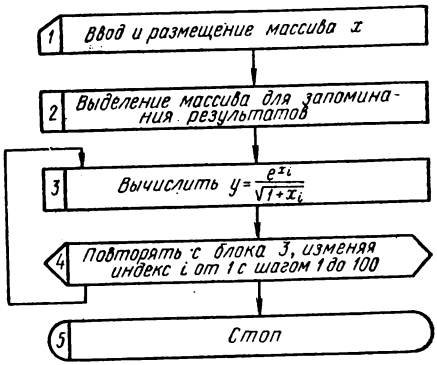


Рис. 3.8. Блок-схема алгоритма простого цикла с запоминанием результатов

В случае, если результаты нужно не только запомнить, но и вывести на печать, внутри цикла должен быть также блок печати.

Использование рекуррентных выражений. Предположим, что переменная x задана законом своего изменения, который в математике записывается выражением $x_{i+1} = x_i + h$, где h — шаг изменения переменной, отличный от нуля.

Согласно сказанному выше, при программировании такую переменную надо рассматривать как простую переменную x. Поэтому $x = x + h$. В математике такое выражение можно представить лишь как уравнение, не имеющее решения. При программировании это выражение имеет тот смысл, что к x прибавляется h и полученный результат присваивается x как новое его значение. Другими словами, знак «=» здесь имеет смысл присваивания. В некоторых языках программирования, например языке «АЛГОЛ», вместо этого знака ставится специальный знак «:=».

При программировании рекуррентные соотношения используют для вычисления текущих значений переменной, ее предыдущие значения (в левой части выражения ставится обозначение переменной, которое определяется, а в правой части—функция от старых значений этой переменной). Рекуррентные выражения используют также и для вычисления некоторых функций.

Вычисление суммы

Задача 3.8. Пусть необходимо вычислить $Z = \sum_{i=1}^{n} f(x_i)$, где x_i значения переменной x, заданной пределами изменения от a до b с шагом изменения h.

Решение задачи может быть следующим. Вычисляют значения функции $f(x_i)$ и прибавляют их к сумме предыдущих значений этой функции, т. е. последовательно получают все промежуточные суммы:

$$Z_0 = 0;$$
$$Z_1 = Z_0 + f(x_1) = f(x_1);$$
$$Z_2 = Z_1 + f(x_2) = f(x_1) + f(x_2);$$
$$Z_3 = Z_2 + f(x_3) = f(x_1) + f(x_2) + f(x_3);$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$Z_n = Z_{n-1} + f(x_n) = f(x_1) + f(x_2) + \ldots + f(x_n).$$

Поскольку запоминать в памяти машины значения всех слагаемых не требуется, то в целях ее экономии целесообразно хранить эти слагаемые последовательно в одной и той же ячейке, а сумму накапливать в одной ячейке, в которой последовательно будут храниться все ее промежуточные значения.

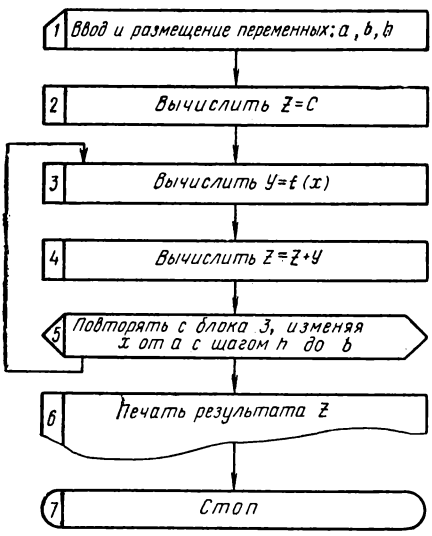


Рис. 3.9. Блок-схема алгоритма вычисления суммы

Поэтому слагаемое вычисляют, как простую переменную $y = f(x_i)$.

Следовательно, переменная Z также будет простой переменной, а формула, по которой накапливается сумма, будет $Z = Z + Y$. Эта формула имеет тот смысл, что к предыдущему значению суммы прибавляется новое слагаемое Y и полученный результат считается новым значением суммы.

Повторяя n раз процесс вычисления слагаемых Y и накопления суммы по формуле $Z = Z + Y$, получают искомое значение $Z = \sum_{i=1}^{n} f(x_i)$.

При первом выполнении такого цикла Z должно быть равно первому значению слагаемого Y. Следовательно, Z_0 должно быть равно нулю. Поэтому перед циклом надо «очищать» ту ячейку, в которой будет накапливаться сумма, т. е. вычислять $Z = 0$.

Блок-схема алгоритма такого вычислительного процесса представлена на рис. 3.9.

Вычисление произведения

Задача 3.9. Вычислить $Z = \prod_{i=1}^{n} f(x_i)$.

Считая $Y = f(x_i)$ и промежуточные произведения простыми переменными, получают выражение для накопления произведения $Z = ZY$, которое имеет следующий смысл. Промежуточное произведение умножается на новый сомножитель Y, и полученный результат считается равным следующему промежуточному произведению. Очевидно, что Z_0 должно быть равно единице, поэтому перед циклом необходимо вычислить $Z = 1$.

Считая, что x_i является переменной с индексами, получают блок-схему алгоритма вычислительного процесса, которая представлена на рис. 3.10.

Определение наибольшего (наименьшего) из чисел.

Задача 3.10. Задана функциональная зависимость $y_i = f(x_i)$ $(i = 1, 2, \ldots, n)$. Требуется найти наибольшее значение функции y_i.

Задачу можно решить следующим образом: после вычисления каждого нового значения функции y_i сравнить его с максимальными из всех вычисленных ранее значений этой функции и считать максимумом наибольшее из этих значений.

Поскольку первое значение сравнивать не с чем, то надо сначала сравнивать его с каким-либо очень маленьким числом, для того чтобы после этого сравнения получить в качестве максимума это первое значение функции.

Пусть перед решением задачи $MAX = -10^{10}$.

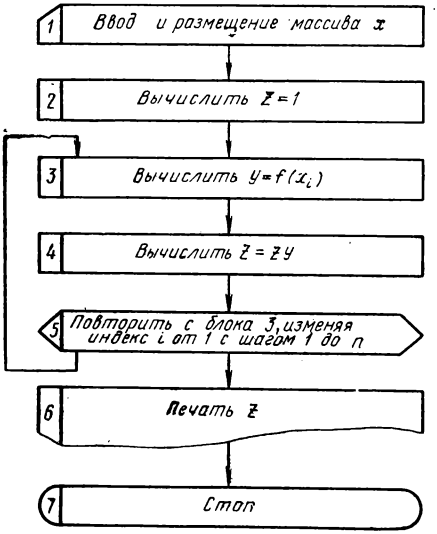


Рис. 3.10. Блок-схема алгоритма вычисления произведения

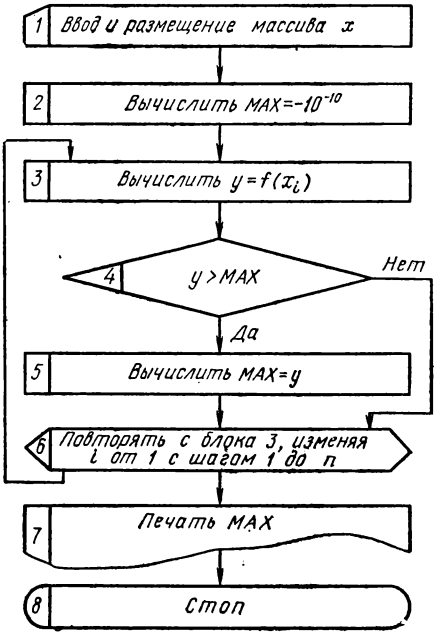


Рис. 3.11. Блок-схема алгоритма вычисления максимума

Тогда после вычисления y_1 $MAX = \max\{MAX, y_1\} = y_1$, поскольку в качестве максимума взято число, которое заведомо меньше любого значения функции y_i.

Далее получают:

$$MAX = \max\{MAX, y_2\};$$
$$MAX = \max\{MAX, y_3\};$$
$$\ldots\ldots\ldots\ldots$$
$$MAX = \max\{MAX, y_n\}.$$

Вычисление максимума может быть реализовано следующим образом:

$$MAX = \begin{cases} y_i, & \text{если } y_i > MAX; \\ MAX, & \text{если } y_i \leqslant MAX. \end{cases}$$

Второе выражение имеет тот смысл, что значение максимума остается прежним, если $MAX \leqslant y_i$.

Повторяя этот разветвленный процесс n раз, получают MAX, равный наибольшему числу.

Если же требуется знать не все значения функции y, а лишь y_{max}, то функцию y_i целесообразно считать простой переменной. Поэтому, считая x_i переменной с индексом, получают блок-схему алгоритма, представленную на рис. 3.11.

По аналогичной схеме осуществляется нахождение минимума. В этом случае в качестве первого значения MIN надо взять очень большое число, чтобы после сравнения его с первым значением функции последнее оказалось меньше его.

Например, $\mathrm{MIN} = 10^{10}$.

Тогда $\mathrm{MIN} = \min\{\mathrm{MIN}, y_1\} = y_1$.

Вычисление минимума реализуется следующим образом:

$$\mathrm{MIN} = \begin{cases} y_i, & \text{если } y_i < \mathrm{MIN}; \\ \mathrm{MIN}, & \text{если } y_i \geqslant \mathrm{MIN}. \end{cases}$$

Вычисление суммы бесконечного ряда

Задача 3.11. Необходимо вычислить значение e^x с помощью ряда:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} + \ldots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

с заданной точностью ε.

Алгоритм целесообразно сформулировать, исходя из следующих предпосылок.

1. Для вычисления членов ряда воспользоваться рекуррентной формулой:

$$H_{n+1} = H_n \frac{x}{n+1},$$

поскольку прямое вычисление $(n+1)$-го члена ряда по формуле $\frac{x^{n+1}}{(n+1)!}$ требует большего времени (возведение числа в n-ю степень реализуется в ЦВМ с помощью следующей цепи действий: $x^n = e^{n \ln x}$, причем логарифмирование и потенцирование требуют в 50—100 раз больших затрат машинного времени, чем однократное умножение при использовании вышеприведенной рекуррентной формулы). Поскольку $H_0 = 1$, то достаточно это значение взять в качестве исходного значения простой переменной H.

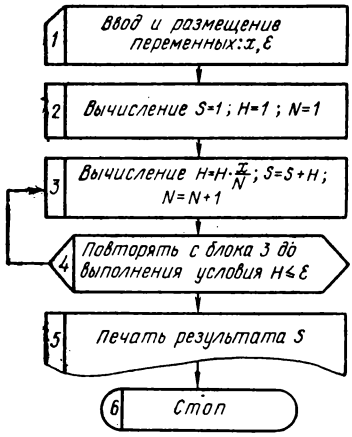


Рис. 3.12. Блок-схема алгоритма вычисления суммы бесконечного ряда с точностью до ε

2. Суммирование вычисляемых последовательно членов ряда производить, используя способ накопления суммы (см. задачу 3.8). Здесь целесообразно начальное значение суммы S считать равным не нулю, а первому слагаемому, так как вычислять его нет необходимости, т. е. $S = 1$.

На основании сказанного выше словесная формулировка вычислительных действий программного алгоритма выглядит так:

1) задать исходные значения $S = 1$, $H = 1$, $N = 1$;

2) вычислить $H = H \cdot x/N$, $S = S + H$, $N = N + 1$;

3) повторять вычисления по пункту 2 до тех пор, пока не будет выполнено условие $H \leqslant \varepsilon$; действительно, в этом случае остаточный член ряда не превысит ε, следовательно, заданная точность будет достигнута.

Блок-схема алгоритма представлена на рис. 3.12. Особенностью этого алгоритма является максимальное использование циклических свойств образующих его формул.

Вычисление значений полинома n-ой степени

Задача 3.12. Вычислить значение полинома n-й степени.

Использованию циклических свойств алгоритмов способствуют разработанные вычислительные приемы, дающие формулы, пригодные для их применения

в качестве основы алгоритмов. Для построения алгоритма вычисления значений полиномов n-й степени вида

$$y = a_1 x^n + a_2 x^{n-1} + a_3 x^{n-2} + \ldots + a_{n+1}$$

удобна формула Горнера

$$y = (\ldots(((a_1 x + a_2) x + a_3) x + a_4) x + \ldots) x + a_{n+1}.$$

Эта формула обеспечивает минимальное количество машинных операций, поскольку для возведения переменной x в n-ю степень используют рекуррентную формулу.

При вычислении m значений полинома y для различных значений переменной x_j формула Горнера принимает вид:

$$y_j = (\ldots((a_1 x_j + a_2) x_j + a_3) x_j + \ldots) x_j + a_{n+1}.$$

Для каждого x в соответствии с вычислительной схемой нужно выполнить следующую последовательность действий:

1) коэффициент a_1 умножить на x_j и к результату прибавить a_2;

2) полученный в пункте 1 результат умножить на x_j и прибавить a_3 и т. д.;

i) полученный в $(i-1)$-м пункте результат умножить на x_j и прибавить a_{i+1}.

Как видно, результаты в каждом из этих пунктов получаются один из другого по рекуррентной формуле:

$$y_{i+1} = y_i x_j + a_{i+1}. \qquad (3.3)$$

При этом y_1, y_2, ..., y_i, y_{i+1} представляют собой последовательные значения простой переменной y. Начальное значение переменной y равно a_1. Для вычисления значения полинома y (x_j) достаточно повторить вычисления по рекуррентной формуле n раз. Значения коэффициентов a_i и аргументов x_j размещают в массивах a и x. Для вычисления значений полинома при m различных значениях аргумента x_j достаточно вычисления для y (x_j) повторить m раз, меняя j.

Блок-схема алгоритма вычисления значения полинома n-й степени приведена на рис. 3.13.

Внутренний цикл этой сложной циклической программы обеспечивает вычисление одного значения полинома, а внешний цикл — вычисление m значений полинома.

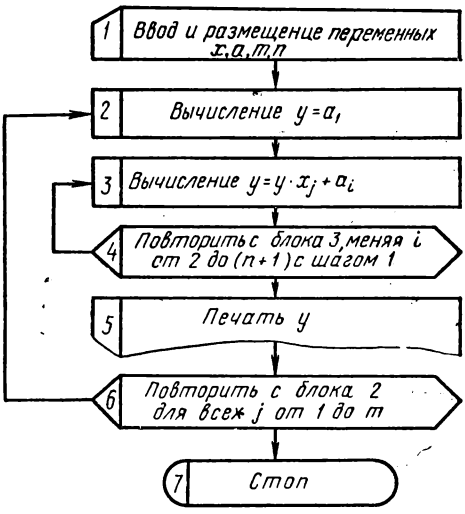


Рис. 3.13. Алгоритм Горнера

Для сложной циклической программы вычисления, предусмотренные во внутреннем цикле, повторяются $p = r_1 \cdot r_2 \cdot \ldots \cdot r_k$ раз, где r_1, r_2, ..., r_k — число повторений для всех k ступеней циклов программы. С увеличением k и r_k количество действий, предписываемых алгоритмом, и время решения задачи быстро растут. В случаях, когда удается в процессе составления сложного циклического алгоритма выделить независимые циклы, за счет усложнения алгоритма, сокращается время вычислений.

Составление программы на алгоритмическом языке. На этом этапе алгоритм решения задачи записывается на алгоритмическом языке в виде программы.

Алгоритмическим языком называют совокупность символов и правил, позволяющих описывать алгоритмы и однозначно истолковывать смысл описания.

Алгоритмический язык, непосредственно воспринимаемый вычислительной машиной, называют *языком команд вычислительной машины* или *языком вычислительной машины.*

Помимо языка вычислительной машины для записи алгоритмов разработаны алгоритмические языки, на которых программы записываются с помощью символики, близкой символике общепринятой в математике. Применение алгоритмических языков этого типа уменьшает трудоемкость программирования. Перевод программы на язык вычислительной машины возлагается на саму вычислительную машину, которая осуществляет такой перевод с помощью специальных программ-трансляторов. Благодаря этому существенно упрощается процесс подготовки задачи для решения на ЦВМ.

В настоящее время существует много алгоритмических языков, отличающихся друг от друга не только способом описания программ, но и назначением. Такие языки называют *проблемно-ориентированными языками.*

Алгоритмический язык ФОРТРАН предназначен для решения задач численного анализа. Он обладает простотой, близостью записи выражений к выражениям, принятым в математике, простыми операторами ввода—вывода информации, простотой транслятора, возможностью выявления синтаксических ошибок. К недостаткам этого языка можно отнести невозможность динамического распределения памяти и необходимость нумеровать элементы массивов с единицы.

Алгоритмический язык АКИ также предназначен для решения задач численного анализа. Его характеристики аналогичны характеристикам языка ФОРТРАН.

Алгоритмический язык АЛГОЛ-60, подобно языкам ФОРТРАН и АКИ, предназначен для решения задач численного анализа. Этот язык более сложный, но и более выразительный: в нем допускается динамическое распределение памяти и произвольная индексация элементов массива. Блочная структура программ, записанных на языке АЛГОЛ-60, обеспечивает возможность независимого написания отдельных частей программы и экономию памяти. Однако это приводит к усложнению транслятора и увеличению времени трансляции.

Для решения задач экономического и планово-производственного характера широкое распространение получил алгоритмический язык КОБОЛ. Достоинством этого языка является развитый аппарат описания структур исходных данных.

Стремление к расширению возможностей языка привело к созданию макроязыков и макропроцессоров, позволяющих включать в программы макрокоманды, которые во время трансляции заменяются текстом на основном языке. Примером такого макропроцессора является макропроцессор системы IBM-360.

ГЛАВА 4

ОСНОВЫ АВТОМАТИЗАЦИИ ПРОГРАММИРОВАНИЯ

§ 4.1. ПОНЯТИЕ О КОМАНДАХ И ПРОГРАММЕ

Машина осуществляет вычисления по программе, записанной определенным образом и называемой машинной программой. *Машинная программа* представляет собой последовательность команд, расположенных в порядке их выполнения.

Для управления работой машины необходимо знать: 1) какое действие должно выполняться в данный момент времени; 2) где хранится первое участвующее в операции число; 3) где хранится второе участвующее в операции число; 4) куда поместить результат выполнения данного действия; 5) к какой следующей команде необходимо перейти.

Информация о том, какое действие или операцию должна выполнить машина по данной команде, задается с помощью кода операции. Все элементарные операции, выполняемые машиной, кодируются в восьмеричной системе счисления. Каждая вычислительная машина имеет систему операций или команд, содержащую набор выполняемых ею элементарных операций и их кодов. Каждая команда должна содержать в себе информацию о коде операции.

Информация о том, где хранятся участвующие в операции числа и куда поместить результат, задается с помощью адресов или номеров ячеек оперативной памяти машины, в которых хранятся эти числа.

Информация о том, какая следующая команда должна выполняться после данной, задается с помощью адреса или номера ячейки, где хранится эта команда. В большинстве современных вычислительных машин принят естественный порядок выполнения команд; команды выполняются в той последовательности, в которой они записаны в памяти машины. В тех случаях, когда необходимо нарушить естественный порядок выполнения команд, используют специальные команды передачи управления или просто управления. Таким образом, информация о том, к какой команде переходить, вырабатывается в машине автоматически путем прибавления после выполнения каждой команды к ее адресу единицы.

Следовательно, команда должна содержать код операции и адресную часть. Например, трехадресная команда записывается так:

$$K)\ \theta a_1 a_2 a_3,$$

где K — адрес команды; θ — код операции; a_1 и a_2 — адреса участвующих в операции чисел; a_3 — адрес результата.

Поскольку команды так же, как и числа, хранятся в памяти машины, то перед каждой командой указывается ее адрес, т. е. номер ячейки, в которой она записана. Адрес команды от самой команды отделяется скобкой и, как и код операции, записывается в восьмеричной

системе счисления, а хранится в ячейках памяти в двоичной системе счисления.

Например, если код операции сложения 01, то команда, записанная в ячейке 520, выполняется следующим образом:

$$0520)\ 01\ 0772\ 1054\ 2000,$$

т. е. к содержимому ячейки 0772 прибавить содержимое ячейки 1054, результат поместить в ячейку 2000; следующей будет выполняться команда из ячейки 521.

Результаты, получаемые при выполнении одной трехадресной команды или трех одноадресных команд типа

$$K)\ \theta\ a_i,$$

будут одинаковы.

При использовании трех одноадресных команд по первой одноадресной команде в сумматор поступит содержимое ячейки a_1 (первое слагаемое), по второй команде—сложится содержимое сумматора с содержимым ячейки a_2 (вторым слагаемым), указанной в ее адресе, по третьей команде результат запишется в память машины в ячейку a_3, адрес которой указан в адресной части третьей команды. Но при использовании одноадресных команд программа получается более длинной (содержит больше команд). Однако для записи одноадресной команды требуется меньшее количество разрядов; поэтому длина разрядной сетки ячейки памяти машины, а следовательно, и количество аппаратуры для хранения одной команды резко уменьшаются. Если же учесть, что при выполнении многих действий в качестве одного из участвующих в операции чисел используется результат предыдущего действия, который находится в сумматоре, а результат нет необходимости записывать в память машины (одно действие может выполняться с помощью одной одноадресной команды), то преимущество машин с одноадресными командами очевидно.

Существуют также двухадресные команды типа

$$K)\ \theta\ a_1 a_2.$$

Здесь в ячейках с адресами a_1 и a_2 записываются либо адреса обоих участвующих в операции чисел, либо адрес одного из участвующих в операции чисел (в ячейку a_1), при этом результат либо записывается в ячейку с адресом a_2, либо запоминается в сумматоре.

Двухадресные машины требуют соответственно меньшего объема аппаратуры и количества команд, чем трехадресная и одноадресная машины.

В машинах, предназначенных для различных целей, решающее значение приобретает один из этих показателей, что и определяет выбор адресности используемых в них команд.

§ 4.2. АВТОМАТИЗАЦИЯ ИСПОЛЬЗОВАНИЯ СТАНДАРТНЫХ ПРОГРАММ

При решении задач на ЦВМ необходимы перевод исходных данных из десятичной системы счисления в двоичную, обратный перевод результатов решения, вычисления элементарных функций $\ln x$, $\sin x$, e^x ..., интегралов, решения систем алгебраических и дифференциальных уравнений и т. д. Составление программ, по которым осуществляются вышеперечисленные действия, очень трудоемкое дело, поэтому программы составляют заранее и используют при решении задач по мере надобности. Такие программы называют *стандартными программами* (СП). СП объединяют в библиотеку стандартных программ (БСП).

Для обращения к СП используется команда безусловной передачи управления с записью возврата. При этом необходима следующая информация; 1) куда заслать аргумент; 2) где будет помещен результат; 3) номер начальной ячейки СП; 4) номер ячейки возврата.

Использование СП позволяет сократить время, затрачиваемое на программирование задачи.

Поскольку в настоящее время БСП насчитывают большое количество различных СП, а объем памяти машины ограничен, то невозможно записывать СП последовательно в памяти машины в действительных адресах. Поэтому все СП составляют в условных адресах и хранят во внешних запоминающих устройствах. При этом используют различные способы автоматизации обращения к БСП, обеспечивающие выбор, ввод, перевод в действительные адреса и размещения в памяти машины необходимых для решения данной задачи СП. Для этих целей применяют компилирующую и интерпретирующую системы.

Компилирующая система. Эта система состоит из БСП, таблицы характеристик программ и компилирующей программы.

Таблица характеристик программ содержит информацию о размещении всех СП библиотеки во внешнем запоминающем устройстве и состоит из кода, указывающего вид накопителя (магнитная лента, магнитный барабан), номер накопителя, зоны или ячейки, с которой начинается данная СП, контрольной суммы кодов, используемой для контроля правильности ввода СП в память машины.

Компилирующая программа просматривает основную программу, определяет с помощью таблицы характеристик программ необходимые для решения задачи СП, вызывает и размещает их в выделенном для них массиве памяти машины. Затем компилирующая программа заменяет условные адреса СП на действительные и согласовывает их с основной программой. В результате выполнения всех этих действий в памяти машины будет готовая к использованию программа. В процессе решения компилирующая программа не используется.

Интерпретирующая система. Эта система состоит из БСП, таблицы характеристик и интерпретирующей программы. Таблица характеристик, как и в компилирующей системе, содержит информацию о размещении всех СП библиотеки во внешнем запоминающем устройстве.

Интерпретирующая программа выявляет команды обращения к СП, вызывает и размещает СП в памяти машины в специальном рабочем

массиве и переходит к ее выполнению. Затем по мере выполнения основной программы выявляет следующую команду обращения к СП и повторяет описанный выше процесс и т. д. Таким образом, интерпретирующая программа все время находится в памяти машины и используется в процессе выполнения основной программы.

Объем рабочего массива, отведенного для хранения вызываемых из внешнего запоминающего устройства СП, ограничен, поэтому СП в массиве хранятся по очереди по мере надобности. Поскольку во время выполнения основной программы приходится обращаться к внешним запоминающим устройствам для вызова очередной СП, то время выполнения основной программы увеличивается.

Каждая из этих систем имеет свои преимущества и недостатки, поэтому целесообразность использования той или иной системы зависит от конкретных условий (типа решаемой задачи, ее сложности, количества используемых СП и т. д.). Но надо помнить, что компилирующая система по сравнению с интерпретирующей системой дает выигрыш во времени решения задачи, так как вызывает СП в память машины лишь один раз, но требует большего объема памяти, поскольку необходимо хранить все используемые СП.

Рассмотренные системы облегчают процесс программирования за счет того, что вместо отдельных участков программы используются уже имеющиеся СП, размещение которых и согласование с основной программой производится автоматически. Однако остальные участки программы все же записываются в коде машины, т. е. сохраняется трудность и неудобство программирования в этом коде. Одним из способов устранения указанных трудностей является использование при программировании псевдокоманд или символических команд.

§ 4.3. СИСТЕМА СИМВОЛИЧЕСКОГО КОДИРОВАНИЯ

Сущность символического кодирования заключается в том, что в командах вместо конкретных числовых адресов и кодов операций записываются буквенно-цифровые символы, которые представляют собой наименования переменных или констант, сокращенное название кодов операций или какие-либо другие обозначения.

Программу, составленную с помощью символических команд, называют *символической программой*. Символическую программу ЦВМ выполнять не может, поэтому перед решением задачи ее необходимо перевести в программу, записанную в коде машины, т. е. заменить символические коды операций на соответствующие коды машины, а символические адреса на соответствующие числовые адреса.

Для перевода символических кодов операций в коды машины в программе-трансляторе есть таблица их соответствия. Чтобы заменить символические адреса на числовые, программа-транслятор производит распределение памяти и на основании этого каждой символической переменной ставит в соответствие действительный числовой адрес (при этом машина должна уметь отличать команды от чисел и констант).

Такая методика составления программ очень полезна в тех случаях, когда нужно записывать команды программы, использующие адреса чисел или команд, местоположение которых еще не известно. Поскольку символическое наименование адресов обычно соответствует их содержимому, а символические коды операций являются сокращенными наименованиями кодов, то символическая программа легко записывается и проверяется. Благодаря методу символического кодирования появляется также возможность программирования задачи по частям (блочное программирование), что особенно важно при решении сложных и громоздких задач, так как для программирования отдельных участков таких программ может привлекаться несколько программистов.

Метод символического кодирования позволяет легко вносить исправления и изменения в программу, а также уменьшает количество ошибок при программировании и дает возможность их автоматически обнаружить и исправить.

Принцип символического кодирования рассмотрим на примере системы символического кодирования (ССК) машин серии «Минск».

Из сказанного выше следует.

1. Система символического кодирования позволяет записать программу решения задачи на некотором промежуточном языке, отличном от языка (кода) машины. Такой язык называют языком символического кодирования (ЯСК).

2. Программа, записанная на ЯСК автоматически переводится самой машиной в программу, записанную в коде машины, т. е. язык СК обеспечивает автоматизацию программирования.

3. Поскольку каждому коду операции машины соответствует символический код операции, представляющий собой его сокращенное наименование, то язык СК является машинно-ориентированным языком, т. е. он учитывает особенности конкретной машины и ее систему команд. По этой же причине каждой символической команде соответствует команда в коде машины, т. е. ЯСК относится к языкам уровня «один к одному».

В качестве алфавита ЯСК используют алфавит второго международного телеграфного кода МТК-2, содержащий 78 различных символов: 31 русскую заглавную букву; 26 латинских заглавных букв; 10 цифр; 11 специальных знаков.

Символическая программа записывается с помощью следующих символических операторов:

1) символических команд, преобразуемых в соответствующие им команды машинной программы (набор символических команд определяется системой команд машины);

2) символических опередений, используемых для создания констант, резервирования зон для рабочих ячеек, исходных данных и результатов;

3) символических директив, применяемых для управления ходом трансляции, указания начала программы, комментариев и др.

Символические операторы. Символические операторы состоят из кода оператора и адресов. Код оператора записывается в виде сокра-

щенного названия этого оператора или непосредственно в машинном коде. Для записи адресов используются либо символические названия, либо действительные адреса в десятичной или восьмеричной системе (в последнем случае после адреса пишется буква В, например 1050В).

Для ссылки символические операторы снабжаются этикеткой, помещаемой перед оператором. Каждый оператор может сопровождаться замечанием, поясняющим этот оператор или группу операторов.

Символическая программа записывается на следующем бланке:

Строка	Этикетка	КОп	Адреса и замечания
010			
020			
030			
040			
050			
060			
070			
080			
. . .			
290			
300			

Рассмотрим назначение отдельных граф бланка.

В графу «*Строка*» записывают номера строк от 010 до 300 с шагом 10 в десятичной системе счисления. Это дает возможность вставлять между ними новые строки, номера которых и их содержимое записывается в пяти последних непронумерованных строках. Например, если нужно после строки 160 вставить еще две символические команды, то они записываются в непронумерованные строки и им присваиваются адреса 161 и 162.

В графу «*Этикетка*» записывают символические наименования операторов, которые служат для ссылок к этим операторам, например, для перехода к операторам с помощью команд управления. Этикетка может содержать от одного до пяти символов (русские буквы и цифры), но начинаться она всегда должна с буквы.

В графу «*КОп*» записывается код операции символических операторов в символическом виде или в виде машинных кодов.

В последнюю графу заносят без пропусков через запятую адреса операторов. Остальные позиции в этой графе могут занимать замечания, отделяемые от адресов одним пробелом. Адреса записываются либо в виде действительных адресов в десятичной или восьмеричной системе, либо в виде символических адресов, состоящих из буквенно-цифровых символов (от одного до пяти). Все символические адреса обязательно должны присутствовать в графе «*Этикетка*». Для обозначения индекс-адреса используют знак «:», который помещают перед соответствующим индексом.

Коды операций символических команд символически обозначают с помощью первых букв их наименований.

В табл. 4.1 представлены некоторые наиболее употребимые символические операции и соответствующие им машинные коды операций для чисел с плавающей запятой .В случае использования чисел с фиксированной запятой в наименовании кода операции достаточно заменить букву П на букву Ф. Например, УП для чисел с плавающей запятой означает, вторую модификацию команды умножения, а для чисел с фиксированной запятой этот же код операции записывают в виде УФ.

Таблица 4.1

Символический КОп	Машинный КОп	Пояснение
СПЗ	+14	Сложение с плавающей запятой
СП	+15	
СПВ	+16	
СПР	+17	
ВПЗ	+24	Вычитание с плавающей запятой
ВП	+25	
ВПВ	+26	
ВПР	+27	
УПЗ	+34	Умножение с плавающей запятой
УП	+35	
УПВ	+36	
УПР	+37	
ДПЗ	+44	Деление с плавающей запятой
ДП	+45	
ДПВ	+46	
ДПР	+47	
П	—10	Пересылка
ПОЗН	—11	Пересылка с обратным знаком
И	—30	Идти (перейти)
ИЗАП	—31	Идти и запомнить (переход с возвратом)
ИЗНА	—32	Идти по знаку (переход по знаку)
ИНУЛ	—34	Идти по нулю (переход по нулю)
ЦИКЛ	—20	Конец цикла
ОСТ	—00	Останов
ВЛЦ	—50	Ввод с перфоленты цифровой
ЫУДП	—60	Вывод на узкую печать десятичных чисел с плавающей запятой
ЫУДФ	—60	Вывод на узкую печать десятичных чисел с фиксированной запятой

Для обращения к библиотечным стандартным программам используют код операции БСП.

В адресной части оператора БСП записывается восьмеричный номер стандартной программы, который присвоен этой программе в библиотеке стандартных программ. Если на этот оператор есть ссылка в символической программе, то ему присваивается этикетка. После номера стандартной программы могут записываться текстовые пояснения, например, название подпрограммы (табл. 4.2).

При задании входной информации для СП требуются специальные строки, которые описываются операторами ПСК или КВ.

По оператору БСП строится машинная команда — 31 00 НСП0017, где НСП — начальный адрес СП в памяти машины. Оператор БСП

Таблица 4.2

Этикетка	КОп	Адреса и замечания
СИНУС	БСП	31В
	БСП	40В Умножение матриц
	ПСК	Н, А1, В1
	ПСК	К, М, С1

дает один из способов включения СП в машинную программу. Другими способами включения СП в программу могут быть способы, использующие компилирующие и интерпретирующие программы.

Символические определения. Символические определения преобразуют в двоичный код и записывают в памяти машины числа и константы, а также резервируют в памяти место для рабочих ячеек. Наименования символических определений заносят в графу «*КОп*», а в графу «*Адреса и замечания*» — сами числа или константы. В конце оператора обязательно ставится знак «␣» (пробел). Операторам, на которые есть ссылка в символической программе, присваивают этикетки.

Операторы определения чисел применяют для записи десятичных чисел, которые перед размещением в памяти машины требуют перевода в двоичную систему счисления.

В зависимости от формы представления чисел используют следующие определения: ЧЦ — числа целые; ЧФ — числа с фиксированной запятой; ЧП — числа с плавающей запятой.

В графе «*Адреса и замечания*» числа записывают, начиная с крайней левой позиции, при этом целое число и число с фиксированной запятой должно иметь не более 12 цифр, а мантисса числа с плавающей запятой не более девяти цифр. Вместо запятой ставится точка, а вместо основания 10 записывается русская буква Ю (табл. 4.3).

Транслятор, в зависимости от формы представления числа, переведет его соответствующим образом в двоичный код и присвоит каждой этикетке адрес ячейки, в которой это число будет находиться.

Операторы определения констант служат для нахождения различных восьмеричных и двоично-десятичных констант, а также для кодирования текстовых слов. Рассмотрим псевдоконстанту (ПСК) и константу восьмеричную (КВ).

Таблица 4.3

Этикетка	КОп	Адреса и замечания	Обычная запись
К1	ЧЦ	+5	+5
К2	ЧЦ	—191	—191
К3	ЧФ	0.0125	+0,0125
К4	ЧФ	—1.125Ю—2	—0,0125
К5	ЧФ	—.4	—0,4
К6	ЧП	28.375	+28,375
К7	ЧП	—.8Ю5	$-0{,}8 \cdot 10^5$
К8	ЧП	Ю4	$+10^4$

ПСК состоит из трех составных частей, которые записываются соответственно в разряды с 0 до 10, с 13 по 24, с 25 по 36.

Последние нулевые части констант, а также незначащие нули в их числовой части можно не писать. Части констант могут записываться с помощью букв, чисел или их комбинации, а также представлять собой арифметические выражения типа сложения и вычитания. Отдельные части констант отделяются друг от друга запятыми.

КВ описывает восьмеричную константу, состоящую не более чем из 12 цифр, не считая знака. Для удобства записи и чтения восьмеричные константы можно разбивать на три группы по четыре восьмеричные цифры в каждой. Примеры записи ПСК и КВ приведены в табл. 4.4.

Таблица 4.4

Этикетка	КОп	Адреса и замечания	Результаты трансляции
К1	ПСК	О, А1, А2	00 00 0100 0101
К2	ПСК	12, 171, 1	00 14 0253 0001
К3	ПСК	Колич., А1+3	02 00 0103 0000
К4	КВ	—17	—00 00 0000 0017
К5	КВ	—17, 0, 0	—00 17 0000 0000
К6	КВ	+0, 0, 17	+00 00 0000 0017

Для ПСК принято следующее распределение памяти: А1 — 0100; А2 — 0101; Колич. — 0200.

Определение РАБ (рабочие ячейки) служит для резервирования нескольких ячеек памяти для промежуточных и окончательных результатов, количество которых указывается в графе «*Адреса и замечания*». Этикетке присваивается адрес первой ячейки массива рабочих ячеек, под все остальные результаты отводятся следующие по порядку ячейки.

Табл 4.5 показывает, что если адрес первой рабочей ячейки Р1 принять 0100, то под рабочие ячейки Р1 будут выделены ячейки с 0100 по 0104, а под рабочую ячейку Р2 — ячейка 0105 и т. д.

После трансляции в рабочие ячейки ничего не заносится.

Символические директивы. *Символические директивы* управляют работой транслятора и не преобразуются в команды машинной программы. Рассмотрим лишь две из них.

Начало (НАЧ). Этот оператор указывает, с какого адреса должна располагаться в памяти машинная программа или ее часть. Значение адреса записывается в графе «*Адреса и замечания*».

Таблица 4.5

Этикетка	КОп	Адреса и замечания	Номера ячеек
Р1	РАБ	5	0100
Р2	РАБ	1	0105
Р3	РАБ	2	0106

Если в программе отсутствует оператор НАЧ, то память будет распределяться начиная с адреса 0112.

Оператор НАЧ является одним из первых операторов программы.

К о н е ц (КОН). Этот оператор замыкает символическую программу. В графе «*Адреса и замечания*» записывается пусковой адрес программы, с которого начнется счет после трансляции программы. Пусковой адрес может указываться в виде действительного адреса или этикетки, например:

КОН 0400 или КОН ПУСК

В первом случае программа будет выполняться с адреса 0400, во втором случае — с адреса, который будет присвоен этикетке ПУСК. При этом этикетку ПУСК должна иметь первая символическая команда программы.

Пример составления программы на языке символического кодирования. Необходимо вычислить $Z = x + x^2/2 + \ldots x^i/i$, где $i = 1, 2, 3, \ldots, x = 0{,}5$.

Сумму вычислять до тех пор, пока $x^i/i \geqslant 10^{-5}$.

Приемы программирования в языке символического кодирования те же, что и при машинном кодировании; разница заключается лишь в том, что вместо действительных кодов операций и адресов записываются символические.

Программу расположим начиная с ячейки 0200.

Для хранения исходных чисел с плавающей запятой $\delta = 1 \cdot 10^{-5}$; $x = 0{,}5$; 1 выберем соответственно ячейки ДЕЛТА, ИКС, ЕДИН.

Для хранения искомой суммы выберем ячейку СУМ, а для хранения промежуточных результатов значения числителя a, номера слагаемого i, слагаемого y соответственно рабочие ячейки ЧИСЛ, НОМ, Р1.

В символической программе, приведенной в табл. 4.6, в отличие от машинной программы появились лишь две символические директивы НАЧ и КОН, задающие информацию, необходимую для трансляции и решения задачи.

Таблица 4.6

Строка	Этикетка	КОп	Адреса и замечания
010		НАЧ	128
020	ПУСК	П	О, СУМ
030		П	ЕДИН, ЧИСЛ
040		П	О, НОМ
050	ЦИКЛ	СПЗ	ЕДИН, НОМ
060		УПЗ	ИКС, ЧИСЛ
070		ДПВ	НОМ, Р1
100		ВП	ДЕЛТА
110		ИЗНА	К1, К2
120	К1	СПЗ	Р1, СУМ
130		И	ЦИКЛ
140	К2	ОСТ	
150	ДЕЛТА	ЧП	Ю—5
160	ЕДИН	ЧП	1
170	ИКС	ЧП	0.5
200	ЧИСЛ	РАБ	1
210	НОМ	РАБ	1
220	СУМ	РАБ	1
230	Р1	РАБ	1
		КОН	ПУСК

Таким образом, применение ЯСК является первым шагом автоматизации программирования, облегчающим как процесс самого программирования, так и процесс исправления и отладки программ. Однако ЯСК требует представления алгоритма решения задачи в виде последовательности элементарных действий, что очень громоздко.

Поскольку все вычислительные процессы описываются языком математики, то соответственно возникает следующий вопрос: нельзя ли в качестве промежуточного языка использовать язык математики или близкий к нему язык?

Ответом на этот вопрос послужило создание ряда алгоритмических языков программирования высокого уровня и трансляторов для перевода программ, записанных на этом языке, в код машины.

§ 4.4. ПРИНЦИПЫ ПОСТРОЕНИЯ АЛГОРИТМИЧЕСКИХ ЯЗЫКОВ

Ранее было показано, что использование библиотеки стандартных программ (БСП) позволяет существенно облегчить процесс программирования.

Пополняя БСП, можно свести ручное программирование к какому-то разумному минимуму, за счет использования все большего количества СП, вычисляющих различные функции. Однако даже для простых арифметических выражений количество вариантов, описываемых различными программами, столь велико, что полная автоматизация программирования на этом пути невозможна.

В предыдущем параграфе был рассмотрен один из вариантов автоматизации программирования, использующий ССК, а также была записана программа уже не на языке машины, а на языке символического кодирования. Программа, записанная на ЯСК самой машиной, переводится на язык машины с помощью транслятора (программы-переводчика). Правила перевода с ЯСК на язык машины сравнительно просты. Но ССК имеет существенный недостаток: программа получается громоздкой, так как каждому символическому оператору соответствует одна машинная команда.

Наиболее компактная запись алгоритма вычислительного процесса будет в случае использования общепринятых математических символов или запись на языке математики. Однако сложность и многообразие правил записи и толкования выражений на языке математики приведет к резкому усложнению программы транслятора и увеличению времени, затрачиваемого на перевод алгоритма на язык машины, что делает практически невозможным его применение.

Попытка найти компромисс между сложностью программирования и сложностью транслятора привела к появлению целого ряда промежуточных языков для записи алгоритмов вычислительных процессов, перевод с которых на язык машины производится автоматически. Такие языки получили название *алгоритмических языков.*

Рассмотрим некоторые принципы, положенные в основу автоматического перевода программы с алгоритмических языков на язык машины, на примере перевода простого алгебраического выражения.

Пусть необходимо перевести в машинную программу вычисление алгебраического выражения

$$Z = ax + (b - c) : x.$$

Для осуществления этого перевода программа-транслятор должна выполнить следующие основные функции.

1. Дешифрировать алгебраические символы, т. е. определить, какие из них являются переменными, а какие — знаками действий и в каком порядке и над какими переменными должны выполняться эти действия. Результатом выполнения этой функции будет некоторая символическая программа.

2. Присвоить переменным и командам действительные адреса, т. е. осуществить переход к машинной программе.

В общем случае эти две функции могут выполняться одновременно. Но для простоты понимания используемых при этом принципов будем рассматривать их раздельно.

Сначала преобразуем исходную формулу. Для этого примем следующие допущения:

1) проставим все знаки действия, чтобы избежать неоднозначности (например, знак умножения запишем в виде точки, т. е. $a \cdot x$, а не ax, поскольку в противном случае нельзя будет отличить произведения a, умноженного на x, от переменной ax) и чтобы в машине все символы выражения, в том числе и знаки действия, можно было представить в виде соответствующих двоичных кодов;

2) проставим скобки таким образом, чтобы все действия над парами чисел были обязательно заключены в скобки.

С учетом допущений исходное выражение примет вид:

$$Z = ((a \cdot x) + ((b - c) : x)).$$

Присвоим результату выполнения действия в первых скобках $(a \cdot x)$ наименование S, т. е. $S = a \cdot x$. Аналогично результату выполнения действия во вторых скобках $(b - c)$ — наименование Q, т. е. $Q = b - c$; далее $R = (Q : x)$ в третьих скобках и, наконец, $Z = (S + R)$.

Пронумеруем скобки в выражении таким образом, чтобы каждая новая открывающая скобка увеличивала свой номер по сравнению с предыдущей открывающей скобкой на единицу, а каждая закрывающая скобка уменьшала его на единицу:

$$Z\,\underset{1}{(}\,\underset{2}{(}a \cdot x\,\underset{1}{)} + \underset{2}{(}\,\underset{3}{(}b - c\,\underset{2}{)} : x\,\underset{1}{)}\,\underset{0}{)}.$$

Поскольку арифметические действия выполняются над числами, заключенными в пару скобок, то может быть предложен следующий алгоритм, по которому транслятор автоматически будет программировать вычисление алгебраических выражений.

Будем отсчитывать скобки слева направо. Уменьшение номера скобки на единицу указывает на то, что надо выполнить какое-то действие. Первой такой парой скобок будет $\underset{2}{(}a \cdot x\,\underset{1}{)}$, где код точки указывает

на то, что необходимо произвести действие умножения, а совокупность символов этой пары скобок — на последующую замену ее результатом S:

$$Z = \underset{1}{(}\ S + \underset{2\,3}{(}\ \underset{2}{(b-c)} : x\ \underset{1\,0}{)}\ \underset{1}{)}.$$

Далее будет производиться операция вычитания в скобках.

Полагая $b - c = Q$, получим

$$Z = \underset{1}{(}S + \underset{2}{(}Q : x\ \underset{1\,0}{)}\ \underset{1}{)}.$$

Действие деления осуществим в скобках $\underset{2}{(}Q : x\ \underset{1}{)}$. Полагая $Q : x = R$, получим

$$Z = \underset{1}{(}S + R\ \underset{0}{)}.$$

Последнее действие — сложение дает искомую функцию Z.

Таким образом определяется порядок выполнения действий и последовательность выполнения соответствующих символических команд.

Если транслятор для расшифровки таких выражений умеет отличать друг от друга буквы, используемые для обозначения переменных, скобки, задающие порядок действия, и сами знаки действий, которые определяют код соответствующих операций, то буквы будут задавать символические адреса переменных, скобки — порядок расположения команд, а знаки действий — коды их операций.

Примем для обозначения кодов операций следующие символические обозначения: С — сложение (+); В — вычитание (—); У — умножение (.); Д — деление (:). Систему команд для простоты возьмем трехадресную.

Символическая программа для рассматриваемого примера, построенная по такому принципу, будет иметь вид:

У a x S
В b c Q
Д Q x R
С S R Z

В первой колонке записаны символические коды операций, а в последующих — адреса. При этом сначала записывают адреса участвующих в операции чисел, а затем — адрес результата.

Аналогичным образом можно было бы закодировать и другие действия, например возведение в степень и извлечение корня, присвоив им соответствующие знаки действий и двоичные коды. Но поскольку этим действиям нет эквивалентных операций в коде машины, то для их выполнения надо иметь специальные стандартные программы (СП). В алгоритмических языках наличие символов, указывающих на необходимость подобных действий, служит причиной для обращения к

соответствующей СП. Так же обращаются к некоторым другим СП, предназначенным для вычисления основных функций. Например, записи sin (x) в алгоритмическом языке достаточно, чтобы произошло обращение к СП синуса. При этом в круглых скобках обязательно должен быть задан аргумент. Набор таких СП для каждого языка строго определен.

Для обращения к СП, предназначенных для вычисления неосновных функций, используют специальные операторы.

В рассмотренном в данном параграфе примере проставлялись скобки для всех действий. Но этого можно и не делать, если, как в математике, определить порядок выполнения действий, т. е. сначала выполнить действия типа возведения в степень и вычисления элементарных функций, затем действие умножения и деления и, наконец, действия сложения и вычитания. Например, при вычислении $Z = (a + x^3) : c$ в скобках необходимо выполнить действия сложения и возведения в степень. Первым должно быть выполнено по старшинству действие возведения в степень, а затем действие сложения.

Действие сложения будет выполнено раньше действия деления, хотя и имеет более низкий порядок старшинства, потому что оно заключено в скобки.

После составления символической программы машина должна, пользуясь таблицей соответствия символических и машинных кодов операций, присвоить каждой команде действительный код операции, а вместо символических адресов записать действительные адреса.

Задача замены символических кодов операций на машинные не представляет трудности.

Для замены символических адресов на действительные машина сначала составляет список команд, затем список исходных данных, констант и рабочих ячеек. После этого каждой команде или числу присваиваются последовательные действительные адреса.

Для рассмотренной символической программы такой список будет иметь вид:

0100	У a x S	Команды
0101	В b c Q	
0102	Д Q x R	
0103	С S R Z	
0104	a	Исходные числа
0105	x	
0106	b	
0107	c	
0110	S	Рабочие ячейки
0111	Q	
0112	R	
0113	Z	Ячейка ответа

Если каждой ячейке будут присвоены действительные адреса, начиная с 0100, как показано в левой графе, то, пользуясь данной таблицей, машина вместо символов в адресах этих команд запишет соответствующие им номера ячеек, т. е. вместо a номер 0104, вместо x номер 0105 и т. д. Но машина не может отличить команду от числа или константы, поэтому ей необходимо указывать последнюю команду программы, например, с помощью команды останова, так как изменять адресную часть ячеек, где хранятся исходные числа и константы, нет необходимости.

Выше был рассмотрен упрощенный вариант задачи перевода. В действительности же задача перевода усложняется тем, что иногда в командах, например командах сдвига, не надо заменять содержимое того адреса, где указана константа сдвига, а это приводит к тому, что машина по коду операции должна узнавать такие команды.

Часто в формулах, кроме переменных, могут присутствовать реальные числа, например $Z = (5{,}34 - x)\ b$. В этом случае этим числам, как и переменным, предварительно должны отводиться ячейки памяти.

Итак, при использовании программы транслятора, которая строится, как правило, по принципу компилирующих программ, необходимо хранить в памяти машины пять типов информации: 1) программу транслятора; 2) программу в символах входного языка; 3) таблицы соответствия символов, используемых в исходной программе, машинным кодам операций и адресам, которые будут содержать значения исходных переменных; 4) значения исходных переменных; 5) составленную машинную программу.

Вся эта информация хранится в памяти машины лишь в процессе трансляции программы. При ее решении необходима лишь информация четвертого и пятого типов. Это показывает, что для трансляции программы с алгоритмического языка требуется значительно бо́льшая память, чем для ее решения.

ГЛАВА 5

АЛГОРИТМИЧЕСКИЙ ЯЗЫК АЛГОЛ-60

§ 5.1. ОСНОВНЫЕ СИМВОЛЫ И ОБЪЕКТЫ ДЕЙСТВИЙ ЯЗЫКА АЛГОЛ-60

В настоящее время общее признание получил алгоритмический язык АЛГОЛ-60, предназначенный для описания вычислительных процессов. Этот язык явился в некотором смысле обобщением и усовершенствованием способов представления алгоритмов, применявшихся на более ранних стадиях развития программирования. В то же время АЛГОЛ включает в себя почти все основные понятия других алгоритмических языков.

В основе языка АЛГОЛ-60 лежит формульно-словесный способ записи алгоритмов, когда часть указаний дается в виде формул, а часть указаний описывается словами в виде пояснений. Поэтому этот язык близок не только к обычной математической символике, но и к разговорному языку.

Как и другие алгоритмические языки, АЛГОЛ-60 строится на основе трех компонентов: алфавита, синтаксиса и семантики.

Алфавит языка содержит набор символов, которыми разрешается пользоваться для изображения записей алгоритмов.

Синтаксис представляется совокупностью правил, определяющих построение допустимых конструкций языка на основе символов алфавита или других компонент.

Семантические правила определяют свойства, приписываемые различным конструкциям для однозначного их толкования с целью определения соответствующей последовательности этапов переработки информации.

Любая запись алгоритма на алгоритмическом языке образует строгую линейную последовательность символов, представленных в алфавите. Алфавит АЛГОЛа содержит достаточно богатый набор символов, что обеспечивает хорошую наглядность и выразительность записи алгоритма на этом языке.

Основными символами в алгоритмическом языке АЛГОЛ-60 являются цифры, буквы, логические значения и ограничители.

В АЛГОЛе используются арабские цифры 0—9 (другие цифры употреблять запрещается).

Буквы представлены строчными и прописными буквами латинского алфавита, которые можно дополнять буквами русского алфавита.

Логические значения имеют два символа: **true** (**истина**) и **false** (**ложь**).

Ограничители представлены знаками операций, скобками, разделителями, описателями и спецификаторами.

Знаки арифметических операций:

+ прибавить;
— отнять;
× умножить;
/ разделить;

÷ разделить нацело;
↑ возвести в степень.
Знаки логических операций:
∨ логическое сложение (ИЛИ);
∧ логическое умножение (И);
¬ отрицание (НЕ);
⊃ импликация (ВЛЕЧЕТ);
≡ эквивалентность.
Знаки операций отношения:
< меньше;
⩽ не больше;
> больше;
⩾ не меньше;
= равно;
≠ не равно.
Символы операций следования:
: = **присвоить значение;**
go to — перейти к;
if — если;
then — то;
else — иначе;
for — для;
do — выполнить.

Скобки служат для выделения частей записи программы, играющих роль самостоятельных единиц:
( ) — арифметические скобки;
[ ] — индексные скобки;
Операторные скобки
begin — начало end — конец

Разделителями, используемыми для разделения элементов конструкций языка, являются:
, запятая;
; точка с запятой;
. десятичная точка, которая отделяет целую часть числа от дробной части;
$_{10}$ основание степени — десять;
: двоеточие;
comment — примечание;
step — шаг;
until — до;
while — пока.

Описатели указывают на роль и свойства той информации, которой они предшествуют в программе:
real — действительный;
integer — целый;
Boolean — логический;
array — массив;
procedure — процедура;
switch — переключатель;
own — собственный.

Спецификаторы:
value — значение;
label — метка;
string — строка.

АЛГОЛ в основном ориентирован на решение задач вычислительного характера, поэтому к основным объектам действия языка следует отнести числа, переменные, указатели функций, которые с помощью знаков операций объединяются в арифметические, логические и именующие выражения.

Числа. Числа являются основными объектами, над которыми производятся действия. Иррациональные, трансцендентные и т. п. выражения заменяются их приближенными значениями с какой угодно степенью точности. Мнимые и комплексные числа в АЛГОЛ-60 не определены.

Запись чисел на языке АЛГОЛ-60 мало отличается от общепринятой. Для записи чисел используются десять арабских цифр, десятичная точка «.», символ «10» и знаки «+» или «—».

Все числа делятся на целые и вещественные (действительные). Основная разница между ними состоит в следующем.

Целые числа считаются в АЛГОЛе точными и все арифметические операции, которые над ними определены, тоже должны выполняться точно.

Вещественные числа считаются в АЛГОЛе приближенными и все арифметические операции выполняются над ними по общим правилам над приближенными числами.

Числа, состоящие только из комбинации символов цифр, со знаком или без него, считаются принадлежащими к типу **integer (целый)**, например, 42, — 60, 1935, +14. Вещественные числа могут быть целыми и дробными. Их можно записывать в любой привычной форме.

Десятичные дроби записываются как обычно, только вместо запятой используется десятичная точка. Если целая часть дроби равна нулю, то нуль можно не писать, однако заканчивать число символом «.» нельзя, например, 34.012, .471, . 00069 и т. д.

Для записи чисел, являющихся целыми степенями десяти, применяется особая конструкция — десятичный порядок. Для этой цели используется основной символ «10» — единственный подстрочный символ языка. Показатель степени записывается в одну строчку с самим числом после основания десять.

Например, числа $4 \cdot 10^7$ и $-35 \cdot 10^{-5}$ на языке АЛГОЛ-60 запишутся так: $4_{10}7$ и $-35_{10} - 5$.

Идентификаторы. Одним из основных понятий языка АЛГОЛ является понятие идентификатора.

При описании вычислительных процессов приходится оперировать с различными объектами: переменными, массивами, функциями и т. д. При записи алгоритма, естественно, приходится ссылаться на эти объекты и тем самым вводить для них соответствующие обозначения (идентификаторы).

Идентификаторы в записи алгоритма являются представителями соответствующих объектов. С точки зрения синтаксиса идентификатор — любая последовательность букв и цифр, начинающаяся с буквы.

Например,

x, *y*, *z*, *p*16, *q*828, *r*12;

или

time, *temperature*, *Simpson*.

Записи типа *a*. 5 или 6 *alpha* идентификаторами не являются, поскольку в первом случае, кроме букв и цифр, используется другой основной символ «.», а по определению такая запись не может быть иден-

тификатором. Во втором случае, хотя используются только буквы и цифры, но первый символ не буква, а цифра, что противоречит определению.

Идентификаторы в АЛГОЛе не имеют постоянного присущего им смысла. Они используются только для обозначения тех или иных объектов. В математике для этих целей применяют в основном отдельные буквы, например *z*, *p*, *t*, за исключением некоторых часто используемых обозначений, таких, как ln, tg и др. В АЛГОЛе в этом смысле накладывается гораздо меньше ограничений на выбор обозначений и тем самым предоставляется широкая свобода с тем, чтобы выбор обозначений позволял сделать запись алгоритма наглядной, понятной и выразительной. Так, какую-либо физическую величину, например время, которую обычно обозначают через букву *t*, в АЛГОЛе можно обозначить идентификатором *time*, который соответствует названию этой величины.

Скажем, если в алгоритме одновременно с временем используется и такая величина, как температура, то для устранения трудностей можно ввести такие идентификаторы, как *time* и *temp*.

Как видно, АЛГОЛ допускает большую свободу действий и каждый автор алгоритма может выбрать такие обозначения, которые больше всего отвечают смыслу задачи.

Переменные. *Переменной* называют объект, обозначенный идентификатором, который в процессе вычислений может принимать какие-либо значения.

Под *значением* в АЛГОЛе понимается либо отдельное число (целое или вещественное), либо отдельное логическое значение, либо упорядоченная последовательность чисел одного и того же типа, либо упорядоченная последовательность логических значений, либо метка.

Если в качестве значения переменной выступает одно число или одно логическое значение, то такую переменную называют простой.

В качестве идентификаторов простых переменных выбирают общеупотребительные обозначения, например идентификаторы *a*, *b*, *s*, *beta* и т. д.

Так как числа, представляющие значения переменных, могут быть целыми или вещественными, то и переменные могут быть целыми или вещественными в зависимости от того, какие числовые значения они будут принимать в процессе вычислений. Целые и вещественные числа по-разному записываются, хранятся и обрабатываются в ЦВМ. Поэтому совместное использование целых и вещественных переменных в одном выражении не допускается.

Если в качестве значения объекта выступает некоторое конечное множество чисел или логических значений, то для обозначения каждой компоненты этого объекта используется переменная с индексами. Запись переменной с индексами состоит из идентификатора, за которым следует заключенный в квадратные скобки список индексов, представленный одним или несколькими арифметическими выражениями (в частном случае просто идентификаторами), разделенными запятыми. Например, запись

$x[i]$, $b[j, k]$, $q[16]$, $p[k+2, l-4]$

соответствует обычной математической записи

x_i, $b_{j,k}$, q_{16}, $p_{k+2,\,t-4}$.

Индексы переменной могут быть любыми целыми числами. Кроме того, индекс может быть задан и арифметическим выражением. Например,

$gamma\,[ln\,(2\times k-3),\ m+n-1]$.

Значения индексных выражений при вычислениях автоматически округляются до ближайшего целого числа, так как индекс по своему смыслу может быть только целым числом.

Конечная совокупность переменных с индексами, представленными своими значениями, образует в общем случае многомерный массив значений, который обозначается тем же идентификатором, что и составляющие его компоненты.

Массив характеризуется своей размерностью — количеством перечисленных в списке индексных выражений, значения которых определяют местоположение каждого элемента в массиве. Так, последовательность величин $x\,[i]$ при заданных значениях i является примером одномерного массива x. Если каждый элемент определяется двумя индексами — номерами строки и столбца, то совокупность этих элементов будет представлять собой матрицу, т. е. двумерный массив. Возможны и более сложные структуры, образующие многомерные массивы.

Простые переменные и переменные с индексами объединяются в единое понятие переменной.

Указатели функций. В алгоритмическом языке под *функцией* понимают совокупность правил, определяющих вычисление какого-либо значения в зависимости от некоторых переменных и выражений, называемых *фактическими параметрами*.

Так, зависимость $y = (ax^2 + bx)/\,(1 + x)$ может быть в наиболее формальном виде записана как $y = f\,(a, b, x)$, где символ f указывает на вполне определенные операции над аргументами и служит указателем данной функции.

Указатель функции определяет обращение к соответствующей процедуре, задающей данную функцию, для вычисления одного ее значения. Он состоит из идентификатора, определяющего функцию, вслед за которым в скобках записываются фактические параметры, отделяемые друг от друга запятыми, В качестве фактических параметров может быть любое арифметическое выражение, например

$f\,(z)$, $F\,(n, m, k)$, $log\,(x)$.

Для обозначения указателей функций можно использовать любые идентификаторы. Однако так же, как и в математике, за некоторыми наиболее употребительными функциями в языке АЛГОЛ закрепляются стандартные обозначения.

Стандартная функция записывается в виде стандартного идентификатора и аргумента, заключенного в круглые скобки. В АЛГОЛе имеется следующий набор стандартных функций:

abs (x) — абсолютное значение;

sign (x) — знак x, где sign $(x) = \begin{cases} 1 \text{ при } x > 0, \\ 0 \text{ при } x = 0, \\ -1 \text{ при } x < 0; \end{cases}$

sqrt (x) — квадратный корень из x;
sin (x) — синус (аргумент в радианах);
cos (x) — косинус (аргумент в радианах);
arctan (x) — главное значение (от $-\pi/2$ до $+\pi/2$);
ln (x) — натуральный логарифм;
exp (x) — показательная функция e^x;
entier (x) — целая часть x.

Функции *entier* и *sign* дают результат целого типа.

Аргументом функции может быть любое арифметическое выражение.

Выражения. В алгоритмическом языке любое *выражение* служит для определения некоторого значения и представляет собой компактную последовательную запись, включающую основные объекты языка (числа, переменные, указатели функций и т. д.), разделенные знаками операций. В зависимости от типа значения, вычисляемого по данному выражению, все выражения разделяют на арифметические, логические и именующие.

Арифметические выражения. Процесс решения задачи вычислительного характера в конечном счете сводится к выполнению некоторой последовательности арифметических операций над отдельными числами.

Поэтому понятие арифметического выражения является очень важным в АЛГОЛе, так как именно с помощью арифметических выражений и формулируются правила переработки исходных данных. Заметим, что это понятие очень близко к обычному, но только в АЛГОЛе оно более строго определено и существенно расширено.

В АЛГОЛе имеется два вида арифметических выражений: простые и условные. Каждое арифметическое выражение в АЛГОЛе задает правила для определения единственного значения, каковым является одно число — целое или вещественное.

В качестве арифметических операций используют операции сложения, вычитания, умножения, деления и возведения в степень. Для обозначения этих операций в АЛГОЛе применяют знаки «+», «—», «×», «/», «↑».

В качестве отдельных компонент арифметического выражения используют *первичные арифметические выражения*, которые в АЛГОЛе могут быть четырех видов: 1) число без знака; 2) переменная (простая или с индексом); 3) указатель функции; 4) арифметическое выражение, заключенное в круглые скобки.

Под *числом без знака* понимается либо целое число без знака, либо вещественное число без знака.

Простые арифметические выражения представляют собой либо одно первичное выражение, либо некоторую последовательность первичных выражений, связанных между собой знаками арифметических опера-

ций, причем перед самым левым первичным выражением может быть записан знак «+» или знак «—», например:

3.14;

$-x \uparrow 3+b$;

$a[i, j] \uparrow n+0.5 \times m$;

$ln(abs(x-y \times z))$.

При составлении арифметических выражений необходимо придерживаться следующих правил.

Два знака операций не могут стоять рядом; они должны быть разделены, по крайней мере, круглыми скобами.

Если целое число возводится в целую степень (например, $i \uparrow k$), то результатом операции является также целое число; во всех других случаях возведение в степень дает вещественный результат. Возведение в степень производится многократным умножением. Если показатель степени является действительным числом, то используются логарифмы.

Для того чтобы вычислить значение простого арифметического выражения, необходимо взять текущие значения входящих в него первичных выражений и выполнить заданные арифметические операции. При этом в АЛГОЛе принято обычное старшинство выполнения арифметических операций. В первую очередь должна быть выполнена операция возведения в степень «$\uparrow$», затем — операции умножения «$\times$» и деления «/», которые имеют одинаковое старшинство и только потом — операции сложения «+» и вычитания «—», которые равноценны.

Если в простом арифметическом выражении подряд записано несколько операций одного и того же старшинства, то они выполняются в порядке их следования слева направо. Так, например, арифметическое выражение $x \uparrow n \uparrow 2$ вычисляется в последовательности x^n, а затем $(x^n)^2$. Запись же $x/y \times z$ произвольно можно интерпретировать как $\frac{x}{y \cdot z}$, однако на самом деле она соответствует выражению $\frac{x}{y} \cdot z$.

Иногда необходимо задать порядок выполнения операций, отличный от их обычного старшинства, например, при задании вычислений по формуле $\frac{x}{y+z}$ сначала нужно осуществить операцию сложения, а затем деления. В таких случаях в АЛГОЛе для указания порядка выполнения арифметических операций используются круглые скобки, потому что по определению любое арифметическое выражение, заключенное в круглые скобки, есть первичное выражение.

Поскольку значение первичного выражения должно быть вычислено до того, как будут выполняться арифметические операции, которые связывают его с другими первичными выражениями, то сначала должны выполняться те действия, которые указаны в круглых скобках. Например, $\frac{(a^2+2)^3}{p-\ln x}$ запишется на языке АЛГОЛ, как

$(a \uparrow 2+2) \uparrow 3/(p-\ln(x))$.

Условные арифметические выражения включают в себя, кроме простых арифметических выражений, еще дополнительные условия, определяющие, когда должны использоваться представленные в них простые арифметические выражения. Примером условного арифметического выражения может служить следующая запись:

$$y=\begin{cases} ax^2+b & \text{при } x\geqslant 2; \\ \sqrt{x+b} & \text{при } x<2. \end{cases}$$

Для описания подобного рода разветвленных вычислений в АЛГОЛе вводится специальное понятие *условие*. С точки зрения синтаксиса условием называют запись вида

if *B* **then**,

где **if (если)** и **then (то)** являются основными символами языка из числа ограничителей, смысл которых очевиден, а буквой *B* обозначено логическое выражение, которое может принимать одно из двух возможных логических значений **true (истина)** или **false (ложь)**.

Условие считается выполненным, если фигурирующее в нем логическое выражение *B* принимает значение **true (истины)** при текущих значениях входящих в него компонент.

Если это логическое выражение принимает значение **false (ложь)**, то считается, что условие не выполнено.

Условия используются при образовании разных конструкций языка АЛГОЛ, в том числе и для образования условных арифметических выражений.

Условные арифметические выражения позволяют в простой и компактной форме представить некоторые разветвляющиеся вычислительные процессы. С точки зрения синтаксиса условным арифметическим выражением называют запись вида:

if *B* **then** *A1* **else** *A*,

где символ **else (иначе)** относится к группе ограничителей и является основным символом языка АЛГОЛ; *B* — логическое выражение; *A1* — простое арифметическое выражение; *A* — произвольное арифметическое выражение.

В условном арифметическом выражении после условия должно идти обязательно простое арифметическое выражение, а после символа **else** — любое арифметическое выражение. Структуру условного арифметического выражения нетрудно понять, если сначала в этой записи в качестве арифметического выражения *A* использовать только простые арифметические выражения:

if *B* **then** *A1* **else** *A2*,

где *A1*, *A2* простые арифметические выражения. Например,

if $x \neq 0$ **then** $x \uparrow 3$ **else** 0

или

if *abs* $(a+b) < c$ **then** $a \times b$ **else** a/c.

Если же после символа **else** записать условное арифметическое выражение, то структура выражения будет следующей:

if *B1* **then** *A1* **else if** *B2* **then** *A2* **else** *A3*,

где *A1*; *A2*; *A3* — простые арифметические выражения.

Продолжая этот процесс, если есть необходимость, можно получать все более сложные конструкции условных арифметических выражений. Каждое условное арифметическое выражение, как и простое, определяет единственное значение.

Правила вычисления значения условного арифметического выражения следующие.

Проверяется первое условие, если это условие выполнено, то в качестве значения всего условного арифметического выражения принимается значение простого арифметического выражения *A1*; если условие не выполнено, то в качестве значения всего условного выражения принимается значение арифметического выражения *A*, которое следует за символом **else**; если *A* в свою очередь является условным выражением, то для вычисления его значения следует пользоваться тем же правилом. Например, требуется записать арифметическое выражение:

$$\begin{cases} a+b, & \text{если } -\infty \leqslant x < a; \\ \sqrt{a^2+b^2 x}, & \text{если } a \leqslant x < b; \\ \dfrac{ab}{x}, & \text{если } b \leqslant x < +\infty. \end{cases}$$

На языке АЛГОЛ оно будет записано так:

if $x < a$ **then** $a+b$ **else**
if $x < b$ **then** *sqrt* $(a \uparrow 2+b \uparrow 2 \times x)$
else $a \times b/x$.

Логические выражения в отличие от арифметических выражений используются для вычисления логических значений переменных или выражений и строятся с помощью логических операций и операций отношения. При этом все логические выражения или переменные могут принимать лишь одно из двух значений — **true** (**истина**) или **false** (**ложь**).

Наиболее употребительным логическим выражением является отношение, которое представляет собой два простых арифметических выражения, соединенных знаком отношения, например, $a \times y < z \uparrow 2$; $x + 5 \leqslant 2$.

Если при заданных значениях величин, входящих в арифметические выражения, условие, определяемое отношением, выполняется, то отношение принимает значение **true**, в противном случае — **false**.

Отношения, логические значения **true** и **false**, логические переменные (а также произвольные логические выражения в круглых скобках) называют *первичными логическими выражениями*.

Так же как и арифметические, логические выражения могут быть простыми и условными.

Простое логическое выражение строится из первичных с помощью знаков логических операций $\neg$ (НЕ), $\wedge$ (И), $\vee$ (ИЛИ), $\supset$ (импликация) и $\equiv$ (эквивалентность).

При этом истинность простого логического выражения, например, состоящего из элементарных логических переменных a и b или только a в зависимости от вида логических операций, определяется табл. 5.1, где истинность (**true**) изображается символом «1», а ложность (**false**) — символом «0».

Таблица 5.1

Переменные	Значения переменных			
a	0	0	1	1
b	0	1	0	1
$\neg a$	1	1	0	0
$a \wedge b$	0	0	0	1
$a \vee b$	0	1	1	1
$a \supset b$	1	1	0	1
$a \equiv b$	1	0	0	1
Логическая операция	Результат логической операции			

Например, два элементарных высказывания $x > 5$ и $x < 10$ могут быть объединены с помощью операции логического умножения ($\wedge$). Получаемое при этом логическое выражение $x > 5 \wedge x < 10$ является истинным, если x находится внутри числового интервала с концами 5 и 10, и ложным — в противоположном случае.

Логическое выражение $x < 5 \vee x > 10$ является истинным, если x находится вне указанного числового интервала, и ложным, если x принадлежит этому числовому интервалу.

При построении логических выражений, состоящих более чем из двух логических выражений, логические операции должны выполняться последовательно в порядке их записи слева направо с учетом правил старшинства.

Правила старшинства логических операций определяются списком последних, где каждая операция является старшей по отношению к операциям, записанным справа от нее: $\neg \wedge \vee \supset \equiv$.

Использование скобок при построении сложных логических выражений аналогично использованию скобок при построении сложных арифметических выражений. Так, для определения истинности выражения $(a \vee b \supset c) \vee (\neg b \vee d \wedge e)$ вначале должна быть определена истинность выражений, заключенных в скобки, в соответствии с правилами старшинства операций, а затем уже истинность всего выражения в целом.

Условное логическое выражение по правилам синтаксиса имеет такую же структуру, как и условное арифметическое выражение:

if *B1* **then** *B2* **else** *B3*,

где *B1*, *B3* — произвольные логические выражения; *B2* — простое логическое выражение.

Именующие выражения. Вычислительный процесс редко бывает *линейным*, т. е. таким, когда его этапы выполняются последовательно один за другим, в порядке их написания. В алгоритм всегда включаются некоторые условия, которые определяют последовательность выполнения этапов вычислительного процесса, управляют ходом вычислений. Так, например, бывает, что после выполнения i-го этапа необходимо перейти к выполнению не следующего за ним $(i + 1)$-го этапа, а к выполнению некоторого другого l-го этапа, расположенного до i-го или после $i + 1$ этапов.

Для того чтобы осуществить такой переход, необходимо иметь какой-либо признак. Этот признак в языке АЛГОЛ-60 называют *меткой* и записывают впереди оператора, определяющего некоторый этап вычислений. Метка от оператора отделяется двоеточием (:). В качестве метки может использоваться любой идентификатор.

Иногда метка оператора, к которому необходимо перейти после выполнения i-го этапа, заранее неизвестна и может быть определена только после его выполнения.

Выражение, с помощью которого производится определение направления перехода к следующему этапу вычислений, называют *именующим*.

В качестве простого именующего выражения может быть либо метка, либо указатель переключателя.

Условное именующее выражение строится по известной структуре

if *B* **then** *R1* **else** *R2*,

где *R1* — простое именующее выражение, а *R2* — любое именующее выражение. Например, именующее выражение может быть задано в виде одной метки *m*, или в виде условного выражения:

if $a < b$ **then** *m* **else** *n*

Здесь логическое выражение $a < b$ вместе с метками *m* и *n* составляет именующее выражение, которое определяет направление перехода к оператору, отмеченному меткой *m* при $a < b$, и к оператору, отмеченному меткой *n* при $a \geqslant b$.

В качестве простого именующего выражения используют указатель переключателя, который имеет вид переменной с индексом. Идентификатор этой переменной совпадает с идентификатором переключателя, а индекс указывает порядковый номер метки, представленной в списке меток описания переключателя. Например, именующие выражения в виде указателей переключателей $p[i]$ и $q[j]$ определяют порядок перехода к операторам, имеющим метки, номера которых в описании переключателей соответствуют значениям индексов i и j.

§ 5.2. ЭЛЕМЕНТЫ ОПИСАНИЯ АЛГОРИТМИЧЕСКИХ ПРОЦЕССОВ

Любая запись арифметического или логического выражения обозначает порядок и тип действий, которые необходимо произвести над переменными, входящими в это выражение. Однако выражение само

по себе не содержит никакой инструкции, предписывающей выполнение этих действий. Такой инструкцией в алгоритмическом языке является *оператор*.

Программа решения задачи на языке АЛГОЛ представляет собой линейную последовательность операторов S; S; S; ...; S, задающих порядок выполнения и тип операций, которые необходимы для решения поставленной задачи. Один оператор от другого отделяется разделителем «;».

Самые разнообразные алгоритмы решения задачи могут быть описаны с помощью алгоритмического языка конечным числом операторов. В языке АЛГОЛ различают восемь типов операторов. Рассмотрим их

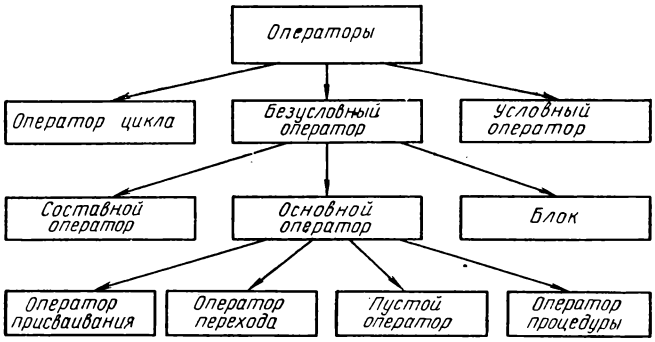


Рис. 5.1. Состав операторов языка АЛГОЛ-60

классификацию и дадим некоторые определения. С точки зрения структуры одни из операторов являются *элементарными*, т. е. в их «теле» нет более элементарных операторов. К ним относят операторы: присваивания; перехода; процедуры; пустой. Эти операторы *основные* в языке АЛГОЛ.

Операторы, в «теле» которых содержатся более элементарные операторы, являются *производными*.

Если оператор состоит из более элементарных операторов, описывающих некоторую совокупность действий, которая должна быть выполнена полностью и один только раз, то этот оператор называют *составным*.

При многократном последовательном выполнении указанной совокупности действий составной оператор представляет собой *оператор цикла*.

Если в зависимости от определенных условий выполняются не все элементарные операторы, входящие в составной оператор, а только некоторые из них, то такой оператор является *условным*.

Более сложным оператором будет *блок*, включающий в себя описание типов используемых переменных.

Связь между рассмотренными операторами представлена на рис. 5.1.

Оператор присваивания. Наиболее распространенным в алгоритмическом языке является оператор присваивания, который служит для

присваивания одной или нескольким переменным конкретного значения, полученного в результате вычисления выражения.

С точки зрения синтаксиса оператор присваивания есть запись вида $v := A$, где v — переменная; A — какое-либо арифметическое или логическое выражение. Символ присваивания : = является основным символом языка АЛГОЛ из числа ограничителей; он означает «присвоить значение вычисленного выражения одной или нескольким переменным, стоящим в левой части». Например,

$y := a \times b \times ln(x) \uparrow n;$

$u :=$ **if** $sin(x1) > sin(x2)$ **then** $x1$ **else** $x2.$

При выполнении оператора присваивания имеется в виду, что в выражении, стоящем справа от символа :=, всем переменным уже присвоены конкретные числовые значения предыдущими операторами и вычисление выражения может быть выполнено.

В отличие от математических формул при использовании оператора присваивания возможна запись вида $y := y + a$, которая соответствует выражению $y_{i+1} = y_i + a$, где y_i — значение переменной до выполнения оператора; y_{i+1} — новое значение переменной после выполнения вычислений. Этот пример наглядно показывает, что символ := не эквивалентен знаку равенства.

Значение одного и того же арифметического выражения может быть присвоено сразу нескольким переменным. В этом случае идентификаторы переменных, которым необходимо присвоить определенное арифметическое выражение, записываются последовательно. После каждого идентификатора ставится знак присваивания. Арифметическое выражение записывается за последним знаком присваивания. Например,

$y := x[i] := z[j, k] := sqrt(abs(b)).$

В этом случае необходимо, чтобы все переменные, входящие в список левой части оператора присваивания, имели один и тот же тип — **real** или **integer**, а если правую часть составляет логическое выражение, то тип **Boolean**.

Оператор присваивания выполняется в три этапа:

1) если в списке левой части есть переменные с индексами, то определяются значения этих индексов;

2) вычисляется значение выражения, стоящего в правой части оператора;

3) полученное значение присваивается всем переменным списка левой части.

Сложные арифметические выражения при программировании могут быть разбиты на отдельные части, каждая из которых должна быть обозначена определенной переменной и описана отдельными операторами присваивания. Вычисление всего арифметического выражения в этом случае описывается последовательностью операторов присваивания. Например, арифметическое выражение

$$z = |ab^3 + e^{x+y}| / \sqrt{a + \ln(c + d)}$$

можно разбить на два арифметических выражения (числитель и знаменатель) и обозначить их соответственно через переменные p и q. Тогда вычисление переменной z будет представлено последовательностью трех операторов присваивания:

$p := abs(a \times b \uparrow 3 + exp(x + y))$;

$q := sqrt(a + ln(c + d))$;

$z := p/q$.

Оператор перехода. В АЛГОЛе принят естественный порядок выполнения операторов (операторы обычно выполняются в той последовательности, как они записаны).

Однако иногда естественный порядок выполнения операторов должен быть нарушен. Например, в какой-либо момент может понадобиться осуществить переход с тем, чтобы еще раз выполнить некоторую последовательность операторов, или наоборот может возникнуть необходимость обойти какую-либо группу операторов. Для этого в АЛГОЛе имеются операторы перехода.

Поскольку для указания нужного преемника приходится ссылаться на те или иные операторы, то все операторы, которые могут быть возможными преемниками оператора перехода, помечают или снабжают метками. Помеченный оператор имеет структуру $m : S$, где S — помеченный оператор; m — метка. Метка является как бы наименованием оператора и поэтому ссылаются на этот оператор посредством указания его метки.

Функции оператора перехода состоят в том, что он сам выбирает себе преемник, а для определения нужного оператора в нем указывается соответствующая метка.

В этом случае оператор перехода имеет вид

go to m,

где **go to** — основной символ языка; m — метка оператора, к которому необходимо совершить переход. Например,

$z := a/b$; **go to** $m8$; $x := z - c$;

$m8 : y := x + ln(z)$.

Эта запись означает, что следующий за оператором перехода оператор присваивания $x := z - c$ выполняться не будет и запись эквивалентна операторам присваивания:

$z := a/b$; $y := x + ln(z)$.

В общем случае оператор перехода может быть представлен символом **go to**, после которого записывается именующее выраженне.

В некоторых программах необходимо передать управление из одного места в различные места в зависимости от выполнения тех или иных условий. В этих случаях можно применять указатель переключателя. Так, оператор перехода

go to $p[i]$

в зависимости от значения индекса передает управление различным операторам программы, метки которых определяются порядковым номером списка меток, представленного в описании переключателя. Например, при $i = 2$ выбирается вторая метка списка, а при $i = 5$ — соответственно пятая.

Пустой оператор. При написании пустой оператор никак не обозначается и не предписывает выполнения каких-либо действий. Однако он необходим в некоторых случаях, например, для обеспечения конца вычислений или пояснений о том, что в некоторой ситуации не нужно выполнять действий.

Операторы стандартных процедур ввода—вывода. К операторам процедур, предписывающим выполнение обособленных и наиболее употребительных алгоритмов, относятся операторы стандартных процедур ввода в машину исходных данных и вывода из нее результатов вычислений. Эти и подобные им операторы стандартизируют обращение к программам, заранее составленным и записанным в памяти машины с целью вызова их для выполнения необходимых действий на соответствующем этапе алгоритма решения задачи.

Рассмотрим в общих чертах правила записи операторов ввода данных и печати результатов, поскольку в каждом конкретном представлении языка АЛГОЛ эти операторы имеют свои особенности, обусловленные характеристиками устройств ввода—вывода.

Оператор «ввод» осуществляет ввод в память ЦВМ чисел и логических значений и имеет общий вид:

ввод (список фактических параметров);

Список фактических параметров (объектов ввода) представляет собой последовательность разделенных запятыми идентификаторов массивов и простых переменных. Например,

ввод (*a*, *b*, *c*, *s*, *p*, *q*);

Каждому фактическому параметру должна соответствовать одна группа данных. Простым переменным и идентификаторам массивов соответствуют группы числовых или логических данных. Количество данных в группе не должно превышать размеров объектов ввода, которые определяются описаниями, но может быть меньше их.

Группы числовых данных и логических значений, вводимые в машину с помощью оператора «*ввод*», переносятся на перфоносители (перфокарты или перфоленту) и помещаются в устройстве ввода машины после перфокарт или перфоленты, содержащих АЛГОЛ-программу решения задачи. В результате выполнения оператора «*ввод*» весь массив информации с перфокарт или перфоленты считывается в память машины и объектам ввода, перечисленным в списке фактических параметров, присваиваются соответствующие числовые и логические значения.

Стандартный оператор «*печать*» предназначают для организации вывода чисел, логических значений и текста на печатающее устройство машины и редактирования печатаемой информации. Общий вид оператора следующий:

печать (список фактических параметров);

Список фактических параметров оператора «*печать*» представляет собой одну или несколько разделенных запятыми групп, каждая из которых задает печать числовых или логических значений или текста алфавитно-цифровой информации. Например,

печать (*x*, *y*, *z*);

Действие оператора «*печать*» заключается в организации печати с помощью печатающего устройства машины значений переменных и массивов, которыми они обладают непосредственно перед выполнением оператора. Значение переменной печатается в виде одного десятичного числа, значение массива — в виде последовательности чисел, являющихся значениями его компонент. Значение одного параметра от другого отделяется интервалом.

Вид операторов «*ввод*» и «*печать*» зависит от конструкции конкретной машины, ее входных и выходных устройств, а также от конкретных трансляторов. Практически это выражается в различных способах записи данной инструкции и списков фактических параметров. Поэтому перед работой на конкретной машине необходимо познакомиться с допустимыми правилами записи этих операторов.

Приведем в качестве примера фрагмент программы на языке АЛГОЛ-60, описывающий определение значения площади треугольника, если известны длины его сторон:

```
ввод (a, b, c);
p: = (a + b + c)/2;
s: = sqrt (p × (p — a) × (p — b) × (p — c));
печать (s)
```

Операторы присваивания, перехода, пустой и ввода — вывода позволяют описать на языке АЛГОЛ-60 алгоритм практически любой вычислительной задачи. Все конструкции алгоритмов, изложенные в дальнейшем с помощью других операторов, обеспечивают лишь большее удобство записи программы вычислительных процессов в смысле ее наглядности и компактности.

§ 5.3. ОРГАНИЗАЦИЯ ВЕТВЯЩИХСЯ И ЦИКЛИЧЕСКИХ ПРОЦЕССОВ

Для большинства встречающихся на практике вычислительных процессов характерна разветвляемость: когда для дальнейшего исполнения возможны переходы к различным последовательностям операторов в зависимости от каких-либо условий. Для организации такого рода процессов в АЛГОЛе служит условный оператор и оператор цикла.

Условный оператор. С точки зрения синтаксиса условным оператором является запись вида

if *B* **then** *S*1 **else** *S*,

где *S1* — безусловный оператор, а *S* — произвольный оператор. Таким образом, условный оператор всегда имеет такую структуру: условие, безусловный оператор *S1*, символ **else**, произвольный оператор *S*.

Смысл условного оператора заключается в следующем: при выполнении условного оператора выбирается один из входящих в него операторов (либо оператор *S1*, либо оператор *S*). Выбор оператора производится в зависимости от условия: если условие выполняется на самом деле, то выбирается оператор *S1*, а оператор *S* пропускается; если условие не выполняется, то выполняется оператор *S*, а оператор *S1* пропускается.

При определении структуры условного оператора можно использовать те же приемы, что и при рассмотрении условных арифметических выражений. В частности, в качестве оператора *S* можно сначала брать только безусловные операторы, тогда получится следующая структура:

if *B* **then** *S*1 **else** *S*2,

где *S*1 и *S*2 — безусловные операторы. Например,

if $a<(b+c)$ **then** $x:=x+a$ **else go to** m.

При выполнении этого оператора, если текущее значение a будет меньше значения $(b+c)$, то в результате выполнения всего условного оператора значение a будет добавлено к x. Если же текущее значение a будет больше значения $(b+c)$, то будет выполняться оператор перехода

go to m.

Оператор *S* может быть любым, в том числе и условным. Например, в качестве оператора *S* можно взять условный оператор только что рассмотренного вида (правила его выполнения остаются теми же самыми). Поэтому, если имеется несколько условий, то с помощью одного условного оператора можно определить всю необходимую последовательность действий:

if *B*1 **then** *S*1 **else if** *B*2 **then** *S*2 **else** *S*3,

где *S*3 — может быть условным оператором.

В частности, если нужно решить задачу определения значений функции y по различным формулам в зависимости от интервалов, в которые попадает значение аргумента:

$$y=\begin{cases} ax^2, & \text{если } -\infty<x\leqslant x_1;\\ a+x, & \text{если } x_1<x\leqslant x_2;\\ a/x, & \text{если } x_2<x\leqslant x_3;\\ \sqrt{x}, & \text{если } x_3<x<+\infty, \end{cases}$$

то условный оператор, определяющий порядок выполнения действий в зависимости от значения x, будет представлен в следующем виде:

if $x \leqslant x1$ **then** $y := a \times x \uparrow 2$ **else**
if $x \leqslant x2$ **then** $y := a + x$ **else**
if $x \leqslant x3$ **then** $y := a/x$ **else** $y := sqrt\,(x)$

Довольно распространенным является такой случай, когда на каком-либо этапе нужно производить вычисления только при выполнении определенного условия. И если это условие не выполняется, то вообще никаких действий производить не требуется.

Например, если очередной этап заключается в том, чтобы текущее значение x заменить его абсолютным значением, то надо либо выполнить оператор присваивания $x := -x$, если текущее значение x отрицательно, либо x оставить неизменным в противном случае, поскольку неотрицательное значение одновременно является и модулем числа. Такая возможность предусмотрена в общей схеме условного оператора, так как любой из операторов (оператор S или оператор $S1$) может быть пустым оператором.

Подобная ситуация достаточно часто встречается на практике, поэтому в АЛГОЛе предусмотрен специальный вид условного оператора, имеющий следующую структуру:

if B **then** $S1$.

Это означает, что если условие B выполняется, то необходимо выполнить оператор $S1$, а если условие не выполняется, то весь условный оператор будет эквивалентен пустому оператору. Такой вид условного оператора называют *оператор типа «если»*.

С помощью условного оператора не сложно описать процесс решения квадратного уравнения, соответствующий алгоритму, представленному на рис. 3.4

```
ввод (a, b, c);
D:=b↑2-4×a×c;
g:=2×a;
if D<0 then go to m5;
p:=sqrt (D); x1:=(-b+p)/g;
x2:=(-b-p)/g; печать (x1; x2);
go to m7;
m5: alpha:=-b/g; beta:=sqrt (abs (D))/g;
печать (alpha, beta);
m7;
```

С точки зрения синтаксиса в условном операторе после условия и после символа **else** допускается только по одному оператору. Однако в ряде случаев в зависимости от условия нужно выполнить или пропустить некоторый этап вычислений, состоящий из нескольких операторов.

Составной оператор. С точки зрения синтаксиса составной оператор имеет следующую структуру:

begin S; S; S; ... ; S **end**,

где **begin** и **end** — основные символы языка из числа ограничителей (операторные скобки), а S — операторы.

Смысл составного оператора состоит в том, что последовательность операторов, заключенная в операторные скобки, рассматривается как отдельный безусловный оператор.

Выполнение составного оператора сводится к выполнению последовательности операторов, входящих в него, в том порядке, в котором они записаны.

Пример. Пусть задан вектор x, состоящий из двух компонент x_1 и x_2. Задача очередного этапа состоит в том, чтобы упорядочить компоненты этого вектора по их возрастанию. Если окажется, что $x_1 \leqslant x_2$, то делать ничего ненужно, так как компоненты уже упорядочены в соответствии с условием. Если же это неравенство не выполняется, т. е. $x_1 > x_2$, то необходимо значения этих компонент поменять местами, что реализуется следующей последовательностью операторов:

$p := x[1]; x[1] := x[2]; x[2] := p.$

Как видно, здесь в зависимости от текущих значений компонент нужно выбрать для исполнения или пропустить уже не один, а три оператора вместе. Для этого используют составные операторы:

if $x[1] > x[2]$ **then**

begin $p := x[1]; x[1] := x[2]; x[2] := p$ **end.**

Если $x_1 > x_2$ должен выполняться составной оператор, состоящий сразу из трех операторов присваивания, а если $x_1 \leqslant x_2$, то действие условного оператора эквивалентно пустому оператору.

Оператор цикла. В вычислительных процессах часто возникает необходимость повторения той или иной части программы с изменением значений некоторых величин. Практически каждый алгоритм переработки числовой или логической информации включает в себя неоднократные повторения одного или нескольких действий. Такой многократно повторяющийся вычислительный процесс называют *циклическим (циклом)*.

Для описания циклических процессов могут быть использованы условные операторы. Однако в языке АЛГОЛ для этой цели есть специальный оператор — оператор цикла, дающий более компактную и наглядную запись. Оператор цикла указывает, какая величина должна изменяться при повторении части программы (параметр цикла), обеспечивает изменение параметра цикла нужное число раз и содержит описание повторяемой части вычислительного процесса.

Оператор цикла имеет следующую структуру:

for v: = {список цикла} **do** S.

Он состоит из заголовка и тела цикла и представлен основными символами АЛГОЛа **for** (**для**) и **do** (**выполнить**), идентификатором параметра цикла v и списком цикла.

Список цикла состоит из элементов, отделенных друг от друга запятой. Каждый элемент описывает закон изменения параметра цикла.

Заголовок цикла обеспечивает изменение только одной величины. После символа **do** следует тело цикла S, представленное одним оператором, описывающим повторяемую часть вычислительного процесса.

Тело цикла S исполняется столько раз, сколько укажет заголовок цикла, если внутри тела не содержится условий, прекращающих выполнение оператора цикла. Если в цикле нужно повторить несколько операторов, то они объединяются операторными скобками в один составной оператор.

Оператор цикла выполняется следующим образом. Параметру цикла v присваиваются значения, определяемые первым, вторым и т. д. элементами списка цикла. При каждом присваивании, если нужно, проверяется необходимость выполнения оператора, записанного после символа **do**. Выполнение оператора цикла заканчивается, если список цикла исчерпан или если оператор, стоящий после **do**, содержит оператор, осуществляющий выход из оператора цикла.

Рассмотрим, каким образом изменяется значение параметра цикла и как определяется число повторений. Выполнение этих функций обеспечивает список цикла, каждый элемент которого может быть записан в одном из следующих видов:

A — арифметическое выражение;

A **step** h **until** D — элемент типа арифметической прогрессии;

A **while** B — элемент типа пересчета.

Цикл с элементами типа арифметического выражения. Циклу этого типа соответствует первый вид элемента списка — арифметическое выражение. Его значение при выполнении оператора цикла присваивается параметру цикла, и с этим значением выполняется оператор, стоящий после символа **do**.

Оператор цикла с элементами типа арифметического выражения записывается в виде

for v: = $A1$, $A2$, ..., An **do** S.

Содержание вычислительного процесса можно сформулировать так: для переменной v, принимающей последовательно значения $A1$, $A2$, ..., An, выполнять оператор S.

Действие оператора цикла эквивалентно действию следующей последовательности операторов:

for v: = $A1$ **do** S;

for v: = $A2$ **do** S;

. . . . . .

for v : = An **do** S.

Пример. Вычислить таблицу значений функции $y=\sqrt{|\ln(1-x)|}$ при $x=0{,}5$; 0,635; 0,7; 0,718; 0,92.

Программа выглядит следующим образом:

for x: = 0.5, 0.635, 0.7, 0.718, 0.92 **do**

y: = $sqrt\,(abs\,(ln\,(1-x)))$.

Цикл с элементами типа арифметической прогрессии. Часто параметр цикла имеет вполне определенную зависимость изменения. Например, он может изменяться от не-

которого начального значения A до конечного значения с шагом h, т. е. по закону

$$v_1 = A;$$
$$v_2 = A + h;$$
$$v_3 = A + 2h;$$
$$v_n = A + (n - 1)\,h.$$

В этом случае значения параметров образуют арифметическую прогрессию.

Оператор цикла записывается в виде:

for $v := A$ **step** h **until** D **do** S

и читается так: для переменной v, изменяющейся от A с шагом h, пока $v \leqslant D$ выполнять оператор S. Смысл такого оператора цикла состоит в том, что переменной v присваивается последовательность значений арифметических выражений A, $A + h$, $A + 2h$ и т. д. Каждый раз значение параметра увеличивается на значение, определяемое по выражению h, и после очередной проверки условия $(v - D) \times \times \operatorname{sign}(h) \leqslant 0$ производится выполнение тела цикла S.

Как только очередное значение переменной v перейдет границу, заданную арифметическим выражением D, выполнение оператора цикла заканчивается.

Формально описанный процесс реализации оператора цикла в этом случае может быть представлен следующей записью:

$v := A$;
n : **if** $(v - D) \times sign\,(h) > 0$ **then go to** m;
S; $v := v + h$; **go to** n;
m:

Пример. Записать алгоритм вычисления последовательности значений функции

$$x = u/\sqrt{r^2 + (2\pi fL - 1/(2\pi fC))^2}$$

при изменении параметра r от 0,1 до 1,0 с шагом $h = 0{,}05$

for $r := 0.1$ **step** 0.5 **until** 1.0 **do**
$x := u/sqrt\left(r \uparrow 2 + (2 \times 3{\cdot}14 \times f \times L - 1/(2 \times 3\,14 \times f \times C)) \uparrow 2\right)$

Шаг цикла может быть переменным, если оператор, идущий после **do**, изменяет значения тех переменных, от которых зависит арифметическое выражение, стоящее после символа **step**. Например, в операторе цикла

for $z := a$ **step** $b + l$ **until** $a + 3 \times d$ **do**
if $x\,[z] < 0$ **then begin** $y\,[z/4] := 0$;
$b := b/3$; $l := l/3$ **end else** $y\,[z/4] := x\,[z] + h$

шаг уменьшается втрое всякий раз, как только при выполнении условного оператора встречается отрицательное значение выражения $x\,[z]$.

Используя оператор цикла с элементом типа арифметической прогрессии, можно в компактной форме записывать алгоритмы решения циклических задач, связанных, в частности, с последовательным выбором элементов массивов данных.

Так, для решения задачи 3.5, алгоритм которой изображен на рис. 3.5, программа в АЛГОЛе может иметь следующий вид:

```
ввод (alpha);
for x: = 1 step 1 until 100 do
begin y: = sin (x × alpha)/x; печать (y) end,
```

а для задачи 3.7 (рис. 3.8)

```
ввод (x);
for i: = 1 step 1 until 100 do
y [i]: = exp (x [i])/sqrt (1 + x [i]).
```

Цикл с элементами типа пересчета. В рассмотренных разновидностях оператора цикла число повторений оператора S всегда можно подсчитать до начала его выполнения, т. е. оно заранее известно. Но на самом деле это обстоятельство имеет место не всегда. В ряде случаев к моменту начала вычислений по оператору цикла нельзя сказать, сколько раз будет выполнено тело цикла. Для описания такого рода вычислительных процессов в АЛГОЛе служит оператор цикла, имеющий следующую структуру:

```
for v: = A while B do S,
```

где A — арифметическое выражение для определения значения параметра v; B — логическое выражение.

Смысл выполнения такого оператора цикла можно пояснить, используя основные операторы АЛГОЛа:

```
m : v : = A;
if B then begin S; go to m end.
```

Эта запись аналогична выражению «для параметра цикла v, принимающего значение A, выполнять тело цикла до тех пор, пока выполняется условие, указанное в заголовке цикла». Примером использования оператора подобного вида является алгоритм вычисления квадратного корня $x = \sqrt{a}$ с заданной точностью $\varepsilon > 0$ по итерационному методу Ньютона.

Формула для вычислений по этому методу записывается в виде: $x_{i+1} = x_i + (a/x_i - x_i)/2$ или $x_{i+1} = x_i + q_i$, где q_i — поправка при очередном вычислении значения корня; x_0 — заданное (или выбранное) начальное приближение.

Из формулы видно, что каждое вычисленное значение x равно его предыдущему значению плюс очередная поправка q_i, причем для достижения заданной точности неизвестно, сколько итераций необходимо сделать.

Алгоритм вычисления квадратного корня можно записать на языке АЛГОЛ следующим образом:

```
x:=xo;
for q:=(a/x−x)/2 while abs(q) ⩾ eps do
x:=x+q
```

Очевидно, что при каждом новом повторении рассмотренного процесса новые значения добавки q получаются до тех пор, пока не будет достигнута заданная точность вычисления корня.

Как только выражение B примет значение **false**, элемент списка цикла типа пересчета считается исчерпанным, и если после него нет других элементов (т. е. стоит основной символ **do**), то выполнение цикла заканчивается. Например, оператор цикла

```
for k:=l+4 while l < 25 do
begin x[k]:=5; l:=l+3 end
```

выполняется до тех пор, пока l будет меньше 25. Если перед выполнением оператора цикла l было равно 2, то составной оператор выполняется восемь раз, и переменной x с индексами 6, 9, ..., 27 будет присвоено значение пять.

Как уже отмечалось, оператор цикла может закончить работу двумя способами, когда список цикла исчерпан или когда оператор, стоящий после **do**, осуществил переход к другому оператору вне цикла. В первом случае параметр цикла после выхода из него не определен, а во втором случае параметр цикла имеет вне оператора цикла значение, которое он имел в момент выхода. Так, в предыдущем примере после окончания работы оператора цикла значение k будет не определено (а не равно 27).

Тело цикла, так же как условный и составной операторы, содержит в себе внутренние операторы. Каждый внутренний оператор может быть помечен. При этом имеет место следующая особенность обращения к помеченным операторам: запрещается обращение к ним из вне оператора цикла, так как в оператор цикла можно войти только через его начало, например, запись

```
go to m; for i:=1 step 1 until 10 do
begin x:=a↑2+b; m:y:=Q[i]+R[i]+x end
```

недопустима, так как первый оператор перехода ведет внутрь оператора цикла.

Воспользуемся элементом списка цикла типа пересчета для описания алгоритма вычисления суммы бесконечного ряда с заданной точностью ε (см. рис. 3.12).

```
ввод (x, eps);
S:=1; H:=1; N:=1;
for H:=H×x/N while H > eps do
begin S:=S+H; N:=N+1 end;
печать (S)
```

Как видно из фрагмента приведенной программы, накопление суммы S ряда происходит до тех пор, пока не будет достигнута заданная точность ее вычисления. После этого искомое значение суммы ряда выдается на печать.

В соответствии с общей синтаксической структурой оператора цикла его тело может быть представлено любым оператором, в том числе и оператором цикла, что позволяет достаточно гибко использовать оператор цикла для организации сложных циклических программ, вложенных друг в друга. Например, для двойного цикла по разным параметрам v и u структура оператора цикла записывается так:

```
for v:=A1 step A2 until A3 do
for u:=A4 step A5 until A6 do S.
```

Приведенную запись можно использовать для описания алгоритма решения задачи 3.6 в соответствии с блок-схемой алгоритма (см. рис. 3.7):

```
ввод (x, y);
for i:=1 step 1 until 10 do
for j:=1 step 1 until 20 do
z[i, j]:=x[i]×y[j];
```

По мере исчерпывания элемента списка внутреннего цикла программа будет изменять параметр внешнего цикла до тех пор, пока не выберет все его возможные значения.

Проиллюстрируем возможности оператора цикла для вычисления полинома n-й степени на основе формулы Горнера. Воспользуемся алгоритмом решения этой задачи, представленным на рис. 3.13

```
ввод (n, m);
ввод (a, x);
for j:=1 step 1 until m do
begin y:=a[0];
for i:=1 step 1 until n do
y:=y×x[j]+a[i];
печать (y) end
```

§ 5.4. ОПЕРАТОР БЛОК

Конечной целью разработки любого алгоритма является составление программы на языке конкретной ЦВМ, которую машина в состоянии выполнить. При переводе алгоритмической записи в код машины приходится решать некоторые специфические проблемы, связанные с размещением в памяти значений различных объектов, над которыми в процессе работы осуществляет преобразования ЦВМ. Так, например, для размещения значения простой переменной необходима одна ячейка памяти, а для хранения массива данных — группа ячеек. Чтобы решить, сколько ячеек памяти отвести для массива данных, необходимо знать характеристику его размерности. При организации обработки данных важно также заранее знать о свойствах

и характерных особенностях тех объектов, которые участвуют в операциях. По этой информации транслятор проверяет правильность использования величин в различных конструкциях программы на языке АЛГОЛ.

Отмеченные особенности учитываются в самой обобщенной структурной конструкции языка АЛГОЛ — *блоке*, который синтаксически представляет собой заключенную в операторные скобки последовательность описаний D и операторов S:

begin D; D; ... ; D; S; S; ... ; S **end**

Блок является безусловным оператором АЛГОЛа и внешне от составного оператора отличается лишь наличием описаний. Но блок играет особую роль в программе: вводит в употребление новые объекты; закрепляет за объектами систему обозначений — идентификаторов; содержит полную информацию, характеризующую каждый из вводимых объектов.

Всего в АЛГОЛе имеется четыре вида описаний, соответстующих тем классам объектов, которые обозначаются идентификаторами. Некоторым исключением из этого определения являются метки.

Описываемые объекты включают в себя простые переменные, массивы, переключатели и процедуры. Причем каждый вид описания относится к соответствующему классу объектов.

Описание простой переменной. Это описание в общем случае представляет собой указатель типа или **real (действительный)**, или **integer (целый)**, или **Boolean (логический)**, за которым следует список идентификаторов переменных, вводимых в употребление в данном блоке. Например:

begin real a, b, c; **integer** i, j, k;
Boolean p, q, r; ... **end**

Семантика последнего описания состоит в том, что в блоке вводится в употребление три новых объекта — три логических переменных, которым присвоены наименования p, q, r, причем в процессе вычислений эти переменные могут принимать только одно из логических значений: либо **true**, либо **false**.

Описание массива. Если при вычислениях в качестве определяемого объекта нужно ввести в употребление какой-нибудь массив, то для этой цели служит описание массива, характерной чертой которого является наличие символа **array**. В описании массива указывается тип значений, принимаемых его компонентами, и характеристика размерности массива в виде списка граничных пар по каждому измерению. Например,

integer array x [1 : 10], y [1 : 20, 0 : 9];

Тип значений компонентов массива, как и для простых переменных, указывается в виде указателей типа **real, integer, Boolean.**

Если перед символом **array** нет обозначения типа, считается, что описываемый массив представлен числами действительного типа.

Размерность массива определяется числом граничных пар в списке, а количество элементов массива по каждому измерению — соответствующей граничной парой.

В общем случае граничная пара состоит из двух арифметических выражений (частный случай — целые числа), разделенных символом «:». Левое выражение называют нижней границей, указывающей начальное (наименьшее) значение индекса. Правое выражение называют верхней границей, указывающей конечное (наибольшее) значение индекса.

При определении значений граничных пар необходимо иметь в виду следующие обстоятельства. Программа при входе в блок должна вычислить фактические значения пределов граничных пар. Поэтому переменные, встречающиеся в выражениях граничных пар, к моменту вхождения в блок обязательно должны иметь значения. После соответствующих вычислений полученные значения необходимо округлить до ближайшего целого. При этом левая граница пары не может быть больше правой границы пары, иначе соответстующий индекс не будет определен.

Следовательно, при решении вариантов задач с переменными значениями параметров массивов, их размерности в описаниях целесообразно задать максимальными значениями. Это приводит к тому, что при выбранном значении граничной пары по любому измерению в памяти машины будет зарезервировано необходимое количество ячеек для размещения исходных данных или результатов решения.

При использовании одной и той же программы для решения задачи при различных вариантах задания значений размерности массивов можно выделять стандартную процедуру «ввод» во внешний блок специально для осуществления ввода размерностей массивов. Дело в том, что идентификаторы, входящие в граничные пары, обязательно описывают во внешнем блоке, причем там же им присваивают значения. Так, например, при вычислениях значений полиномов n-й степени, по формуле Горнера, где n имеет разные значения, программа будет представлена двумя блоками, один из которых на правах безусловного оператора входит в другой:

```
begin integer n, m;
ввод (n, m);
begin real y; integer i, j;
array a [0 : n], x [1 : m];
ввод ( a, x);
for j:=1 step 1 until m do
begin y:=a [0];
for i:=1 step 1 until n do
y:=y×x [j]+a [i];
печать (y) end end
end
```

При такой структуре программы правомочно описание массивов с использованием в граничных парах переменных (в данном случае n и m, так как к моменту выполнения внутреннего блока значения этих переменных определены во внешнем блоке по оператору «*ввод*»).

Если несколько массивов имеют общую размерность и границы, то они могут быть описаны вместе, например описание

integer array $a1$, $q2$, bt [1 : 6, —2 : 12, j : n]

означает, что идентификаторы $a1$, $q2$, bt являются идентификаторами трехмерных массивов с границами для первого индекса от 1 до 6, для второго индекса — от —2 до 12 и для третьего индекса — от j до n.

Описание переключателя. Данное описание состоит из символа алфавита **switch**, идентификатора переключателя и списка меток операторов, которым необходимо передать управление при соответствующих значениях индекса указателя переключателя, например

switch pr: = $m1$, $m2$, $m3$, $m4$, $m5$

Описание переключателя располагается в начале блока и имеет силу только в рамках этого блока. В случае, когда переключательный список состоит не только из меток, но и из произвольных именующих выражений, указатель переключателя отсылает к одному из именующих выражений списка, которое либо непосредственно определяет метку, либо вновь отсылает к другому именующему выражению. Так, описание переключателя

switch p: = **if** B **then** $m1$ **else** $m8$, n, q

в первый раз определяет метку по условному именующему выражению, а второй и третий раз ставит в соответствие метки n и q.

Метки, используемые в блоке, хотя и обозначаются с помощью идентификаторов, но отдельного описания не имеют, так как метка считается описанной непосредственно в том месте, где она стоит перед оператором.

Принцип локализации объектов языка. Блок среди операторов АЛГОЛа является наиболее обобщенным оператором и содержит все необходимые данные и сведения для реализации записанного в нем алгоритма на ЦВМ. Поэтому считается, что программа на языке АЛГОЛ представляет собой блок. Внутри такой программы можно выделить в качестве самостоятельных операторов другие блоки, которые в свою очередь могут содержать среди своих операторов блоки низких рангов.

Пример структуры программы на языке АЛГОЛ показан на рис. 5.2. В приведенном примере внутри блока *1* имеются на правах самостоятельных операторов блоки *2* и *3*, которые в отношении друг к другу независимы, а по отношению к блоку *1* являются внутренними. Такая сложная иерархическая структура программы является универсальным средством для описания алгоритмов решения самых разнообразных задач и вносит соответствующее упорядочение в соподчинение тех объектов, которые используются в этой программе. Одним из основных принципов, положенных в основу организации

структуры программы на языке АЛГОЛ, является принцип локализации, состоящий в том, что объекты, описания которых помещены в начале какого-либо блока, имеют силу только внутри этого блока. По выходе из блока они теряют свой смысл и в дальнейшем не могут быть использованы. Другими словами, область действия объектов ограничена или локализована в том блоке, где дано их описание. Границами такой области и служат операторные скобки, охватывающие блок.

Так, в приведенном на рис. 5.2 примере переменные f, e, g, представленные значениями целого типа, имеют смысл и могут использоваться только внутри блока *2*, а переменные d, f, g, являющиеся переменными действительного типа, — только в блоке *3*. Поскольку блоки *2* и *3* независимые, то переменные этих блоков одновременно встретиться не могут. Поэтому, несмотря на то что переменные f и g имеют одинаковые наименования, используются они в разных блоках и имеют совершенно разный смысл.

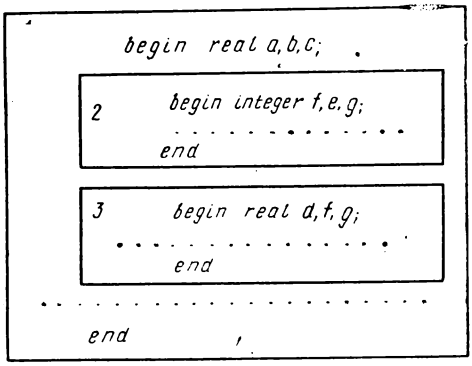


Рис. 5.2. Структура программы на языке АЛГОЛ-60

Переменные, в том числе и переменные, имеющие одинаковые наименования (идентификаторы), которые используют в независимых блоках, называют *локальными*.

Переменные, описанные во внешних блоках и используемые во внутренних блоках, называют *глобальными*. Например, переменные a, b, c (рис. 5.2) имеют один и тот же смысл и в блоке *2* и в блоке *3*, являясь глобальными переменными для этих блоков. Для блока *1* они локализованы.

Иногда глобальные переменные могут терять свои свойства глобальности. Это происходит тогда, когда во внешнем (охватывающем) и внутреннем блоках используются переменные, имеющие одинаковые наименования. В этом случае глобальная переменная считается недоступной для внутреннего блока и в нем игнорируется, поскольку существует локальная переменная, приобретающая свои права с момента вхождения программы в данный внутренний блок, например:

```
begin real a, p, q;
 . . . . . .
begin real b, p, q;
 . . . . . .
end
 . . . . . .
end
```

В этом примере величины первого блока p и q при входе во второй блок теряют свою глобальность и числовое значение, так как в этом блоке действуют свои, локализированные в нем, переменные p и q, относящиеся совершенно к другим объектам. Только после выхода из внутреннего блока продолжают действовать глобальные величины p и q, описанные во внешнем блоке.

Переменная a, описанная во внешнем блоке, не теряет свой смысл и значение во внутреннем блоке, сохраняя всюду глобальность.

Локальными переменными, как правило, описывают результаты промежуточных расчетов и вспомогательные величины, хранить которые в продолжении всего времени выполнения программы не имеет смысла, поскольку они нужны только в данном месте программы.

Принцип локализации позволяет пользоваться одними и теми же символами для обозначения разных величин в разных блоках. Это удобно при разработке программы решения одной задачи одновременно несколькими программистами, поскольку им не нужно согласовывать внутренние переменные каждой части задачи, а достаточно лишь каждую часть программы изложить в форме блока и в нем описать внутренние переменные. Тогда для получения программы нужно объединить отдельные блоки операторными скобками **begin** и **end** в один составной оператор или внешний блок с выделением описаний глобальных переменных.

Принцип локализации позволяет также экономнее использовать память вычислительной машины, так как после окончания вычислений, изложенных в одном блоке, место, отведенное для размещения его переменных, может быть занято другими величинами.

Каждый блок является оператором, и поэтому может быть помечен. Но метки не описываются и действуют только в том блоке, в котором встречаются, т. е. каждая метка локализована в наименьшем блоке, ее охватывающем. Это означает, что обращение к помеченному оператору извне блока недопустимо. Следовательно, войти в блок можно только через его начало. Это естественно, так как в противном случае были бы утеряны описания, стоящие в начале блока, и не были бы определены классы величин, ими определяемые и используемые в блоке. Пример, поясняющий эти положения, следующий:

```
m1 : begin integer a, b; ...
n1 : a := b := 0;  go to m1; ...
go to n1; ... go to k3; ...
go to l4;
k3 : begin real c, d; ...
l4 : c := d + 2; go to m1; ... go to n1; ...
go to k3; ... go to l4; ... end
end
```

Из внутреннего блока, помеченного меткой $k3$, можно передать управление на любую метку внутреннего и внешнего (охватывающего) блоков. А из внешнего блока, помеченного меткой $m1$, передача разрешена только на те метки, которые не входят во внутренний

блок. Запрещенной для внешнего блока в данном случае является метка *l4*, так как передача управления из внешнего блока в середину внутреннего блока невозможна.

После выхода из блока из-за локализации значения всех переменных, описанных в нем, теряются в силу того, что на их место в ячейках памяти машины располагаются переменные следующего блока. Даже если после выхода из блока управление будет передано снова на начало блока, то значения всех его переменных могут быть потеряны, поскольку при каждом входе в блок заново происходит распределение памяти ЦВМ под простые переменные и массивы.

Для того чтобы значения переменных (всех или некоторых) при повторном входе в блок сохранились такими же, как и при выходе из него, необходимо эти переменные описать как собственные с помощью символа **own**. Так, в структуре программы

```
begin real a, b;
own real c, d; own integer i, j, k;
own real array g [1 : 10];
  . . . . . .
end
```

переменные *a* и *b* теряют свои значения при повторном вхождении в блок, а переменные *c*, *d*, *i*, *j*, *k* и компоненты массива *g* сохраняют свои прежние значения.

При описании массивов как собственных отбрасывание символа **real** не допускается.

Следует отметить, что ни один идентификатор в блоке не может быть описан дважды как представитель разных объектов, например, ошибочной будет запись

```
begin integer i, j, k; real a, b, k; ... end,
```

так как в ней идентификатор *k* описан дважды.

Неправильно, когда для обозначения массива используют тот же идентификатор, что и для простой переменной, например,

```
begin real a, b; array a [1 : 10]; ... end.
```

Ошибка здесь состоит в том, что идентификатором *a* обозначена и действительная переменная и массив действительного типа.

§ 5.5. ПРОЦЕДУРЫ

При описании алгоритмов решения реальных задач часто возникает необходимость многократного выполнения какой-либо части вычислений, повторяющейся в разных местах вычислительного процесса. Для этого можно каждый раз записывать группу операторов, выполняющих повторяющуюся часть вычислений, но это нерационально. Удобнее же описать повторяющуюся часть один раз и по мере необходимости обращаться к ней, изменяя надлежащим образом входящие в нее исходные величины. Примером такой организации вычислительного процесса является употребление стандартных функций типа

sin x, ln x и т. п., которые выделены в стандартные процедуры — функции, более простые в использовании.

В более общем случае язык АЛГОЛ разрешает выделять отдельные группы операторов, описывающих определенный процесс, в процедуры, обращение к которым организуется с помощью оператора процедуры или указателя процедуры — функции. Поэтому процедура в АЛГОЛе состоит из описания процедуры и оператора процедуры или указателя процедуры — функции.

Описание процедуры наряду с другими описаниями переменных размещается в начале блока. Оператор процедуры или указатель процедуры — функции используется в том месте блока, где необходимо выполнить совокупность действий, отраженных в описании процедуры, над теми фактическими объектами, которые перечисляются в этом операторе процедуры.

Описание процедуры состоит из заголовка и тела процедуры. *Заголовок процедуры* включает в себя идентификатор процедуры, список формальных параметров, заключенный в круглые скобки, список значений и совокупность спецификаций. *Тело процедуры* представляет собой оператор (составной или блок), описывающий алгоритм стандартной части вычислений с использованием формальных параметров.

Формальными параметрами могут быть любые идентификаторы аргументов, в терминах которых описывается выделенный этап вычислений. Формальные параметры определяют входные объекты, необходимые для выполнения процедуры, и выходные величины, которые являются результатом работы процедуры.

Список значений определяет специфику замены формальных параметров фактическими при обращении к процедуре и включает в себя идентификаторы всех формальных параметров, которые до выполнения процедуры должны иметь значения. Таким выделением достигается разбиение всех параметров на параметры-значения и параметры-наименования, что в значительной степени облегчает работу транслятора и способствует рациональной организации процесса вычислений. В список значений могут быть включены формальные параметры, относящиеся к классам простых переменных, массивов или меток. Иногда список значений может отсутствовать.

Совокупность спецификаций вводится в заголовок процедуры для характеристики классов и типов формальных параметров. При этом для спецификации используют символы **real**, **integer**, **Boolean**, **array**, **switch**, **procedure** и **label**, после которых перечисляются идентификаторы объектов, относящихся к данному спецификатору. Например, заголовок процедуры в соответствии с указанными правилами можно записать как

procedure *variant* (x, y, z, p, r, m); **value** z, p, r;

real x, z; **array** p; **integer** r; **Boolean** y; **label** m;

Рассмотренные символы алфавита языка АЛГОЛ в описании процедуры сообщают лишь информацию об используемых параметрах, не вызывая никаких действий. В частности, в спецификации массивов

отсутствует указание о граничных парах, так как оно должно быть сделано при описании соответствующего фактического параметра по обычным правилам.

Тело процедуры является главной компонентой описания последней, и чаще всего представлено блоком, содержащим описания локальных величин тела процедуры. Операторы, входящие в состав тела процедуры, могут использовать формальные параметры, специфицированные в заголовке процедуры, локальные величины тела процедуры и глобальные величины, описанные во внешних блоках.

Обращение к описанной процедуре осуществляется по оператору процедуры, который синтаксически представляет собой идентификатор процедуры и заключенный в круглые скобки список фактических параметров. Например,

variant $(a, s \leqslant 0, b \times c, q, k+l, n)$.

Фактические параметры представляют совокупностью выражений, идентификаторов массивов и меток, переключателей и процедур, соответствующих конкретной задаче. Количество фактических параметров должно точно соответствовать количеству формальных параметров, а класс и тип каждого фактического параметра должен совпадать с классом и типом соответствующего (в порядке перечисления слева направо) формального параметра.

Фактические параметры процедуры в зависимости от специфики формальных параметров должны удовлетворять следующим условиям:

1) если формальный параметр специфицирован как **real** или **integer**, то в качестве фактического параметра может быть использовано любое арифметическое выражение, а если используется спецификатор **Boolean** — логическое выражение;

2) если формальный параметр специфицирован как массив, то в качестве фактического параметра может быть взят только идентификатор массива;

3) если формальный параметр специфицирован как метка, переключатель или процедура, то роль фактического параметра может выполнить только идентификатор соответствующего типа.

Итак, процедура в записи алгоритма представляет собой некоторый самостоятельный этап вычислений. Содержательно этот этап определяется описанием соответствующей процедуры, а оператор процедуры служит лишь средством обращения к ее выполнению. Выполнение оператора процедуры, в случае когда телом процедуры является оператор языка АЛГОЛ, эквивалентно выполнению следующих действий.

1. Всем формальным параметрам, перечисленным в списке значений заголовка описания процедуры, присваиваются значения соответствующих параметров, вычисляемые непосредственно перед входом в тело процедуры (вызов параметров — значений). Эти фактические параметры должны представляться объектами, способными принимать значения.

2. Все формальные параметры, не внесенные в список значений, заменяются в теле процедуры на соответствующие им фактические параметры (вызов параметров по наименованию). Этими фактическими параметрами могут быть только идентификаторы конкретных объектов; в противном случае подстановка наименований не определена.

3. Тело процедуры, модифицированное указанным способом, подставляется на место оператора процедуры и выполняется.

Пример. Поясним процесс выполнения оператора процедуры, если описание некоторой процедуры *prim* задано в виде

```
procedure prim (x, y, z, m); real x, y; integer m;
array z; begin real a, b, c; integer k;
a := 3.14 × x + 4.23 × y;
for k := 1 step 1 until m do
z [k] := a × k end.
```

При обращении к заданной процедуре с помощью оператора *prim* (*p*, *q*, *s*, 10) в тело описания процедуры вместо формальных параметров *x*, *y*, *z*, *m* подставляются соответствующие им фактические параметры *p*, *q*, *s*, 10 и тело процедуры, модифицированное таким образом, приобретает вид:

```
begin real a, b, c; integer k;
a := 3.14 × p + 4.23 × q;
for k := 1 step 1 until 10 do
s [k] := a × k end
```

В качестве фактических параметров в операторе процедуры можно применять идентификаторы переменных, массивов, процедур, стандартные функции, логические выражения, элементы массивов, метки, переключатели.

Если из тела процедуры необходимо выйти по какому-либо условию в программу, из которой было обращение к данной процедуре, то в качестве формального параметра используют метку. При выполнении указанного условия управление в процедуре передается на эту формальную метку, которая в свою очередь организует обращение в основную программу. Так, в программе

```
procedure Q (a, b, m); real a, b; label m;
begin real x, y;
x := a + b ↑ 3; if x > a then go to m
end
```

при обращении к данной процедуре оператором *Q* (*t*, *z*, *l*) в случае, когда значение *x* действительно будет больше, чем *a*, произойдет передача управления на метку *l* основной программы, поскольку в теле процедуры вместо *m* при использовании подставляется метка *l*.

В процедуре может быть предусмотрена выдача на печать любого текста, если в качестве формального параметра использовать строку, описанную с помощью символа **string** (строка). Например,

```
procedure prt (x, y, z); real x, y; string z;
begin real a, b; a := 5.4; b := a + ln (a ↑ 2);
печать (z) end
```

При обращении к этой процедуре по оператору

prt (*k*, *n*, *ведомость* |__| *ватрат*)

на печатающее устройство машины будет выдан текст: «*ВЕДОМОСТЬ ЗАТРАТ*».

Рассмотренные правила использования процедур относятся к процедурам общего вида, когда в результате выполнения процедуры определяется одно или множество значений разных объектов. В ряде случаев процедура нужна для того, чтобы в качестве промежуточного результата определить лишь одно некоторое значение какой-либо функции с тем, чтобы его можно было в дальнейшем использовать как значение аргумента, например в арифметической операции. Для этой цели используют другую разновидность процедуры, называемую *процедурой-функцией*. Ее отличие от процедуры общего вида состоит в том, что в заголовке процедуры перед символом **procedure** указывается тип получаемого значения функции (**integer**, **real** или **Boolean**), а в теле процедуры-функции среди составляющих его операторов должен быть, по крайней мере, один оператор присваивания, содержащий в левой части идентификатор данной процедуры.

Обращение к процедуре-функции осуществляется с помощью указателя функции, структура которого аналогична структуре оператора процедуры, но указатель функции должен быть элементом арифметического или логического выражения. Например, процедура-функция *kvk* (*x*, *y*) имеет описание

real procedure *kvk* (*x*, *y*); **value** *x*, *y*; **real** *x*, *y*;
begin real *a*; *a* := *x* ↑ 2 + *y* ↑ 2;
kvk := *sqrt* (*a*) **end**

При необходимости вычисления квадратного корня из суммы квадратов двух величин достаточно в каком-либо арифметическом выражении записать указатель данной процедуры-функции с соответствующими фактическими параметрами.

Допустим, если $z := \ln (b + c)/ (1 + kvk (b + c,\ m + n))$, то с учетом описания процедуры-функции *kvk* эта запись эквивалентна алгебраической записи $z = ln (b + c) / (1 + \sqrt{(b + c)^2 + (m + n)^2})$.

Используем оператор процедуры для формального описания часто встречающегося на практике процесса вычисления суммы различных компонент. Допустим, что при выполнении основной программы возможны многократные вычисления по формуле $s = \sum_{i=n}^{m} y_i$ при различных значениях фактических параметров. Рассматриваемая процедура имеет следующие формальные параметры: *s*, *l*, *n*, *m*, *y*.

Распишем процедуру вычисления суммы:

procedure *summa* (*s*, *l*, *n*, *m*, *y*); **value** *l*, *n*, *m*; **integer** *l*, *m*, *n*;
real *s*; **array** *y*; **begin integer** *i*; *s* := 0;
for *i* := *n* **step** *l* **until** *m* **do** *s* := *s* + *y* [*i*] **end**

Если в основной программе есть необходимость в вычислении нескольких подобных сумм, например $z = \sum_{j=0}^{k-l} x_j$; $p = \sum_{i=1}^{100} z_i$; $q =$

$= \sum_{t=l+2}^{a+b} v_t z_l$, то обращения к описанной ранее процедуре вычисления суммы можно выразить соответствующей последовательностью операторов процедуры:

```
summa (z, 1, 0, k—l, x);
summa (p, 1, 1, 100, z);
summa (q, 1, l+2, a+b, v[i]×r[i])
```

При исполнении процедур может возникнуть ситуация, когда необходимо обратиться к этой же процедуре. Такой вид процедуры, описанной на языке АЛГОЛ, называют *рекурсивной процедурой*. Пользуясь этой процедурой, иногда удается коротко и наглядно описать процесс решения задачи. Однако следует иметь в виду, что перевод рекурсивного описания на машинный язык очень сложен и доступен не каждой программе-транслятору машины.

Примером рекурсивной процедуры служит процедура вычисления факториала, описание которой в компактной форме можно представить так:

```
integer procedure factorial (n); value n; integer n;
if n > 1 then
factorial := n × factorial (n—1) else 1.
```

Если теперь в основной программе употребить указатель процедуры u: = *factorial* (k) с целым значением k, то будет вычислено значение $u = k!$ Рассмотрим, как произойдет вычисление данной функции. Если $k = 1$, то модифицированное тело процедуры присвоит переменной *factorial* значение 1. Если $k = 2$, то модифицированное тело

```
if k > 1 then
factorial := k × factorial (k—1) else 1
```

снова обратится к описанию функции *factorial*. В результате модифицированное тело процедуры примет вид:

```
if k—1 > 1 then
factorial := (k—1) × factorial (k—2) else 1
```

При выполнении этого условного оператора переменной *factorial* будет присвоено значение 1 ($k = 2$), которое в свою очередь будет подставлено в предыдущее модифицированное тело процедуры как значение *factorial* (k — 1). При больших значениях k рекурсивных обращений будет, конечно, больше.

В данном примере можно обойтись и без рекурсивного обращения к процедуре, если организовать вычисление факториала в цикле

```
integer procedure factorial (n),
value n; integer n;
begin integer i, g;
g := 1;
for i := 1 step 1 until n do g := g × i;
factorial := g
end
```

ГЛАВА 6

АЛГОРИТМИЧЕСКИЙ ЯЗЫК ФОРТРАН

Алгоритмический язык ФОРТРАН, разработанный в 1956 г. специалистами фирмы IBM (США), исторически явился первым языком программирования высокого уровня. Так как при его создании большое внимание уделялось вопросам простоты реализации (создания программ-трансляторов) на существующих ЦВМ, то поэтому этот язык содержит больше ограничений, чем, например, алгоритмический язык АЛГОЛ, и предоставляет программисту относительно меньше возможностей для компактного и изящного изображения сложных алгоритмов. Однако по этой же причине основные конструкции языка и сама структура ФОРТРАН-программы проще соответствующих структур АЛГОЛа, что облегчает изучение языка, составление и отладку программ, а также обусловливает, как правило, большую эффективность (с точки зрения скорости трансляции и выполнения) ФОРТРАН-программ.

Существенной особенностью ФОРТРАНа по сравнению с АЛГОЛом является наличие в нем операторов, управляющих вводом—выводом с представлением значения каждого данного в определенном формате, что позволяет осуществлять сложное редактирование данных, в частности при выдаче печатных документов (в АЛГОЛе это осуществляется посредством специальных процедур ввода—вывода).

ФОРТРАН, как и АЛГОЛ, являясь универсальным языком программирования, допускающим описание алгоритмов любых задач, в первую очередь предназначен для программирования математических и инженерно-технических задач, что отражено в самом названии языка (ФОРТРАН — FORmula TRANslator).

Работая с ФОРТРАНом, программист должен составить *исходную программу* (ФОРТРАН-программу), которая пробивается на перфокартах и вводится в ЦВМ. Записанная обычно на магнитной ленте *программа-транслятор* осуществляет перевод исходной программы в *рабочую программу* на языке данной ЦВМ с одновременным анализом синтаксических ошибок в исходной программе и выдачей соответствующей диагностической информации. Если трансляция прошла без обнаружения ошибок, полученная рабочая програма может автоматически выполняться ЦВМ.

Большинство современных отечественных ЦВМ средней и большой мощности оснащены трансляторами с ФОРТРАНа. В частности, имеются трансляторы для ЦВМ М-220, БЭСМ-6, машин серии ЕС ЭВМ и др.

С момента своего создания ФОРТРАН претерпел изменения. Имеется ряд версий языка, различающихся широтой допускаемых возможностей и ориентированных на разные классы ЦВМ. В настоящее время наиболее распространены две версии языка: Basic-ФОРТРАН, предназначенный для малых ЦВМ, представляющий собой минималь-

ное подмножество языка ФОРТРАН, и просто ФОРТРАН, близкий к версии ФОРТРАН-IV, содержащий целый ряд дополнительных возможностей описания алгоритмов по сравнению с Basic-ФОРТРАНом.

Рассмотрим изобразительные средства языка Basic-ФОРТРАН в сравнении с соответствующими конструкциями языка АЛГОЛ и отметим особенности других версий языка.

§ 6.1. ОСНОВНЫЕ ЭЛЕМЕНТЫ ЯЗЫКА ФОРТРАН

Алфавит ФОРТРАНа. Алфавит большинства версий языка включает в себя цифры 0, 1, ..., 9, заглавные латинские буквы, A, B, ..., и специальные символы, такие как + — */ ( ) = . ,

Для разделения символов, слов и выражений в ФОРТРАНе используют пробел. Он не имеет начертания и в программе представляется в виде пропуска, который в большинстве случаев (кроме специально оговоренных) не влияет на выполнение программы и поэтому может использоваться программистом достаточно произвольно для улучшения читаемости ФОРТРАН-программы.

В реализациях языка для отечественных ЦВМ в число основных символов входят также заглавные буквы русского алфавита.

ФОРТРАН-программа. Чтобы иметь представление о характере средств языка, используемых для записи алгоритмов, и структуре ФОРТРАН-программы, рассмотрим следующий пример: вычислить и напечатать действительные корни квадратного уравнения $Ax^2 + Bx + C = 0$ для 100 значений коэффициентов A, B и C, записанных на перфокартах (предполагается, что $A \neq 0$).

Действительные корни вычисляются по формуле

$$x_{1,2} = [-B \pm \sqrt{B^2 - 4AC}/(2A).$$

Если для какого-то набора коэффициентов A, B, C действительных корней нет ($B^2 - 4AC < 0$), то следует напечатать фразу «КОРНИ КОМПЛЕКСНЫЕ».

ФОРТРАН-программа решения этой задачи имеет следующий вид:

```
DIMENSION A(100), B(100), C(100)
1  FORMAT (3F 9.3)
2  FORMAT (3X, H17 КОРНИ КОМПЛЕКСНЫЕ)
3  FORMAT (2F 9.3)
READ 1, A, B, C
DO 4 J=1, 100
D=B(J)**2-4.*A(J)*C(J)
IF (D) 5, 6, 6
5  PRINT 2
GO TO 4
6  SD=SQRTF(D)
X1=(-B(J)+SD)/(2.*A(J))
X2=(-B(J)-SD)/(2.*A(J))
PRINT 3, X1, X2
```

```
4 CONTINUE
STOP
END
```

Каждая строка программы представляет собой *оператор* языка ФОРТРАН, служащий либо для указания действия, которое должна выполнить ЦВМ, либо для описания размерности массивов и форматов данных в операциях ввода—вывода.

Первый оператор, DIMENSION (РАЗМЕРНОСТЬ), указывает программе-транслятору, что величины A, B, C суть одномерные массивы (векторы), каждый из которых содержит по 100 компонент. По этой информации в оперативной памяти ЦВМ для каждой из величин A, B, C будет отведено по 100 ячеек.

Операторы FORMAT (ФОРМАТ), снабженные метками (номерами) 1, 2, 3, служат для описания представления исходных данных и выводимых результатов.

Первый из «исполняемых» операторов — оператор READ (ЧИТАТЬ). Он указывает ЦВМ, что необходимо считывать с перфокарт значения переменных A, B и C и записывать в отведенные для них ячейки памяти, причем чтение производится в соответствии с форматом, определенным оператором с меткой 1, т. е. оператором 1 FORMAT (3 F 9. 3). Символы 3 F 9. 3 говорят ЦВМ, что все три вводимых переменных имеют одинаковый формат F 9.3, означающий, что переменная занимает 9 десятичных разрядов, включая знак, причем после десятичной точки идут еще три цифры (например, переменная A в этом формате может иметь значение —2974.910). Поскольку оператор DIMENSION описал величины A, B и C как массивы, содержащие по 100 компонент, то ввод закончится, когда будут считаны все 100 значений A, B и C и записаны в память ЦВМ.

Следующий оператор DO 4 J = 1, 100 (DO — ВЫПОЛНИТЬ) — оператор цикла. Он заставляет ЦВМ повторять всю последовательность операторов, расположенных между оператором DO и оператором с меткой 4. При этом переменная J, выполняющая роль индекса, при первом прохождении цикла имеет значение 1, при втором — 2 и т. д., последний раз цикл выполняется при J = 100, после чего ЦВМ должна перейти к выполнению оператора, следующего за оператором с меткой 4.

Оператор D = B (J) ** 2 — 4. * A (J) * C (J) соответствует вычислению величины D по формуле $D = B_j^2 - 4A_jC_j$.

Оператор IF (D) 5, 6, 6 (IF — ЕСЛИ) передает управление оператору с меткой 5, если $D < 0$ и оператору с меткой 6, если $D \geqslant 0$. Случай $D < 0$ соответствует отсутствию действительных корней, и оператор с меткой 5 обеспечивает печать (PRINT — ПЕЧАТАТЬ) в соответствии с форматом, описанным оператором с меткой 2, сообщения «КОРНИ КОМПЛЕКСНЫЕ». После этого осуществляется переход (GO TO — ПЕРЕЙТИ К ...) к оператору с меткой 4.

Если $D \geqslant 0$, то подсчитывается значение $SD = \sqrt{D}$ и $X1 = (-B_j + SD)/2A_j$, $X2 = (-B_j - SD)/2A_j$. Затем значения X1

и X2 печатаются в соответствии с форматом, указанным оператором с меткой 3, т. е. в формате F 9.3, как и при вводе величин A, B и C.

Оператор с меткой 4, CONTINUE (ПРОДОЛЖАТЬ) используется для осуществления возврата к оператору, следующему за оператором DO. После прохождения цикла 100 раз выполняется оператор STOP (СТОП), который указывает ЦВМ, что программа закончена, и в зависимости от логики работы управляющей программы ЦВМ либо останавливается, либо автоматически переходит к решению следующей задачи. Оператор END (КОНЕЦ) служит для того, чтобы сообщать программе-транслятору, что в данной программе операторов больше нет (этот оператор всегда стоит последним в ФОРТРАН-программе).

Константы и переменные. В зависимости от наличия или отсутствия десятичной точки в записи константы в языке ФОРТРАН различают константы *вещественные* (*действительные*) *и целые*. Максимально допустимое количество цифр в записи константы зависит от типа ЦВМ, на которой реализован транслятор (как правило, это число не меньше 7). Вещественные и целые константы отличаются способом представления при хранении в памяти ЦВМ и способом выполнения над ними арифметических операций. Вещественные константы хранятся в памяти в виде числа с плавающей точкой, например: 972.13; —0.21; +1.; .14; 0.0. Целые константы имеют, например, такой вид: 98; —721; +1; +1234567; 0.

Вещественные константы могут быть представлены в полулогарифмической форме, т. е. с порядком. Порядок обозначается буквой E, например: $-0.0123E + 3 = -0.0123 \times 10^3 = -12.3$; $6.E02 = 6.0 \times 10^2 = 600$; $1.2E - 3 = 1.2 \times 10^{-3} = 0.0012$.

Таким образом, *константа* — величина, заданная в ФОРТРАН-программе в явном виде (своим значением).

Величину, к которой обращаются, используя соответствующее *имя*, а не значение, называют *переменной*. Имена, присваиваемые переменным, могут содержать от одного до шести символов (букв и цифр), причем первым символом должна быть буква. Если имя переменной начинается с одной из букв I, J, K, L, M, N, то переменная считается *целой*, в противном случае — *вещественной*. В зависимости от этого ее значение в ЦВМ хранится и обрабатывается по правилам арифметики с фиксированной или плавающей точкой.

Например переменные:

вещественные	целые	«неправильные»	ошибки
Q 48 A	N1	EPSILON	больше 6 символов
ALFA	KA	1X2Z	начинается не с буквы
SUM	LAMBDA	X-12	содержит специальный символ
X	M12B	R7.4	» » »
Y1	I	P 50	» » »

Из приведенного примера видно, что имеется некоторое отличие от языка АЛГОЛ: в АЛГОЛе шире алфавит — допускаются как заглавные, так и строчные буквы; тип переменной (действительный

или целый) определяется посредством явной спецификации типа **real** или **integer**. Такая возможность явного определения типа переменной допускается в ФОРТРАНе-IV, например: INTEGER X, P, A1BM, где переменные X, P, A1BM хотя и не начинаются с букв I, J, K, L, M, N, но рассматриваются как целые.

Запись REAL N, JOTA, MR позволяет рассматривать переменные N, JOTA, MR как вещественные.

В ФОРТРАНе, как и в АЛГОЛе, допускается, что в качестве значения величины, имеющей определенное имя (идентификатор), может выступать некоторое множество чисел. Тогда величину называют *массивом*, и отдельные элементы массива обозначают с помощью *переменных с индексами*. Индекс заключается в круглые скобки и ставится после имени массива (имена массивам даются по тем же правилам, что и простым переменным). Например, ссылка на восьмую компоненту одномерного вектора X в ФОРТРАН-программе осуществляется так: X (8). Это соответствует X_8 в обычной записи и X [8] в записи АЛГОЛа.

Индексы в ФОРТРАНе должны принимать положительные целые значения и могут задаваться в виде:

V
C
V ± C
C * V
C * V ± D,

где V — целая переменная без индексов; C, D — целые константы без знака. Например,

X (K), A1B(2), N (2 * I), Y (N1P+3), N1 (4 * K—1).

В различных реализациях языка допускается разное максимально допустимое число индексов (размерность массива): 2; 3; 7. Индексы, относящиеся к одному массиву, разделяются запятыми, например: A (I, J), GAMMA (I, 2, J + 1).

В ФОРТРАНе наложено бо́льшее количество ограничений на конструкцию индексов, чем в АЛГОЛе. Так, в индексное выражение могут входить только целые константы и переменные, и не допускается индексирование самого индекса. Например, в ФОРТРАНе не разрешена следующая запись:

X (K (L), I) = ... Вместо этого нужно использовать запись

J = K (L)
X (J, I) = ...

приводящую к тому же результату.

Операторы ФОРТРАНа. Основной синтаксической единицей языка ФОРТРАН является *оператор* (предложение). На бланке программы оператор занимает одну строку (в случае необходимости возможно продолжить его в следующих строках). Каждая строка обычно перфорируется на отдельную перфокарту. Для ссылки в программе на операторы им могут присваиваться *метки* (номера) — целые

числа без знака. В разных реализациях языка максимальное число меток различно. Метки могут стоять в программе в любом порядке, но никакие два оператора не должны иметь одинаковых меток. Метки рекомендуется присваивать только тем операторам, на которые есть ссылки в программе. Операторы ФОРТРАНа можно разделить на операторы: арифметический; передачи управления; ввода—вывода; спецификации (описатели), содержащие информацию для программы-транслятора относительно объема памяти, необходимой для хранения массивов, и формата данных при выполнении операций ввода—вывода; подпрограммы.

§ 6.2. АРИФМЕТИЧЕСКИЙ ОПЕРАТОР

Арифметические выражения. Под *арифметическим выражением* в ФОРТРАНе понимают любую последовательность констант, переменных и функций, разделенных знаками операций и круглыми скобками таким образом, чтобы получилось имеющее смысл математическое выражение. В качестве знаков операций в ФОРТРАНе используют: + сложение; — вычитание; * умножение; / деление; ** возведение в степень.

Например:

выражение	ФОРТРАН-выражение	АЛГОЛ-выражение
$\frac{X+Y}{2T^2}$	(X+Y)/(2.*T**2)	(X+Y)/(2 × T ↑ 2)
$\frac{x_i}{(x_i+a_{ik})^n}$	X(I)/(X(I)+A(I, K))**N	x [i]/(x [i]+a[i, k]) ↑ n

При записи выражений необходимо учитывать следующее:

1. Величины, входящие в выражение, должны быть одного типа (либо все целые, либо все вещественные), кроме показателей степени, которые могут быть целыми и для вещественных выражений. В некоторых реализациях ФОРТРАНа допускается использование переменных разного типа в одном выражении, в этом случае программа-транслятор автоматически преобразует типы переменных по правилу: если хотя бы одна компонента выражения вещественная, результат также вещественный.

2. Если целое число возводится в целую степень, результатом будет целое число. Во всех других случаях возведение в степень дает действительный результат.

3. Порядок выполнения операций определяется круглыми скобками, а при их отсутствии старшинством операций, убывающим в таком порядке: а) вычисление функций; б) возведение в степень; в) умножение и деление; г) сложение и вычитание.

Операции одного ранга выполняют по порядку слева направо. В «сомнительных» случаях (например, когда подряд идут два возведения в степень) лучше поставить лишние скобки.

Стандартные функции. В ФОРТРАНе предусмотрена возможность включения в арифметическое выражение различных функций, вычисляемых автоматически по стандартным подпрограммам. Эти функции называют *стандартными* или *библиотечными*. Правила записи таких функций несколько различаются в Basic-ФОРТРАНе и ФОРТРАНе-IV. В список этих функций входят следующие функции:

Basic-ФОРТРАН	ФОРТРАН-IV	АЛГОЛ	Математический смысл
EXPF (X)	EXP (X)	exp (x)	e^x
SQRTF (X)	SQRT (X)	sqrt (x)	$\sqrt{x}$
LOGF (X)	ALOG (X)	ln (x)	$\ln x$
SINF (X)	SIN (X)	sin (x)	$\sin x$
COSF (X)	COS (X)	cos (x)	$\cos x$
ATANF (X)	ATAN (X)	arctan (x)	arctg x
ABSF (X)	ABS (X)	abs (x)	$\lvert x \rvert$

В ФОРТРАНе-IV, кроме того, в состав библиотечных функций входят модификации почти всех перечисленных функций для аргументов, заданных с удвоенной точностью, и для комплексных значений аргументов. Например, DLOG (X) — функция ln x для x, заданного с удвоенной точностью, CLOG (X) — комплексное значение ln x, где x — комплексное число.

При решении задач на ФОРТРАНе предварительно следует ознакомиться с составом библиотечных функций, допускаемых конкретной реализацией языка. Общее требование при записи стандартных функций состоит в том, что аргументы должны заключаться в круглые скобки. В качестве аргумента стандартной функции может быть не только переменная или константа, но и арифметическое выражение, в которое могут входить другие функции или эта же функция. Например,

```
SQRTF (LOGF (1. +SQRTF (X**2+ABSF (Y)))).
```

Эта запись соответствует формуле

$$\sqrt{\ln(1+\sqrt{x^2+|y|})}.$$

Арифметический оператор. Арифметический оператор языка ФОРТРАН соответствует оператору присваивания в АЛГОЛе и записывается в виде a = b, где a — имя переменной (простой или с индексом); b — арифметическое выражение. Знак равенства имеет здесь смысл «следует заменить на ...».

Примеры

`Z=X ** 2+Y ** (1/3)`	$z=x^2+\sqrt[3]{y}$
`A (I, K)=B (I) * AL+C (I, K)`	$a_{ik}=b_i\alpha+c_{ik}$
`Y=2. * X * EXPF (X)`	$y=2xe^x$
`K=K+1`	Значение K увеличивается на 1.
`B=C`	Значение C присваивается переменной B.

После того как значение выражения в правой части арифметического оператора вычислено, оно запоминается в качестве нового значения переменной, указанной в левой части, если типы переменных в обеих частях совпадают. Если же типы переменных в обеих частях различны, то вычисления производятся с использованием типа переменной правой части, а затем результат преобразуется в значение типа переменной левой части. Например, в результате выполнения двух операторов

```
X=.1
M=8.*X+4.1
```

значение M окажется равным 4 (результат вычисления правой части, равный 4.9, преобразуется в значение целого типа, поскольку в левой части стоит переменная M — целого типа).

§ 6.3. ОПЕРАТОРЫ ПЕРЕДАЧИ УПРАВЛЕНИЯ

В случае отсутствия каких-либо операторов (команд) передачи управления ЦВМ будет выполнять операторы ФОРТРАНа последовательно в том порядке, в котором они записаны на бланке программы и пробиты на перфокартах. Эту последовательность выполнения операторов можно изменить, используя операторы управления.

Оператор GO TO (ПЕРЕЙТИ К ...). Этот оператор имеет вид

```
GO TO n.
```

где n — метка того оператора, который должен выполняться следующим. Например,

```
R=1.0
S=0.0
12 S=S+SINF (R)/R**4
R=R+1.0
GO TO 12
```

При выполнении этой части программы ЦВМ вначале последовательно заносит в ячейку памяти, отведенную для переменной R, единицу, а в ячейку, отведенную для переменной S, — нуль, затем вычисляет значение выражения $0 + \sin 1/1^4$ и результат заносит в ячейку, отведенную для переменной S (на место прежнего значения, равного нулю), потом увеличивает значение R на единицу и полученную двойку заносит в ячейку, отведенную для переменной R. Следующий оператор заставляет ЦВМ вернуться к оператору с меткой 12 и вычислить значение $\sin 1/1^4 + \sin 2/2^4$, записывая этот результат в ячейку для S. Таким образом, в ходе выполнения этой программы в ячейке для переменной S накапливается значение суммы $S = \sum_{R=1}^{\infty} \sin R/R^4$.

Чтобы заставить машину перестать накапливать сумму («выйти из цикла»), ограничившись, например, тридцатью слагаемыми, можно использовать оператор условной передачи управления IF.

Оператор IF (ЕСЛИ). Этот оператор записывается в форме

IF (a) n_1, n_2, n_3,

где a — арифметическое выражение; n_1, n_2, n_3 — метки.

Если $a < 0$, после оператора IF следующим будет выполняться оператор с меткой n_1, если $a = 0$, следующим выполняется оператор с меткой n_2, если $a > 0$, то выполняется оператор с меткой n_3.

С учетом сказанного, фрагмент программы для подсчета суммы $S = \sum_{R=1}^{30} \sin R/R^4$, на языке ФОРТРАН можно записать в виде:

```
R=1.0
S=0.0
12 S=S+SINF (R)/R ** 4
R=R+1.0
IF (R—30.) 12, 12, 5
5 ПРОДОЛЖЕНИЕ ПРОГРАММЫ
```

В качестве оператора с меткой 5 может быть, например, оператор печати значения переменной S.

Пример. Написать участок ФОРТРАН-программы для вычисления функции

$$y=\begin{cases} \dfrac{1}{\sqrt{d}}\operatorname{arctg}\dfrac{b+c}{\sqrt{d}}, & \text{если } d>0; \\ -\dfrac{1}{b+c}, & \text{если } d=0; \\ \dfrac{1}{2\sqrt{-d}}\ln\dfrac{\sqrt{-d}-b-a}{\sqrt{-d}+b+c}, & \text{если } d<0, \end{cases}$$

где $d = ac - b^2$.

```
D = A * C — B ** 2
IF (D) 1, 2, 3
1 S = SQRTF (—D)
  Y = 1./(2. * S) * LOGF ((S — B — A)/(S + B + C))
  GO TO 4
2 Y = — 1./(B + C)
  GO TO 4
3 S = SQRTF (D)
  Y = 1./S * ATANF ((B + C)/S)
4 ПРОДОЛЖЕНИЕ ПРОГРАММЫ
```

Примечание. Чтобы написанный фрагмент программы мог выполняться, необходимо значения A, B и C определить в программе заранее (например, при помощи оператора ввода или арифметических операторов).

Пример. Член ряда с номером n определяется выражением

$$a_n = \frac{x^{2n}\sin(x^n)}{n^2}.$$

Написать группу операторов, вычисляющую сумму членов ряда от первого члена до (включительно) члена с наименьшим номером, не превосходящего по абсолютной величине $1 \cdot 10^{-6}$.

```
      WN = 1. 0
      S = 0.0
25    A = X** (2. * WN) * SINF (X ** WN)/WN * * 2
      S = S + A
      WN = WN + 1.
      IF (ABSF (A) — 1. E — 6) 10, 10, 25
10    ПРОДОЛЖЕНИЕ ПРОГРАММЫ
```

Примечание. Использована переменная с именем WN вместо переменной N, поскольку в правой части оператора, вычисляющего значение A, все переменные должны быть одного (вещественного) типа. Это делать было бы не обязательно, если бы переменная N входила только в показатель степени (показатели могут быть целыми даже в вещественных выражениях.) По этой же причине используется запись WN = 1.0 вместо WN = 1; S = 0. 0 вместо S = 0 и WN = WN + 1 вместо WN = WN + 1.

Операторы END (КОНЕЦ) и STOP (СТОП). Оператор END является самым последним в ФОРТРАН-программе. Он сообщает транслятору, что в программе операторов больше нет. Оператор STOP указывает на окончание выполнения программы. Используется он для возврата в управляющую (служебную) программу, которая осуществляет переход к следующей программе.

Оператор DO (ВЫПОЛНИТЬ). Этот оператор называют *оператором цикла*. Принцип его действия удобно сначала проиллюстрировать на примере. Предположим, что необходимо вычислить сумму нечетных членов натурального ряда от 1 до 999. ФОРТРАН-программа, решающая эту задачу, выглядит так:

```
      KS=0
      DO 5 K=1, 999, 2
5     KS=KS+K
```

Записанный в программе оператор DO можно расшифровать следующим образом: «выполнять многократно все операторы, следующие за оператором DO, вплоть до оператора с меткой 5, причем при первом выполнении переменная K должна быть равной 1, при каждом следующем повторении она увеличивается на 2, последнее повторение происходит, когда K = 999».

```
DO 5 K=1, 999, 2 <=>
```

«Выполнять цикл, кончающийся оператором 5, меняя при этом K от 1 до 999 через 2».

Общий вид оператора DO следующий:

```
DO n i=m1, m2, m3
   или
. DO n i=m1, m2,
```

где n — метка; i — целая переменная без индексов; m_1, m_2, m_3 — целые константы без знака или целые переменные без индексов, принимающие положительные значения (m_1 представляет собой начальное

значение переменной i, m_2 — ее конечное значение, m_3 — шаг изменения (приращение) переменной i. Если величина m_3 не задана, она принимается равной 1. Если $m_1 \geqslant m_2$, то операторы выполняются только один раз).

Следует помнить, что параметр i, изменение которого указывается оператором DO, может быть только переменной целого типа, причем все ее значения и приращение строго положительны. В АЛГОЛе таких ограничений при использовании оператора цикла **for ... do** ... нет. Для иллюстрации правил записи оператора DO приведем примеры операторов, записанных с ошибкой.

```
DO 14 B = 1, 25, 3, (параметр цикла вещественного типа);
DO 29 J = 0, 16, JS (начальное значение нуль);
DO 1 K = 21, 3, —1 (приращение отрицательное);
DO 7, KT = 2, 15, 3 (лишняя запятая);
DO 2 K = 1, N, H (приращение веществ типа);
DO N1 L = L1, L2 (нечисловая метка);
DO 197 M = 2N 1 (не хватает запятых).
```

Операторы, составляющие цикл, должны подчиняться следующим правилам.

1. Внутри цикла не должно быть операторов, изменяющих значения параметров цикла (i, m_1, m_2, m_3). Параметр i определяется только оператором DO, а значения m_1, m_2, m_3 (если это не числа, а переменные) должны быть определены до входа в цикл. В ФОРТРАНе не допускается, например, такая конструкция цикла

```
NH = 2
DO 15 N = 2, 25, NH
. . . . . .
NH = NH + 1
15 . . . . . .
```

2. Последним оператором цикла не может быть оператор передачи управления (например, GO TO или IF). При необходимости обойти это ограничение можно воспользоваться «фиктивным» оператором CONTINUE (ПРОДОЛЖАТЬ), который используется всегда, когда нужен пронумерованный оператор, не выполняющий на самом деле никаких фактических вычислений.

Пример. Имеется массив из 26 значений y_j ($j = 1, 2, \ldots 26$) и переменная X. Требуется занести в ячейку Z первое из y_j, большее, чем X. Если такого y_j не найдется, то в ячейку Z следует поместить X.

```
  DO 5 J = 1,25, 1
  IF (X — Y (J)) 4, 5, 5
5 CONTINUE
  Z = X
  GO TO 6
4 Z = Y (J)
6 STOP
```

В данном случае в цикл входит только один фактически выполняемый оператор (IF), оператор CONTINUE состоит в том, чтобы выполнить роль последнего оператора цикла, поскольку цикл не может заканчиваться оператором IF.

3. Любая последовательность передач управления внутри цикла должна в конечном счете приводить к последнему оператору цикла, а не к самому оператору DO.

Пример. Написать участок программы, обеспечивающей получение наименьшего из 60 значений A_i, i = 1, 2, ..., 60, которое следует присвоить переменной AMIN.

```
      AMIN = A (1)
      DO  11 I = 2,50
      IF (AMIN — A (I)) 11, 11, 3
 3    AMIN = A (I)
11    CONTINUE
```

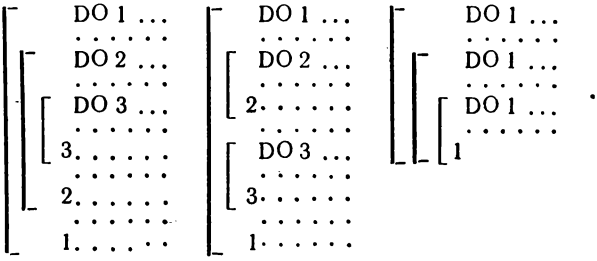
Ошибочной будет запись вида

```
      AMIN = A (1)
15  DO 11 I = 2, 50
      IF (AMIN — A (I)) 15, 15,3
3   AMIN = A (I)
11  CONTINUE
```

4. Вход в цикл DO должен производиться через оператор DO. Не разрешается передача управления, например при помощи операторов GO TO или IF, лежащих вне цикла, оператору, расположенному внутри цикла DO. Исключением из этого правила будет следующее: возможен выход из цикла DO, например для выполнения стандартной подпрограммы (см. ниже), с последующим возвратом внутрь цикла при условии, что отсутствуют изменения значений любого из параметров цикла.

5. Разрешается использование циклов в цикле. В этом случае область действия внутреннего цикла должна полностью находиться в области действия внешнего цикла. Такие циклы называются *вложенными*.

Допустима следующая структура записи циклов.



Недопустима следующая структура записи циклов:

```
DO 1 ..
. . . .
DO 2 ...
. . . . .
1 ...
. . . .
2 ...
```

Пример. Написать программу транспонирования прямоугольной матрицы размера 20 × 50.

Транспонирование заключается в том, чтобы получить матрицу, строки которой являются столбцами исходной матрицы, а столбцы — строками исходной. Таким образом, элемент A_{kl} исходной матрицы A, стоящей на пересечении k-й строки и i-го столбца, в транспонированной матрице AT должен находиться на пересечении i-й строки и k-го столбца: $AT_{ik} = A_{ki}$. Участок ФОРТРАН-программы, осуществляющий транспонирование матрицы A, имеет вид:

```
DO 1 I = 1,50
DO 1 K = 1,20
1 AT (I, K) = A (K, I)
```

Эта программа производит запись элементов массива A в массив AT в таком порядке

```
AT (1,1) = A (1,1)
AT (1,2) = A (2,1)
AT (1,3) = A (3,1)
. . . . . .
AT (1,20) = A (20,1)
AT (2,1) = A (1,2)
. . . . . .
AT (50, 20) = A (20, 50)
```

Оператор DIMENSION (РАЗМЕРНОСТЬ). В ряде предыдущих примеров встречались переменные с индексами, являющиеся элементами одномерного (векторы) или двумерного (матрицы) массивов. Для каждой переменной с индексами в начале ФОРТРАН-программы должен быть указан максимальный размер массива, т. е. максимальные значения, которые могут принимать индексы. Это нужно для того, чтобы транслятор мог зарезервировать соответствующий объем оперативной памяти для этих переменных. Обычно информация о размерности массивов содержится в операторе DIMENSION. Например,

```
DIMENSION A (20, 50), AT (50, 20), Y (26).
```

Размеры массивов, определяемые оператором DIMENSION, могут быть только числами (целыми без знака), а не переменными.

Оператор DIMENSION является *невыполняемым оператором* в том смысле, что не порождает команд в рабочей программе, поэтому иногда его называют не оператором, а описателем. Оператор DIMENSION соответствует описателю **array** в языке АЛГОЛ.

В языке ФОРТРАН-IV, имеющем предложения явной спецификации типов переменных (REAL, INTEGER и т. д.), допускается включение информации о размерности массивов в эти предложения, например

```
REAL A (20, 30), GAM (150)
INTEGER R, W, Y4 (15, 6), Q (20)
```

§ 6.4 ОПЕРАТОРЫ ВВОДА — ВЫВОДА

Для выполнения операции ввода или вывода данных ЦВМ должна иметь сведения о том, какие данные надо вводить или выводить и каков формат этих данных (вещественные числа или целые, цифровые данные или буквенные, сколько разрядов занимают, в каких колонках пер-

фокарты содержатся, и т. д.). В языке ФОРТРАН для каждой операции ввода или вывода эти сведения содержатся в двух операторах: оператор READ (ЧИТАТЬ) или PRINT (ПЕЧАТАТЬ) определяют списки переменных, а оператор FORMAT (ФОРМАТ) специфицирует формат данных и их расположение в записи ввода — вывода.

Оператор READ. Этот оператор используют для считывания данных с перфокарт в оперативное ЗУ ЦВМ. Общая форма его записи имеет вид

```
READ m, список,
```

где m — метка соответствующего оператора FORMAT, список — последовательность имен переменных (простых или с индексами), разделенных запятыми, которые должны быть считаны с перфокарт. Например,

```
READ 25, X, Y, A (5),
```

где 25 — метка оператора FORMAT, содержащего информацию о том, каким образом отперфорированы на карте три числа, значение которых ЦВМ должна вводить в ячейки, отведенные для переменных X, Y, и в пятую ячейку массива A.

Оператор PRINT. Этот оператор имеет вид

```
PRINT m, список.
```

Отличается он от оператора READ только тем, что определяет вывод, а не ввод данных.

В ФОРТРАНе-IV в операторах ввода—вывода дополнительно указывается номер устройства ввода — вывода, используемого в данной операции. Там эти операторы имеют структуру:

```
READ (n, m), список,
WRITE (n, m), список,
```

где n — целое без знака либо переменная целого типа, имеющая положительное значение, определяющая используемое устройство ввода — вывода; WRITE—«ПИСАТЬ».

Оператор FORMAT. Этот оператор сообщает ЦВМ информацию о о способе представления вводимых или выводимых данных (целые, алфавитно-цифровые или вещественные, которые могут быть записаны в обычной форме—с десятичной точкой, или в форме с порядком), а также об сбщем количестве символов и количестве цифр после десятичной точки. Разные способы представления данных соответствуют разным форматам данных.

Формат типа I. Этот формат используют, когда имеют дело с целыми числами. Его общая форма

```
Iw,
```

где w — длина поля перфокарты, занятого числом (количество символов в изображении числа на перфокарте — в эти символы входят знак и цифры), например, операторы

```
12 FORMAT (I 4)
   READ 12, NB
```

сообщают ЦВМ, что следует вводить с перфокарты значение целой переменной NB, занимающей первые четыре колонки на карте (знак и три десятичные цифры), а операторы

```
5 FORMAT (I 4, I 6, I 3)
  READ 5, K, L, M
```

определяют ввод целых переменных K, L и M, занимающих на перфокарте, соответственно, колонки с 1 по 4, с 5 по 10, с 11 по 13.

Если поле, определяемое форматом типа I, больше поля, фактически занятого вводимым числом, то неиспользованная часть поля заполняется нулями.

Операторы FORMAT могут располагаться в любом месте ФОРТРАН-программы. Более того, несколько разных операторов READ или PRINT могут ссылаться на один и тот же оператор FORMAT.

Формат типа F. Это формат действительных чисел с фиксированной точкой. Его общая форма

```
Fw. d,
```

где w — общая длина поля; d — количество десятичных цифр после точки. Внутри поля знак и десятичная точка занимают по одной позиции. Поэтому w должно быть больше d, по крайней мере, на два. Например:

Формат	Значение	Вводится / выводится
F 9.4	—125.982	—125.9820
F 4.2	1.3	1.30
F 8.6	546.37	..... (ошибка)

При пользовании форматом типа F программист обязан знать максимальную величину чисел, которые встретятся при вводе — выводе.

Формат типа E. Этот формат используют для перфорации значений вещественных переменных, записанных с порядком. Его общая форма

```
Ew.d,
```

где w — определяет длину поля; d — определяет количество дробных десятичных разрядов (в мантиссе). Например, число —26.4972 в формате E 12.6 будет выведено в виде —.264972 E + 02.

Пропуск поля. Если необходимо при вводе пропустить w последовательных колонок перфокарты или при выводе оставить w пробелов, используют формат типа X. Его вид

```
wX.
```

Например, операторы

```
1 FORMAT (5X, I4, F 9 . 4)
  READ 1, K, Y
```

заставят ЦВМ считывать с перфокарты целое число K, занимающее колонки с 6-й по 9-ю вещественное число Y, занимающее на карте колонки с 10 по 18.

Можно пропускать не только позиции на одной строке или карте, но строки или карты целиком. В этом случае в соответствующем месте оператора FORMAT должна ставиться наклонная черта «/».

Если, например, переменные K и M записаны на одной перфокарте и занимают соответственно колонки с 10-й по 18-ю и с 25-й по 30-ю, а переменная M1X — на следующей перфокарте и занимает колонки с 1-й по 7-ю (переменные K, M, M1X целого типа), то ввод их в память ЦВМ можно осуществить с помощью операторов

```
4 FORMAT (9X, I 9, 6X, I6, /, I 7)
  READ 4, K, M, M 1 X.
```

Ф о р м а т т и п а H. С помощью этого формата можно выдавать на печать буквенно-цифровую информацию, т. е. заголовки, примечания и др. Общий вид этого формата wH символы, где w — число печатаемых буквенно-цифровых символов, перечисленных непосредственно после буквы H, включая пробелы. Например, в результате действия операторов

```
  PRINT 1, X1, X 2
1 FORMAT (6 H X1 = , F 9 . 3, 6H X2 = , F 9 . 3)
```

может быть напечатана такая строка: X1 = — 2345.678 X2 = = + 9876.543, а выполнение операторов

```
  2 FORMAT (25 H ДЕЙСТВИТЕЛЬНЫХ КОРНЕЙ НЕТ)
    PRINT 2
```

приведет к печати фразы «ДЕЙСТВИТЕЛЬНЫХ КОРНЕЙ НЕТ».

Ввод массивов. Если требуется вводить не отдельные переменные, а целые массивы, то это можно реализовать с помощью оператора цикла и рассмотренной формы оператора READ. Например, для ввода массива из 50 значений $X_к$ можно использовать операторы

```
 -DIMENSION X(50)
1 FORMAT (F12 . 6)
  DO 2 K = 1,50
  2 READ 1, X(K)
```

Однако язык ФОРТРАН допускает более компактную форму ввода массивов. Тот же результат получится, если записать операторы

```
DIMENSION X(50)
1 FORMAT (F 12. 6)
READ 1, (X (K), K = 1,50)
```

или операторы

```
  DIMENSION X(50)
1 FORMAT (F12 . 6)
  READ 1, X
```

Таким образом, для ввода массивов можно в операторе READ указать только название массива (размерность массива следует указывать с помощью оператора DIMENSION). Например, при вводе двухмерного массива можно использовать операторы

```
   DIMENSION A(20, 30)
12 FORMAT (F 8 . 4)
   READ 12, A
```

Пример. На перфокартах имеется массив из 260 чисел Z_i. Каждое число записано на отдельной карте в колонках с 6-й по 17-ю, 13-я колонка занята десятичной точкой.

Требуется написать ФОРТРАН-программу для расчета среднего значения и среднеквадратичного отклонения, используя формулы:

$$Z_{ср} = \frac{1}{260} \sum_{i=1}^{260} Z_i,$$

$$\sigma = \sqrt{\frac{1}{259} \sum_{i=1}^{260} Z_i^2 - \frac{260}{259} Z_{ср}^2}.$$

ФОРТРАН-программа имеет вид:

```
  DIMENSION Z(260)
1 FORMAT (5X, F 12 . 4)
2 FORMAT (F 12 . 4, F 12 . 4)
  READ 1, Z
  S = 0.
  S1 = 0.
  DO 3 I = 1, 260
  S = S + Z (I)
3 S1 = S1 + Z (I) * * 2
  ZSR = S/260.
  SIGMA = SQRTF (S1/259. — 260./259. * ZSR **2)
  PRINT 2, ZSR, SIGMA
  STOP
  END
```

Пример. Решить относительно X уравнение $e^{-0,3X} = 0,7\ X$.
Корень уравнения определить с точностью $1 \cdot 10^{-6}$.

Представив уравнение в виде $F(X) = e^{-0,3X} - 0,7\ X = 0$, будем решать его итерационным методом Ньютона, используя формулу $X_{n+1} = X_n - F(X_n)/F'(X_n)$ и взяв в качестве начального приближения значение $X_0 = 0$. Условием окончания итерационного процесса будет $|X_{n+1} - X_n| < 1 \cdot 10^{-6}$. ЦВМ на каждом этапе вычислений имеет дело только с двумя последовательными приближениями: X_n и X_{n+1}, поэтому нет необходимости резервировать массив ячеек для значений X_n, достаточно лишь иметь две ячейки, которые обозначим XN и XN1.

```
1 FORMAT (F 12 . 7)
  XN = 0.
2 EX = EXPF (—0.3 * XN)
  XN1 = XN — (EX — 0.7 * XN) / (—0.3 * EX — 0.7)
  IF (ABSF (XN1 — XN) — 1 . E — 6) 4, 3, 3
3 XN = XN1
  GO TO 2
4 PRINT 1, XN1
  STOP
  END
```

Пример. Решить относительно X с точностью $1 \cdot 10^{-6}$ уравнение $e^{-\rho X} = (1 - \rho)X$ для разных значений параметра ρ, беря $0 \leqslant \rho < 1$ через 0·01 (решение для каждого фиксированного значения ρ рассмотрено в предыдущем примере)

```
1 FORMAT (F 12.7)
  RO = 0.0
5 XN = 0 . 0
2 EX = EXPF (—RO * XN)
```

```
  XN1 = XN — (EX — (1 . —RO) * XN)/(—RO * EX — 1. + RO)
  IF (ABSF (XN1 — XN) — 1 . E — 6) 4, 3,3.
3 XN = XN1
  GO TO 2
4 PRINT 1, XN1
  RO = RO + 0.01
  IF (RO — 1.) 5, 6, 6
6 STOP
  END
```

§ 6.5. ФУНКЦИИ И ПОДПРОГРАММЫ

Оператор-функция. Предположим, что в задаче, программируемой для ЦВМ на языке ФОРТРАН, несколько раз надо вычислить функцию th z по формуле $\text{th}\, z = (1 - e^{-2z})/(1 + e^{-2z})$ для ряда не связанных друг с другом значений аргумента z, например:

$$B_5 = \text{th}\left(\sin(2x+1)\right),$$
$$y = \text{th}\,(A_{16} - R_4 \cos x),$$
$$w = \sqrt[3]{\text{th}\, R_3 + 3{,}14\sqrt{A_5}}.$$

В составе стандартных функций языка Basic-ФОРТРАН нет функции THF (Z). Но эту функцию можно однажды определить в составляемой программе и затем ссылаться на нее по имени, как на стандартную функцию, не расписывая каждый раз формулу для вычисления th$z$:

```
THF (Z) = (1. — EXPF (—2. * Z))/(1 . + EXPF (—2 . * Z)).
```

В соответствующих местах программы будут стоять операторы

```
Z = SINF (2. * X + 1.)
B (5) = THF (Z)
Y = THF (A (16) — R (4) * COS (X))
W = (THF (R (3)) + 3. 14 * SQRTF (A (5))) ** (1. /3.)
```

Оператор, реализующий формулу для вычисления функции, называют *оператором-функцией*. В левой части этого оператора стоит имя функции, за которым в скобках указываются аргументы функции (в данном случае один аргумент). Оператор-функция не нумеруется и стоит в начале программы, раньше любых выполняемых операторов. Аргументами оператора-функции могут быть только простые переменные. Имя функции должно оканчиваться на букву F (этого требования нет в языке ФОРТРАН-IV). В правой части оператора-функции стоит выражение, посредством которого вычисляется определяемая функция. Существенным ограничением в использовании оператора-функции является то, что функция задается только одним выражением.

Подпрограмма-функция. Если функцию невозможно определить одним выражением, то можно воспользоваться ее заданием в виде *подпрограммы-функции*.

Предположим, что в программе несколько раз приходится выполнять вычисления с минимальным из двух чисел для разных пар чи-

сел. Тогда можно ввести подпрограмму-функцию, вычисляющую этот минимум, с тем чтобы не расписывать этого вычисления каждый раз, когда этот минимум понадобится, а просто ссылаться на него по имени (как в операторе-функции). При этом на имя функции накладываются такие же ограничения, как и на имя переменной (например, если имя начинается с букв I, J, K, L, M, N, то значение функции будет целым).

```
      FUNCTION XMIN (X, Y)
      XMIN = X
      IF (X — Y) 3, 3, 2
    2 XMIN = Y
    3 RETURN      (ВОЗВРАТИТЬСЯ)
      END
```

Для вычисления n! можно написать такую подпрограмму-функцию:

```
      FUNCTION FACT (N)
      IF (N) 1, 2, 3
    1 FACT = 0.
      STOP
    2 FACT = 1.
      GO TO 5
    3 NF = 1
      DO 4 K = 2, N
    4 NF = NF * K
      FACT = NF
    5 RETURN
      END
```

Если теперь необходимо вычислить число сочетаний из N по M $C_N^M = N!/(M!\ (N — M)!)$ для некоторых M и N, то это можно записать таким образом:

```
CMN = FACT (N)/(FACT (M) * FACT (N — M)).
```

Из этих примеров видно, что последним выполняемым оператором подпрограммы-функции должен быть оператор RETURN, обеспечивающий возврат в основную программу, а последним в порядке написания — оператор END. Обращение к подпрограмме-функции осуществляется так же, как и к функции, определяемой оператором-функцией, или к стандартной функции, т. е. просто по имени, но с указанием фактических параметров (аргументов). Аргументами подпрограммы функции могут быть переменные без индексов, массивы (в этом случае в подпрограмме должен присутствовать оператор DIMENSION), имена функций или другие подпрограммы. Подпрограмма-функция должна записываться в конце основной программы, после оператора END.

Подпрограмма SUBROUTINE. Подпрограмма-функция, имея несколько аргументов, всегда дает только один результат. Подпрограмма SUBROUTINE применяется в тех случаях, когда необходимо получение нескольких результатов. Эту подпрограмму можно использовать также и в случаях, когда применяют подпрограмму-функцию.

Пример. Составить подпрограмму, вычисляющую для двух чисел их среднее арифметическое и среднее геометрическое

```
SUBROUTINE SR (X, Y, SA, SG)
SA = (X + Y)/2.
SG = SQRTF (X * Y)
RETURN
END
```

Пример. Подсчитать среднее арифметическое и среднее геометрическое двух выражений x^2 — sin $(a_5 x)$ и $a_1 x$. Используем операторы

```
Z = X * * 2 — SINF (A (5) * X)
CALL SR (Z, A (1) * X, V, W)
```

В ячейках V и W будут соответственно значения среднего арифметического и среднего геометрического.

Пример. Оформить в виде подпрограммы расчет среднего значения и среднеквадратичного отклонения для массива N чисел (максимальное значение N не превосходит 1000) в соответствии с формулами

$$XS = \frac{1}{N} \sum_{i=1}^{N} X_i;$$

$$\sigma = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} X_i^2 - \frac{N}{N-1} XS^2}$$

```
  SUBROUTINE STATIS (N, X, XS, SIGMA)
  DIMENSION X (1000)
  S = 0.
  S1 = 0.
  DO 1 I = 1, N
  S = S + X (I)
1 S1 = S1 + X (I) * * 2
  DN = N
  XS = S/DN
  SIGMA = SQRTF (S1/(DN — 1.) —DN/(DN — 1.) * XS**2)
  RETURN
  END
```

Оператор DN = N осуществляет перевод целого числа N в действительную форму, для того чтобы его можно было использовать в вычислениях с величинами действительного типа.

В отличие от рассмотренных функций других классов, подпрограмма SUBROUTINE не может вызываться неявным образом — путем упоминания ее имени. Требуется использование специального оператора CALL (ВЫЗВАТЬ), список параметров в котором должен быть согласован со списком параметров в операторе SUBROUTINE по количеству и типу параметров. Параметрами подпрограммы SUBROUTINE, как и подпрограммы-функции, могут быть переменные, массивы, функции.

Подпрограммы широко используют при программировании больших задач, что упрощает процесс отладки, поскольку каждую подпрограмму можно отлаживать, вносить в нее изменения и дополнения независимо от других подпрограмм. Существенно и то, что подпрограммы транслируются независимо от основной программы, а имена переменных и массивов, а также метки операторов подпрограммы имеют ло-

кальный характер, т. е. «действуют» только в пределах подпрограммы и никак не связаны с соответствующими элементами основной программы (в частности, операторы подпрограммы могут иметь те же метки, которые использовались в основной программе). В языке ФОРТРАН имеются средства, позволяющие в целях повышения компактности и вычислительной эффективности программы занимать общие области памяти данными подпрограммы и основной программой, снимая, таким образом, локальность переменных подпрограммы.

§ 6.6. ПРИМЕР СОСТАВЛЕНИЯ ФОРТРАН-ПРОГРАММЫ

Написать программу вычисления вектора **S** по формуле

$$\mathbf{S} = \sum_{k=1}^{N} \mathbf{A}^k \mathbf{Y},$$

где **A** — квадратная матрица 10-го порядка; **Y** — вектор 10-го порядка. Число N равно первому значению k, при котором норма вектора $\mathbf{Z} = \mathbf{A}^k\mathbf{Y}$ удовлетворяет условию $\mathbf{Z} < q$ (под нормой вектора понимают максимальный модуль его компонент $\mathbf{Z} = \max\limits_{1 \leqslant i \leqslant 10} |Z_i|$.

Полученный вектор **S** напечатать. Исходными данными в программе являются число q, вектор **Y** и матрица **A**. Возможный вариант программы на языке ФОРТРАН имеет следующий вид:

```
  DIMENSION S (10), A (10, 10), Z (10), X (10), Y (10)
1 FORMAT (F 12 . 6)
  READ 1, A
  READ 1, Y
  READ 1, Q
  DO 3 I = 1,10
  S (I) = 0.
3 Z (I) = Y (I)
4 CALL AZ (X, A, Z)
  DO 5 I = 1,10
  Z (I) = X (I)
5 S (I) = S (I) + Z (I)
  IF (ZN (Z) — Q) 6, 4, 4
6 PRINT 1, S
  STOP
  END
   SUBROUTINE AZ (W, T, V)
   DIMENSION W (10), T (10, 10), V (10)
   DO 16 I = 1,10
   W (I) = 0.
   DO 16 J = 1,10
16 W (I) = W (I) + T (I, J) * V (J)
   RETURN
   END
   FUNCTION ZN (F)
   DIMENSION F (10)
   ZN = ABSF (F (1))
   DO 20 I = 2,10
   IF (ZN — ABSF (F (I))) 19, 20, 20
19 ZN = ABSF (F (I))
20 CONTINUE
   RETURN
   END
```

Алгоритм умножения матрицы на вектор выделен в подпрограмму SUBROUTINE AZ (W, T, V), реализующую формулу

$$W_i = \sum_{j=1}^{10} T_{ij} V_j, \quad i = 1, 2, \ldots, 10.$$

Подпрограмма-функция FUNCTION ZN (F) вычисляет норму вектора **F**, определяемую выражением

$$ZN = \max_{1 \leqslant i \leqslant 10} |F_i|$$

методом перебора значений F_i.

Следует отметить, что как подпрограмма-функция, так и подпрограмма SUBROUTINE являются самостоятельными программами, транслирующимися отдельно, с переменными и метками, локализованными по области действия только в этих подпрограммах и не имеющими отношения к переменным и меткам в основной программе. Поэтому, например, массивы, используемые в подпрограммах, должны описываться в этих подпрограммах с помощью операторов DIMENSION.

§ 6.7. ОСОБЕННОСТИ БОЛЕЕ ПОЛНЫХ ВЕРСИЙ ЯЗЫКА ФОРТРАН

Ранее были рассмотрены основные элементы и конструкции минимального подмножества языка ФОРТРАН, реализуемого обычно на малых и средних ЦВМ. Мощные машины используют, как правило, более полные варианты языка, допускающие больше возможностей описания алгоритмов и облегчающие программирование сложных задач. Отметим наиболее важные особенности стандартного языка ФОРТРАН — ФОРТРАН-IV.

Явное описание типов. В языке Basic-ФОРТРАН тип переменной определяется по первой букве ее имени (переменная рассматривается как целая, если ее имя начинается с букв I, J, ..., N). В стандартном языке ФОРТРАН (далее просто ФОРТРАН) ограничение на возможность использования букв не является обязательным. Если программист хочет, чтобы ЦВМ работала с переменными M12 и LAMBDA, как с переменными действительного типа, то он может использовать оператор

```
REAL M 12, LAMBDA
```

Для явного описания переменных целого типа служит оператор INTEGER (ЦЕЛЫЙ). Например, в программе, использующей оператор

```
INTEGER X, Y, Z,
```

переменные X, Y, Z будут рассматриваться как целые. В предложения явной спецификации типа разрешено включать информацию о размерности массивов

```
REAL X (24), KB (5,3)
```

В последнем случае массивы X и KB не надо описывать в операторе DIMENSION. Кроме действительных и целых переменных, в ФОРТРАНе допускаются еще три типа переменных, которые отсутствуют в минимальном ФОРТРАНе:

DOUBLE PRECISION (С УДВОЕННОЙ ТОЧНОСТЬЮ)
COMPLEX (КОМПЛЕКСНЫЙ)
LOGICAL (ЛОГИЧЕСКИЙ)

Значения переменных с удвоенной точностью хранятся в машине как числа с плавающей запятой двойной длины. Комплексные числа представляются в машине в виде упорядоченной пары вещественных чисел (действительная и мнимая части числа), подчиняющихся обычным правилам представления чисел в ФОРТРАНе.

Логическими переменными называют такие переменные, которые принимают только два значения: TRUE (ИСТИНА) и FALSE (ЛОЖЬ). Значения логическим переменным могут присваиваться логическим оператором присваивания ФОРТРАНа, имеющим вид

a = b,

где a — логическая переменная; b — логическое выражение.

Логическое выражение может быть или логической константой, записываемой в виде. TRUE., или. FALSE., или отношением, или совокупностью логических переменных и выражений, разделенных знаками логических операций.

Операции отношения изображаются в ФОРТРАНе так:

.GT. БОЛЬШЕ
.GE. БОЛЬШЕ ИЛИ РАВНО
.LT. МЕНЬШЕ
.LE. МЕНЬШЕ ИЛИ РАВНО
.EQ. РАВНО
.NE. НЕ РАВНО

Точки входят в изображение операции. *Операции отношения* определяют отношения между вещественными или целыми (но не логическими и не комплексными) значениями, причем эти отношения могут быть либо истинными, либо ложными. Например, выражение X . GT . Y (X больше Y) представляет собой отношение, которое при одних значениях X и Y (например, X = 10., Y = 2 . 7) принимает значение TRUE, а при других — FALSE (при X = 1.6, Y = 2461.86). Значение этого отношения может быть присвоено логической переменной, например:

```
R = X . GT . Y
```

При этом в данной программе предварительно должен быть описан тип переменной R:

```
LOGICAL R
```

Над логическими переменными и логическими выражениями, частным случаем которых являются отношения, могут выполняться три логические операции, перечисленные в порядке убывания старшинства:

·NOT. НЕ
·AND. И
·OR. ИЛИ

Примеры

B = . NOT. A

Если A = TRUE, то логической переменной B будет присвоено значение FALSE, и наоборот,

C = X . AND . Y

В результате действия этого оператора логическая переменная C получит значение TRUE в том и только в том случае, когда обе логические переменные X и Y имеют одновременно значение TRUE

Z = T . OR . W

Z будет присвоено значение TRUE, если хотя бы одна из переменных T, W имеет значение TRUE.

P = X . LE . Y . OR . X . GT . Z

Логическая переменная P получит значение TRUE, если $X \leqslant Y$ или $X > Z$ и значение FALSE в противном случае (когда $Y < X \leqslant Z$).

Оператор логический IF. Логические выражения могут использоваться в операторе IF (логическая форма), например:

IF (A. LT . B) GO TO 4 — ЕСЛИ А МЕНЬШЕ В ПЕРЕЙТИ К 4.

Управление передается оператору с меткой 4, если значение логического выражения в скобках равно TRUE, и оператору, непосредственно следующему за оператором IF, в противном случае. Вместо оператора GO TO после скобок может стоять любой другой оператор ФОРТРАНа, кроме оператора DO или другого логического оператора IF. Например,

```
IF (L . AND.. NOT. M) F = 0
F = 1
```

Если логическая переменная L равна TRUE, а логическая переменная M равна FALSE, переменной F присваивается значение 0, в противном случае — значение 1.

Предложение IMPLICIT. Предложения явной спецификации типа REAL, INTEGER и другие используются для описания типа отдельных переменных. Предложение IMPLICIT употребляется для спецификации типа всех переменных, начинающихся с задаваемой буквы или группы подряд идущих букв. Например, предложение вида

IMPLICIT INTEGER (A, K — P), REAL (B — J)

указывает, что все переменные, начинающиеся с букв A, K, L, M, N, O, P, являются целыми, а переменные, начинающиеся с букв B, C, ..., H, I, J, — действительными. Переменные, начинающиеся с остальных букв алфавита, имеют свой первоначальный тип.

Приведенная спецификация может быть отменена использованием предложения явной спецификации типа, например предложение

```
INTEGER DELTA
```

отменяет указанную спецификацию для переменной DELTA, заставляя ЦВМ рассматривать ее как целую.

Стандартное обозначение функций. В ФОРТРАНе правила присвоения имен функциям одинаковы вне зависимости от класса функций (стандартная функция, оператор-функция, подпрограмма-функция).

1. Имя функции может включать до шести символов, из которых первый должен быть буквой. Не допускаются специальные символы.
2. Если значение функции должно быть целого типа, то имя функции должно начинаться с букв I, J, K, L, M, N.
3. Имена стандартных функций и операторов-функций не должны оканчиваться на F (вместо SINF (X) пишется SIN (X) и т. д.).

Тип функций может определяться также при помощи предложений явной спецификации типа (REAL, INTEGER и т. д.), в этом случае правило 2 не выполняется.

Операторы ввода — вывода. В операторах ввода — вывода ФОРТРАНа, реализованного на больших ЦВМ, кроме информации о вводимых и выводимых переменных, указывается информация и об используемых внешних устройствах (их номера).

Приведенное описание элементов языка Basic-ФОРТРАН и особенностей стандартного языка ФОРТРАН предназначено для начинающих программистов и, естественно, не является исчерпывающим. Описанных средств достаточно для составления реальных программ средней сложности. Опытные программисты, решающие сложные задачи, могут воспользоваться более мощными конструкциями, допускаемыми языком, описание которых можно найти в полных руководствах по ФОРТРАНу. К таким конструкциям относятся, например, вычисляемый оператор GO TO, соответствующий оператору перехода по переключателю switch в АЛГОЛе, оператор CALL DUMP, позволяющий производить распечатку определенных частей программы и используемый для ее отладки, а также большое число операторов, обеспечивающих работу с массивами данных на магнитных лентах и дисках.

ГЛАВА 7

АЛГОРИТМИЧЕСКИЙ ЯЗЫК АВТОКОД «ИНЖЕНЕР»

§ 7.1. ОСНОВНЫЕ СИМВОЛЫ И ПРОСТЕЙШИЕ КОНСТРУКЦИИ ЯЗЫКА АВТОКОД «ИНЖЕНЕР»

Алгоритмический язык автокод «Инженер» (АКИ) предназначен для описания алгоритмов, встречающихся в инженерной практике. Описание алгоритмов дается в виде последовательности простых фраз, состоящих из пояснительных слов (главным образом названий операторов) и формул, при записи которых использована символика, весьма близкая к общепринятой в математике.

Всякий вычислительный процесс можно разбить на ряд последовательных этапов, на каждом из которых выполняются арифметические или логические действия. Этапы записываются с помощью соответствующих арифметических или логических операторов.

Алфавит языка АКИ. В качестве алфавита языка АКИ используют второй международный телеграфный код, состоящий из цифр — 1, 2, 3, 4, 5, 6, 7, 8, 9, 0; русских заглавных букв — А, Б, ..., Я (кроме Ё, Ъ), используемых для записи названий всех операторов и поясняющих текстов; латинских заглавных букв A, B, ..., Z, используемых для обозначения переменных и функций. Знаки арифметические — «+», «—», «.», «=», « : »; препинания — «,» «?», «/», «(», «)», специальных — «␣» — пробел, «↑» — возведение в степень, «▽△» — конец оператора.

Элементами входного языка являются числа, переменные величины, элементарные функции и операторы.

Числа. На языке АКИ можно записывать целые и действительные числа.

Целые числа. Целые числа используют при решении многих счетно-экономических задач, в которых переменными являются единицы веса, длины и т. д., а также денежные единицы. Например, целое число 1969 и число — 65 на языке АКИ запишутся следующим образом: 1969 и —65.

Целые числа могут содержать не более девяти цифр и знак минус. Над ними производят лишь три арифметических действия: сложение, вычитание и умножение. Кроме того, целые числа используют для записи индексов переменных.

Действительные числа. Действительные числа могут быть записаны в одной из следующих форм: естественной (десятичной), нормальной (полулогарифмической); обыкновенной дроби.

При естественной форме записи допускается использование не более 24 символов, включая запятую и знак числа. Например, 125,03975; — 0,00017; — 32,125.

Числа, представляемые в нормальной форме с основанием 10, записываются следующим образом: $5700 = 57 \cdot 10^2$ как 57Ю2, $0{,}0135 =$

$=1{,}35\cdot10^{-2}$ как 1,35Ю—2, 10^{-6} как Ю—6. Буква Ю используется для обозначения основания системы счисления. Знак умножения перед буквой Ю не ставится, показатели степени представляются целыми числами, не превышающими по абсолютной величине 19, и не заключаются в скобки. Общее количество символов для изображения числа не должно превышать 24.

Обыкновенная дробь записывается с помощью символа «:»; отделяющего числитель от знаменателя, каждый из которых является целым десятичным числом без знака. Обыкновенная дробь всегда заключается в круглые скобки, так как рассматривается как одно число. Отрицательные дроби имеют знак минус перед скобкой. Например, $^1/_4$ запишется как (1: 4); — 2/5 как —(2 : 5).

Для записи числителя и знаменателя используют не более девяти знаков на каждый, не считая скобок, символа «:» и знака.

Переменные величины. Различают простые переменные и переменные с индексами, применяемые для описания массивов переменных.

Для обозначения переменных используют латинские заглавные буквы и цифры, но начинаться переменная всегда должна с буквы. Длина наименования произвольная, но друг от друга переменные должны отличаться в первых шести символах. Например, X, X2, SIGMA 10, SIGMA 11. Последние две переменные будут восприниматься одинаково, так как совпадают по первым шести символам.

Если переменная обозначается одной буквой, то в качестве такой буквы нельзя использовать буквы I, J, K, L, применяемые для обозначения индексов переменных.

Наименование переменной должно отличаться от наименования элементарных функций.

Для переменных с индексами вслед за их наименованием в индексных скобках указывается индекс. В качестве индексов можно использовать положительные целые числа, латинские буквы I, J, K, L, а также алгебраическую сумму вида $\alpha \pm \beta$, где α—одна из этих букв, β — целое число.

Следует помнить, что переменные с индексом применяют для обозначения элементов массивов, т. е. переменных, значения которых хранятся в последовательных ячейках памяти машины. Например,

X/3/ — третий элемент (x_3) одномерного массива X.

A/I/ — i-й элемент (a_i) массива A.

B/I, J/ — элемент i-й строки, j-го столбца двумерного массива B.

ALFA/K + 1, L — 3/ — элемент (K + 1)-й строки, (L — 3)-го столбца двумерного массива ALFA.

Между индексами двумерного массива ставится запятая. При этом первым указывается индекс строки, а вторым — индекс столбца.

Численные значения переменных могут представляться числами целого или действительного типа.

Функции. В языке АКИ нет каких либо ограничений на использование любых функций, но за девятью наиболее употребимыми функциями закреплены следующие наименования: SIN(X), COS(X), TG (X), ARCSIN (X), ARCCOS (X), ARCTG (X), LN(X), EXP (X), MOD(X), EXP(X)—соответствует операция возведения числа e в сте-

пень аргумента, а MOD (X) — представляет собой операцию вычисления абсолютной величины аргумента, X — аргумент, который в общем случае может быть алгебраическим выражением или переменной. Все эти функции вычисляют по стандартным программам.

Операторы. Оператор является самостоятельной единицей языка, указывающей содержание соответствующего этапа вычислительного процесса.

Операторы состоят из: 1) метки; 2) названия оператора; 3) содержательной части; 4) конца оператора.

Метка. В качестве меток используют любые целые числа от 1 до 127. Применяют метки в разветвляющихся программах для указания адреса оператора, к которому необходимо перейти. Метки в автокодовой программе могут располагаться в любой последовательности.

Название оператора. Название оператора указывает, какие арифметические или логические действия необходимо выполнить на данном этапе. Допускается сокращать название оператора до трех первых букв. Если название оператора состоит из нескольких слов, то между ними должен ставиться знак пробел «␣».

Содержательная часть. В содержательной части оператора указывается, что именно необходимо сделать на данном этапе.

Конец оператора. Конец оператора обозначается значком $\overset{\nabla}{\Delta}$ и указывает, что данный оператор закончился.

§ 7.2. ОПЕРАТОРЫ, ОРГАНИЗУЮЩИЕ СЧЕТ ПО ФОРМУЛАМ

Оператор ВЫЧИСЛИТЬ. Этот оператор организует счет по формулам. В содержательной части оператора записывают одну или несколько формул, разделенных знаком пробел, по которым должны производиться вычисления (формула состоит из вычисляемой переменной, стоящей в левой части), затем после знака равенства записываются константы, переменные и элементарные функции, разделенные знаками действия и круглыми скобками. Все знаки действия проставлять обязательно.

Порядок вычисления по формулам задается скобками. Если скобки не полностью определяют порядок действий, то вначале выполняются операции возведения в степень и вычисления элементарных фунций, затем операции умножения и деления и последними — операции сложения и вычитания.

Все переменные, стоящие в правой части, должны быть определены либо в предшествующих формулах данного оператора, либо в предшествующих операторах, либо в операторе ПОВТОРИТЬ.

Если вычисление выполняется над целыми числами, то перед первой переменной в левой части ставится символ «:».

Нельзя в одном операторе ВЫЧИСЛИТЬ производить действия над целыми и действительными числами.

Рассмотрим пример.

Пример. Пусть необходимо вычислить переменные y_1 и y_2 действительного типа и переменную u целого типа по формулам:

$$y_1=(a\sin x+b\sqrt[3]{x})/(ax+b^2)^5;$$
$$y_2=(ax^3+\ln x)/(a-b)^2;$$
$$u=2u_1-1.$$

Запись этих операторов будет иметь вид:

ВЫЧ — Y1 = (A . SIN(X) + B . X ▽ (1 : 3)) : (A. X+ B ▽ 2) ▽ 5 —
Y2 = (A . X ▽ 3 + LN (X)): (A — B) ▽ 2 ▽
ВЫЧ — : U = 2 . U1—1 ▽

При записи выражений допускается запись лишь в одну строку, например

$$\frac{a+x}{b}\rightarrow (A+X):B.$$

Для возведения в степень используется апостроф (▽), за которым пишется показатель степени, например $x^n \rightarrow$ X▽ N. Корень представляется как возведение в дробную степень, например $\sqrt[n]{x} \rightarrow$ X▽ (1 : N).

При правильной записи оператора количество открывающих скобок всегда равно количеству закрывающих скобок.

Оператор ИНТЕГРАЛ. Этот оператор служит для вычисления определенных интегралов по формуле Симпсона. В содержательной части оператора указывается наименование интеграла, затем в круглых скобках — пределы интегрирования, шаг интегрирования, требуемая точность. Далее записывается обязательно однобуквенное наименование подынтегральной функции и переменной интегрирования и, наконец, алгебраическое выражение подынтегральной функции.

Пример. Пусть необходимо вычислить

$$R=\int_a^b \frac{e^{-x^2\lambda}}{\sin x}\,dx$$

(шаг интегрирования H, точность вычислений E).

Оператор запишется в виде

ИНТ — R (ОТ — A — ДО — B — ШАГ — H — ТОЧ — E) FX =
= EXP (—X ▽ 2 . LAMBDA) : SIN (X) ▽

Пределы интегрирования, шаг интегрирования, точность интегрирования могут быть записаны либо в виде численных значений, либо в виде переменных. В последнем случае их значения должны быть определены в автокодовой программе до оператора ИНТЕГРАЛ.

Оператор АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ СИСТЕМА. С помощью этого оператора решаются методом перекрестного умножения системы линейных алгебраических уравнений до 55-го порядка. В информационной части оператора в круглых скобках записывается порядок системы, выраженный целым числом или простой переменной,

наименования двумерного массива коэффициентов при неизвестных и одномерного массива свободных членов. Например,

АЛГ — УРА — СИС — (25, A, B) ∇_{Δ}

Эта запись означает, что необходимо решить систему уравнений 25-го порядка; коэффициенты при неизвестных расположены в массиве A, а свободные члены в одномерном массиве B. Корни уравнений записываются в одномерный массив B вместо свободных членов.

§ 7.3. ОПЕРАТОРЫ РАЗМЕЩЕНИЯ ИНФОРМАЦИИ

Оператор ВВОД. Этот оператор используют для подготовки ввода в машину исходных данных и перевода их в двоичную систему счисления. Непосредственный ввод исходных данных осуществляется рабочей программой.

Оператор ВВОД позволяет вводить переменные и массивы действительного и целого типов. При этом все исходные данные должны располагаться на информационном бланке в том порядке, в котором они записаны в операторе ВВОД. Порядок расположения исходных данных следующий: сначала располагаются простые переменные действительного типа, затем одно- и двумерные массивы действительного типа, которые отделяются друг от друга запятой, затем после знака «:» записываются простые переменные целого типа, и далее одно- и двумерные массивы целого типа.

Предположим, что необходимо записать оператор ВВОД для следующей информации:

простые переменные x, y, δ_2 действительного типа;

переменные одномерного массива a_i в количестве 15 компонент;

переменные двумерного массива z_{ij} в количестве 100 компонент; причем количество строк равно n, а количество столбцов — 10 компонент;

простые переменные q, m целого типа;

переменные b_i одномерного массива целого типа в количестве 10 компонент;

переменные c_{ij} двумерного массива целого типа в количестве 25 компонент, разбитых на пять строк и s столбцов.

Для одномерных массивов после наименования в круглых скобках записывается количество элементов массива, а для двумерных массивов — количество элементов и после символа пробела — произведение количества строк на количество столбцов. Среди переменных целого типа первыми должны располагаться те переменные, которые указываются в данном операторе в качестве размерности двумерного массива (n, s) в том порядке, в каком они записаны в этом операторе. С учетом всего сказанного можно записать:

ВВО — X, Y, DELTA 2, A (15), Z (100 — N. 10) : N, S, Q, M, B (10), C (25 — 5.S) ∇_{Λ}

Здесь сначала записываются простые переменные x, y, δ_2, а затем одно- и двумерные массивы. Первой среди простых переменных целого типа необходимо записать переменную n (она встречается при описании массива Z), далее — переменную s (она встречается при описании массива С), а затем простые переменные q, m целого типа. Далее записываются одно- и двумерные массивы переменных целого типа.

На информационном бланке числа, соответствующие идентификаторам в операторе ВВОД, записываются группами в том порядке, который указан в этом операторе. Значения простых переменных действительного типа, простых переменных целого типа и каждого массива объединяются в отдельные группы. Начало и конец каждой группы выделяются с помощью слова «граница».

Оператор МАССИВ. Этот оператор предусматривает резерв памяти машины для хранения промежуточных и окончательных результатов. Информация о массиве задается так же, как и в операторе ВВОД. Например,

МАССИВ ⌣ А (90), Х (50 ⌣ М . N) : С (20) ▽△

Размерности массивов M, N являются обязательно целочисленными переменными и должны быть определены до оператора МАССИВ, например, с помощью оператора ВВОД или ВЫЧИСЛИТЬ.

Оператор НАЗВАТЬ. Этот оператор используется для формирования новых одномерных массивов и осуществляется непосредственный ввод чисел, записанных в его содержательной части. Числа, записанные в этом операторе, могут быть лишь действительного типа и отделяться друг от друга символом «.» (точка). Например,

НАЗВАТЬ ⌣ Х = — 17,5. (1 : 5). 2 Ю — 3. —4 ▽△

С помощью этого оператора будет сформирован одномерный массив Х, состоящий из четырех элементов: $x_1 = -17{,}5$; $x_2 = \frac{1}{5}$; $x_3 = 2 \cdot 10^{-3}$; $x_4 = -4$, которые будут записаны в четырех последовательных ячейках памяти машины.

Количество элементов массива не может быть меньше двух.

Другая функция, выполняемая этим оператором, связана с экономией памяти машины, когда на место старых массивов записываются новые. Для переименования массивов в содержательной части этого оператора пишется наименование нового массива и его размерности в круглых скобках, а затем, после знака равенства, наименование старого массива. Например,

НАЗВАТЬ ⌣ Х (100) = В ⌣ Z (40 ⌣ А . С) = Y ▽△

Количество элементов нового массива не должно быть больше количества элементов старого массива. Если количество элементов нового массива меньше количества элементов старого массива, то их можно расположить, начиная не с первого элемента старого масси-

ва; для этого после наименования старого массива в индексных скобках следует записать индекс этого элемента. Например,

НАЗВАТЬ ⌣ X (100) = B | 5 | ⌣ Z (40 ⌣ A . C) = Y | 2, 4 | ▽△

Новые элементы в такой массив будут записываться лишь после их вычисления, а до этого и старый, и новый массивы будут состоять из одних и тех же чисел, поэтому могут использоваться оба наименования.

С помощью оператора НАЗВАТЬ можно переименовать лишь массивы, описанные в операторах ВВОД и МАССИВ.

В общем случае оператор НАЗВАТЬ может выполнять обе эти функции одновременно.

§ 7.4. ОПЕРАТОРЫ ВЫДАЧИ РЕЗУЛЬТАТОВ

Результаты вычислений могут быть выданы на быстродействующий печатающий механизм (БПМ), перфоратор или телетайп.

На БПМ печать производится в один столбец на узкую бумажную ленту. Величина действительного типа печатается в форме с плавающей запятой, причем под мантиссу со знаком отводится восемь разрядов, а под порядок со знаком — три разряда. Числа целого типа вместе со знаком содержат не более девяти символов. Скорость печати 20 чисел/с.

С перфоратора информация выводится в виде системы отверстий на перфоленте. На телетайпе печать производится на широкую бумажную ленту. В каждой строке может быть напечатано 68 символов. Поскольку телетайп работает сравнительно медленно, то целесообразно информацию сначала выводить на перфоленту, а затем печатать с перфоленты на телетайпе независимо от машины. Рассмотрим правила записи операторов выдачи результатов.

Оператор НАПЕЧАТАТЬ ⌴ НА ⌴ БПМ ⌴. В содержательной части этого оператора через запятую записываются наименования переменных и массивов в том порядке, в котором они должны выдаваться на печать. Величины целого типа должны содержать признак (двоеточие).

Для одномерных массивов после наименования в круглых скобках необходимо указывать количество выводимых элементов, которое является целым числом или переменной целого типа. Для двумерных массивов после наименования в круглых скобках записывается его размерность, т. е. количество строк и столбцов, подлежащих выводу.

Если требуется вывести на печать массивы, начиная не с первого элемента, то после наименования в индексных скобках указывается номер элемента, начиная с которого необходимо осуществить печать, а затем в круглых скобках его размерность. Например,

НАП ⌣ НА ⌣ БПМ ⌣ A, X, : D, ALFA / 3 / (8), : B / 2, 1 / (3. 5) ▽△

С помощью этого оператора будут отпечатаны простые переменные A и X действительного типа, простая переменная D целого типа, восемь элементов α одномерного массива, начиная с третьего элемента

α_3, три строки и пять столбцов элементов двумерного массива B целого типа, начиная с элемента $b_{2,1}$, находящегося во второй строке первого столбца.

При печати после каждой простой переменной, одномерного массива переменных и строки двумерного массива будет интервал.

Пример. Рассмотрим, как на бумажной ленте будет отпечатана информация по этому оператору.

Пусть $A = -0{,}13457 \cdot 10^{-3}$; $X = 0{,}79999 \cdot 10^{6}$; $D = -121$.

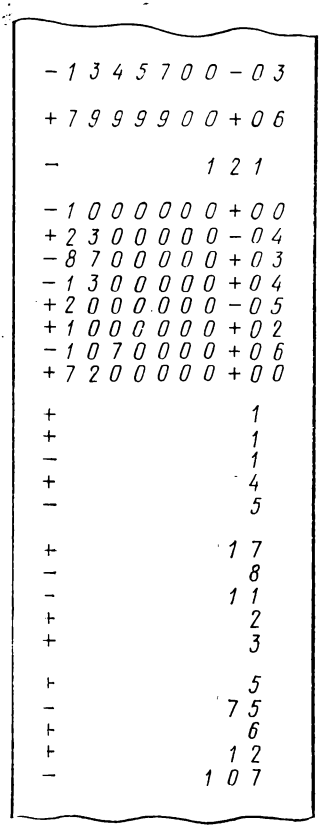
Массив

$$\alpha = \| 0{,}74 \cdot 10^{-2};\ 0{,}5;\ -0{,}1;\ 0{,}23 \cdot 10^{-4};\ -0{,}87 \cdot 10^{3};\ -0{,}13 \cdot 10^{4};$$
$$0{,}2 \cdot 10^{-5};\ 0{,}1 \cdot 10^{2};\ -0{,}107 \cdot 10^{6};\ 0{,}72 \|.$$

Массив

$$B = \begin{Vmatrix} 0 & -1 & 7 & -11 & 4 \\ 1 & 1 & -1 & 4 & -5 \\ 17 & -8 & -11 & 2 & 3 \\ 5 & -75 & 6 & 12 & -107 \end{Vmatrix}.$$

Эти числа на бумажной ленте будут располагаться следующим образом:



Нельзя непосредственно на печать выводить переменную с буквенными индексами, даже если численное значение этих индексов определено предыдущими операторами. В таких случаях необходимо переименовать переменную в простую переменную, например с помощью оператора ВЫЧИСЛИТЬ:

НАП ⌣ НА ⌣ БПМ ⌣ X / I, J / $\begin{matrix}\nabla\\ \Delta\end{matrix}$ — неправильная запись,

ВЫЧ ⌣ Y = X / I, J / $\begin{matrix}\nabla\\ \Delta\end{matrix}$

НАП ⌣ НА ⌣ БПМ ⌣ Y $\begin{matrix}\nabla\\ \Delta\end{matrix}$ — правильная запись.

Оператор НАПЕЧАТАТЬ ␣ НА ␣ ТЕЛЕТАЙПЕ ␣. Этот оператор служит для вывода численных значений простых переменных вместе с их наименованием на перфоратор, с последующей распечаткой на рулонном телетайпе. Печать осуществляется на широкую бумажную ленту. В каждой строке может быть отпечатано не более 68 символов, включая пробелы.

Точность, с которой будут выводиться на печать переменные, указывается в содержательной части оператора перед словом ЗНАК в виде числа, определяющего количество знаков после запятой, выводимых на печать. Если слово ЗНАК и точность отсутствуют, то на печать выводится лишь целая часть числа. Число, определяющее точность, не должно превышать 19.

После точности указывается список переменных, выводимых на печать, так же как и в операторе печати на БПМ. Например,

НАП ⌣ НА ⌣ ТЕЛ ⌣ 3 ⌣ ЗНАКА ⌣ A /1.5/. ı C /2/, N $\begin{matrix}\nabla\\ \Delta\end{matrix}$.

Пример. Если $a_{1,5} = -17{,}30122$, $C_2 = 17$, $N = 0{,}27304 \cdot 10^{-1}$, то на бумажной ленте в десятичной системе счисления будет отпечатано следующее:

A /1,5/ = — 17,301
C /2/ = 17
N = 0,027.

Оператор НАПЕЧАТАТЬ ␣ ТАБЛИЦУ. Этот оператор служит для печати на телетайпе численных значений переменных и массивов в виде таблицы. Как и в предыдущем операторе, в операторе НАПЕЧАТАТЬ ТАБЛИЦУ указывается точность, а выводимые переменные и массивы записываются так же, как и в операторе НАП␣НА␣ БПМ, с той лишь разницей, что каждой переменной предшествует формат. *Формат* — количество позиций, отводимых под данную переменную или массив. Числа, соответствующие каждому наименованию, печатаются в виде одного столбца. При этом для двумерных массивов печатается сначала первая строка, затем после пропуска вторая строка и т. д.

Пример. Надо отпечатать таблицу следующих значений переменных с точностью до трех знаков после запятой:

$$A = -73{,}0;\quad B = \begin{Vmatrix} 0{,}2572 & -4{,}01 \\ 17{,}2 & 0{,}0074 \\ 2{,}751 & -0{,}149 \end{Vmatrix};\quad C = \| 1 \quad -5 \quad 6 \quad 12 \|.$$

Выберем форматы: для переменных A и B отведем по восемь позиций, для переменной C — пять позиций. При выборе формата следует иметь в виду, что суммарный формат по всем переменным не должен превышать 68 позиций и, что формат не должен быть меньше количества символов, необходимых для записи целой части числа, знака числа, запятой и количества знаков после запятой, определяемых точностью, так как в противном случае будут отбрасываться младшие знаки числа. Если формат не позволяет записать даже целую часть числа, то вместо числа печатается знак вопроса «?». Тогда оператор имеет вид:

НАП — ТАБ — 3 — ЗНАКА — 8 — A, 8 — B (3.2), 5 — C (4) $\nabla\Delta$

(печать чисел осуществляется вплотную к правому краю столбца.)

Таблица в этом случае будет иметь вид:

—73,000	0, 2 5 7	1
	—4, 0 1 0	—5
		6
		12
	17, 2 0 0	
	0, 0 0 7	
	2, 7 5 1	
	—0, 1 4 9	

Следовательно, все строки матрицы разворачиваются одна за другой и печатаются в виде одного столбца.

Если нужно отпечатать массив X, состоящий из пяти строк и трех столбцов, в виде матрицы, то оператор имеет вид

НАП — ТАБ — 5 — ЗНАКОВ — 10 — X /1,1/ (5 . 1), 10 — X / 1,2 / (5.1), 10 — X/1,3/ (5.1) $\nabla\Delta$

В первом столбце печатаются элементы матрицы, начиная с x_{11}, в количестве пяти строк и одного столбца, т. е. первый столбец матрицы, во втором столбце, начиная с x_{12}, тоже пять строк и один столбец, т. е. второй столбец матрицы, в третьем столбце, начиная с x_{13}, т. е. третий столбец матрицы.

Оператор НАПЕЧАТАТЬ ▭ ТЕКСТ. Этот оператор предназначен для вывода на печать текстовой информации, которая записывается после его наименования. С помощью этого оператора можно выводить на телетайп название таблиц, пояснения к ним и т. д. В тексте не должны встречаться символы «$\nabla\Delta$» и «?», имеющие служебное значение. Например,

НАП — ТЕКСТ — АРГУМЕНТ — — — — — — ЗНАЧЕНИЕ — ФУНКЦИИ $\nabla\Delta$

С помощью этого оператора будет напечатано слово АРГУМЕНТ, отступив на пять позиций от левого края, затем будут пропущены шесть позиций и напечатаны слова ЗНАЧЕНИЕ ФУНКЦИИ.

Всякая программа начинается с заглавия, в котором указывается название программы или фамилия автора, и т. д. Заглавие заканчивается знаком «$\overset{\nabla}{\Delta}$». Например,

РЕШЕНИЕ — ДИФФЕРЕНЦИАЛЬНОГО — УРАВНЕНИЯ $\overset{\nabla}{\Delta}$

или

ЗАДАЧА — ИВАНОВА $\overset{\nabla}{\Delta}$

Можно вместо заглавия ставить лишь значок «$\overset{\nabla}{\Delta}$».

Заглавие запоминается в памяти машины и никаких действий по нему не производится. После заглавия записываются операторы, осуществляющие решение поставленной задачи. Операторы записываются в том порядке (один под другим), в котором они должны выполняться.

Заканчивается программа оператором КОНЕЦ $\square$ $\overset{\nabla}{\Delta}$, по которому формируется команда останова машины.

Последним в программе записывается служебное слово НАЧАЛО, в содержательной части которого указывается метка самого первого оператора программы, с которого начинается процесс решения задачи. Например,

НАЧАЛО — 5 $\overset{\nabla}{\Delta}$.

Пример. Вычислить

$$Z=1+\frac{\ln[(x_1+x_2)R]}{x_1+x_2+x_3},$$

где x_1, x_2, x_3 — корни системы уравнений

$$\begin{cases} 2x_1-3,5x_2+x_3=0,5; \\ x_1+2x_2+3x_3=4,7; \\ x_1-x_2-0,5x_3=-3; \end{cases}$$

$R=\int\limits_0^{\alpha}\frac{e^{-0,2\alpha^2}}{\sqrt{1+\alpha^2}}d\alpha$, шаг интегрирования h, точность -1.10^{-5}.

Прежде чем вычислить значение функции Z, необходимо найти значение интеграла и корни системы.

Для решения системы уравнений в машину надо ввести значения всех коэффициентов при неизвестных и свободные члены, а для вычисления интеграла — численное значение шага интегрирования h. Вся информация вводится в машину перед решением задачи с помощью оператора ВВОД. Для этого массив коэффициентов при неизвестных обозначается через А, а массив свободных членов — через В, т. е.

$$A=\begin{Vmatrix} 2 & -3,5 & 1 \\ 1 & 2 & 3 \\ 1 & -1 & -0,5 \end{Vmatrix}; \quad B=\|0,5 \quad 4,7 \quad -3\|.$$

Массив А двумерный и состоит из девяти элементов (три строки и три столбца), а массив В одномерный и состоит из трех элементов.

Программа решения этой задачи имеет вид:

```
ВЫЧИСЛЕНИЕ — ФУНКЦИИ — Z ▽△
1. ВВОД — Н, В (3), А (9 — 3.3) ▽△
АЛГ — УРА — СИС — (3, А, В) ▽△
ИНТ — R (ОТ — 0 — ДО — 1 — ШАГ — Н — ТОЧ — Ю — 5)
FT = EXP (— 0,2 . T ▽ 2) : (1 + T ▽ 2) ▽ (1 : 2) ▽△
ВЫЧ — Z = 1 + LN((B/1/ + B/2/). R) : (B /1/ + B/2/ + B/3/) ▽△
НАП — НА — БПМ — Z, В (3), R ▽△
КОН — ▽△
НАЧ — 1 ▽△
```

После оператора ВВОД — расположены операторы, осуществляющие вычисление корней системы уравнений и определенного интеграла. После выполнения оператора АЛГ — УРА — СИС — корни уравнений будут храниться в массиве В вместо ненужных теперь свободных членов. Следовательно, корни уравнений являются элементами этого массива, т. е. имеют наименование В и соответствующие индексы.

При записи оператора ИНТЕГРАЛ — допускается запись лишь однобуквенной переменной интегрирования, а α — греческая буква и записывать ее латинскими буквами ALFA нельзя. Поэтому переменную α переименовали в переменную T.

Следующие операторы вычисляют значение искомой функции Z и осуществляют печать значений интеграла, корней системы уравнений и функции Z.

На этом решение задачи заканчивается, поэтому далее следуют оператор КОНЕЦ — ▽△ и служебное слово НАЧАЛО — 1 ▽△. С оператора с меткой 1 должна начинаться программа, поэтому эта метка присваивается оператору ВВОД—.

§ 7.6. ОПЕРАТОРЫ УПРАВЛЕНИЯ

В автокодовой программе принят естественный порядок выполнения операторов, т. е. операторы выполняются один за другим в порядке их расположения в программе. Если по характеру вычислений необходимо нарушить такую последовательность, то используют операторы, управляющие ходом вычислений. К таким операторам относятся операторы: условной и безусловной передачи управления; обращения к подпрограмме; организации циклов.

Оператор ПЕРЕЙТИ. Если по характеру вычислений необходимо из данного места программы перейти к оператору с меткой N, то используют оператор безусловной передачи управления ПЕРЕЙТИ ⌴, в содержательной части которого указывается метка N. Например,

```
ПЕРЕЙТИ — 16 ▽△
```

В этом случае всегда после оператора ПЕРЕЙТИ будет выполняться не следующий по порядку оператор, а оператор с меткой 16, а далее оператор, находящийся за оператором с меткой 16. Такой оператор называют *оператором безусловной передачи управления.*

Оператор ЕСЛИ. В случае, когда переход к тому или иному оператору осуществляется в зависимости от выполнения какого-то условия, используют оператор условной передачи управления ЕСЛИ. В со-

держательной части этого оператора с помощью знаков отношения записывается условие перехода, в левой части которого будет находиться арифметическое выражение, функция или переменная, а после знака отношения — переменная или число. Знакам отношения на языке АКИ соответствуют следующие знаки:

Знаки отношения	Символы знаков отношения на языке АКИ
$<$	⌙ (
$\leqslant$	⌙ (=
$>$	⌙)
$\geqslant$	⌙) =
$=$	⌙ =

После условия перехода ставится слово ТО, которое выделяется пробелом с обеих сторон, за словом ТО ставится метка того оператора, к которому необходимо перейти, если условие перехода выполнено. Например,

ЕСЛИ — SIN (2.X — A) —) = DELTA — TO — 4 ▽△

В этом случае, если sin $(2x - a) \geqslant \delta$, то следующим будет выполняться оператор с меткой 4, в противном случае — следующий по порядку оператор.

Иногда необходимо в случае невыполнения условия перехода переходить не к следующему по порядку оператору, а к какому-либо другому оператору. В этом случае оператор ЕСЛИ ⌴ ... дополняется словом ИНАЧЕ с указанием метки того оператора, к которому надо перейти. Например,

ЕСЛИ — X ▽ 2 — COS (X) —) 0 — TO — 2 — ИНАЧЕ — 10 ▽△

В этом случае, если $x^2 - \cos x > 0$, то следующим будет выполняться оператор с меткой 2, в противном случае — оператор с меткой 10.

Пример. Пусть необходимо решить квадратное уравнение вида $ax^2 + bx + c = 0$ (см. гл. 3 задачу 4, рис. 3.4).

Решение этой задачи начнем с вычисления подкоренного выражения:

ВЫЧ — D = B ▽ 2 — 4 . A . C — Z = 2.A ▽△

Согласно условию задачи в зависимости от знака D необходимо вычислять либо корни уравнения x_1 и x_2 (если $D \geqslant 0$), либо действительную часть e и коэффициент при мнимой единице f (если $D < 0$).

Значит, следующим оператором должен быть оператор условной передачи управления

ЕСЛИ — D — (0 — TO — 2 ▽△

В этом случае, если $D < 0$, то следующим будет выполняться оператор с меткой 2. Значит с этого оператора должно начинаться вычисление e и f. Иначе будет выполняться следующий по порядку оператор, с которого должно начинаться вычисление x_1 и x_2. Таким образом, следующим должен быть оператор, вычисляющий x_1 и x_2, а затем оператор печати x_1 и x_2:

ВЫЧ — X1 = (—B + D ▽ (1 : 2)) : Z — X2 = (—B — D▽ (1 : 2)) : Z ▽△

НАП — НА — БПМ — X 1, X2 ▽△

После того как будут вычислены и отпечатаны значения корней уравнения, задача считается решенной и необходимо перейти к оператору КОНЕЦ — $\overset{\nabla}{\underset{\Delta}{}}$.

Но оператор КОНЕЦ — следующим по порядку быть не может, так как существует условие, что D < 0. Поэтому для перехода к оператору КОНЕЦ используем оператор безусловной передачи управлений ПЕРЕЙТИ — 3 $\overset{\nabla}{\underset{\Delta}{}}$. С помощью этого оператора перейдем к оператору с меткой 3, которую присвоим оператору КОНЕЦ — $\overset{\nabla}{\underset{\Delta}{}}$.

Далее следует оператор, вычисляющий e и f, если условие, записанное выше, выполняется. Этому оператору уже присвоили метку 2. За ним должен идти оператор печати e и f.

2. ВЫЧ — E = —B: Z — F = MOD (D) $^{\nabla}$ (1 : 2) : Z $\overset{\nabla}{\underset{\Delta}{}}$

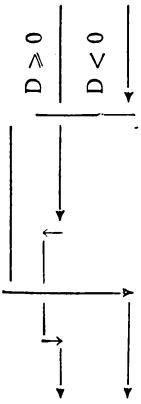
НАП — НА — БПМ — E, F $\overset{\nabla}{\underset{\Delta}{}}$

На этом решение задачи для всех возможных условий заканчивается, поэтому в конце программы записываем

3. КОНЕЦ — $\overset{\nabla}{\underset{\Delta}{}}$

НАЧАЛО — 1 $\overset{\nabla}{\underset{\Delta}{}}$

Таким образом, целиком программа решения этой задачи имеет вид:



$\overset{\nabla}{\underset{\Delta}{}}$

1. ВВОД — A, B, C $\overset{\nabla}{\underset{\Delta}{}}$

ВЫЧ — D = B$^{\nabla}$2 — 4. A. C — Z = 2 . A $\overset{\nabla}{\underset{\Delta}{}}$

ЕСЛИ — D — (0 — ТО — 2 $\overset{\nabla}{\underset{\Delta}{}}$

ВЫЧ — X1 = (—B + D $^{\nabla}$ (1 : 2)) : Z — X2 = (—B — D$^{\nabla}$ (1 : 2)) : Z $\overset{\nabla}{\underset{\Delta}{}}$

НАП — НА — БПМ — X1, X2 $\overset{\nabla}{\underset{\Delta}{}}$

ПЕР — 3 $\overset{\nabla}{\underset{\Delta}{}}$

2. ВЫЧ — E = — B : Z — F = MOD (D) $^{\nabla}$ (1 : 2) : Z $\overset{\nabla}{\underset{\Delta}{}}$

НАП — НА — БПМ — E, F $\overset{\nabla}{\underset{\Delta}{}}$

3. КОН — $\overset{\nabla}{\underset{\Delta}{}}$

НАЧ — 1 $\overset{\nabla}{\underset{\Delta}{}}$

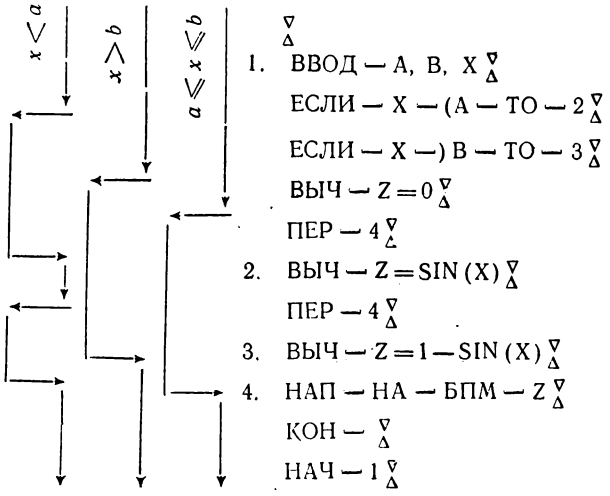
Стрелками указаны возможные последовательности выполнения операторов.

В случае, когда процесс вычисления имеет три разветвления, надо использовать два оператора ЕСЛИ. С помощью одного оператора ЕСЛИ можно проверить выполнение первого условия. Если оно окажется не выполненным, то следующий оператор проверяет второе условие.

Пример. Вычислить и отпечатать

$$Z = \begin{cases} \sin x, & \text{если } x < a; \\ 0, & \text{если } a \leqslant x \leqslant b; \\ 1 - \sin x, & \text{если } x > b. \end{cases}$$

Программа решения этой задачи имеет вид:



Стрелками показан порядок выполнения операторов для всех трех условий.

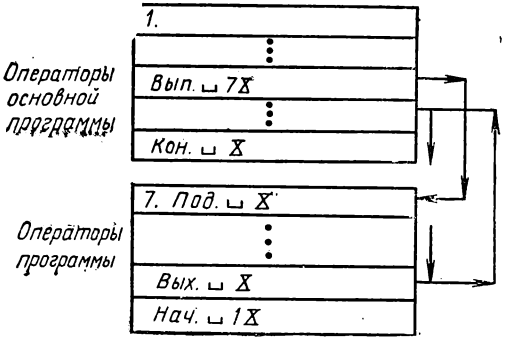
Оператор ВЫПОЛНИТЬ. Неоднократно повторяющиеся участки программы целесообразно выделять в подпрограмму, к которой обращаются всякий раз, как появляется необходимость ее выполнения. За счет этого устраняется многократная запись операторов, образующих подпрограмму. После выполнения подпрограммы необходимо возвращаться в основную программу и продолжить ее выполнение. Для этих целей в языке АКИ используют оператор ВЫПОЛНИТЬ, в содержательной части которого указывается метка подпрограммы. Например,

ВЫП — 7

(по этому оператору будет осуществлен переход к подпрограмме с меткой 7).

Подпрограмма начинается заголовком ПОДПРОГРАММА ␣, после которого может стоять текст, например, название подпрограммы. Заканчивается подпрограмма словом ВЫХОД ␣, по которому осуществляется выход из подпрограммы и возврат в основную программу.

Все подпрограммы записываются после основной программы, а после последней подпрограммы ставится служебное слово НАЧ␣. Таким образом, структура программы выглядит так (стрелками показана последовательность выполнения операторов):



Пример. Вычислить

$$Z=\frac{\operatorname{th} x-\operatorname{th} y}{\operatorname{th}(x-y)},$$

если известно, что

$$\operatorname{th} a=\frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}.$$

При решении этой задачи целесообразно вычисление тангенса гиперболического вынести в подпрограмму, а перед обращением к ней вычислить аргумент a этого тангенса, равным сначала x, затем y и, наконец, $(x - y)$. Тогда программа будет иметь вид:

```
▽
△
1. ВВОД ─ X, Y ▽△
   ВЫЧ ─ A = X ▽△
   ВЫП ─ 5 ▽△
   ВЫЧ ─ A = Y ─ Z1 = TH ▽△
   ВЫП ─ 5 ▽△
   ВЫЧ ─ A = X ─ Y ─ Z2 = TH ▽△
   ВЫП ─ 5 ▽△
   ВЫЧ ─ Z = (Z1 ─ Z2) : TH ▽△
   НАП ─ НА ─ БПМ ─ Z ▽△
   КОН ─ ▽△
5. ПОД ─ ТАНГЕНСА ─ ГИПЕРБОЛИЧЕСКОГО ▽△
   ВЫЧ ─ R1 = EXP (A) ─ R2 = EXP (─A) ▽△
   ВЫЧ ─ TH = (R1 ─ R2) : (R1 + R2) ▽△
   ВЫХ ─ ▽△
   НАЧ ─ 1 ▽△
```

Обращение к подпрограммам библиотеки стандартных программ. С помощью оператора ВЫПОЛНИТЬ осуществляется обращение к подпрограммам, составляемым программистом.

Существует целый набор подпрограмм, которые заранее составлены и хранятся в библиотеке стандартных программ (БСП). Обращение к ним на языке АКИ происходит по-разному. Например, обращение к стандартным подпрограммам вычисления девяти элементарных функций (sin, cos и т. д.) осуществляется автоматически всякий раз, как только в основной программе встречается наименование этих функций; к подпрограммам перевода из десятичной системы счисления в двоичную и обратно — операторами ВВОД и операторами печати соответственно; к наиболее часто встречающимся подпрограммам вычисления определенного интеграла и вычисления корней системы алгебраических уравнений — с помощью специальных операторов ИНТЕГРАЛ ␣ и АЛГ ␣ УРАВ ␣ СИСТ ␣.

Обращение к остальным подпрограммам БСП осуществляется с помощью оператора БИБЛИОТЕЧНАЯ ␣ ПРОГРАММА ␣.

После наименования этого оператора записывается восьмеричный номер соответствующей подпрограммы и затем в круглых скобках — необходимые для ее выполнения параметры, состав и порядок расположения которых определяется при разработке каждой конкретной подпрограммы. Например,

БИБ — ПРО — 51 (X, Y / I /, 9, T / I, K /, 11, Z) $\frac{\nabla}{\Delta}$

Оператор ПОВТОРИТЬ. При решении многих задач требуется многократно повторять одни и те же вычисления. В этом случае целесообразно так составлять программу, чтобы операторы, производящие эти вычисления, циклически повторялись. Для управления циклическими вычислительными процессами используют оператор ПОВТОРИТЬ ␣.

Оператор ПОВТОРИТЬ ␣ ставится в конце циклического участка автокодовой программы. Метка, стоящая за названием оператора, указывает, с какого места программы будет начинаться повторение.

Оператор ПОВТОРИТЬ состоит из: 1) названия оператора; 2) метки; 3) параметров, подлежащих изменению в цикле и правил их изменения; 4) условия окончания цикла.

В качестве параметров цикла, изменяющихся при каждом новом повторении, используются простая переменная или индекс (для переменной с индексом), изменяющиеся монотонно.

После метки через пробел в этом операторе записывается простая переменная или индекс (параметр цикла), далее через знак равенства — их начальные значения и через пробел — величина приращения, заключенная в круглые скобки.

Если в цикле изменяется несколько параметров, то информация о них отделяется точкой.

Начальное значение переменной и шаг ее изменения обязательно должны быть действительными числами или простыми переменными действительного типа, а начальное значение индекса и шаг его изменения — целыми числами или простыми переменными целого типа.

В конце оператора указывается условие окончания цикла, которое может быть задано следующими способами.

1. Количеством повторений цикла. В этом случае условие окончания цикла отделяется точкой от информации о последнем параметре. Количество повторений может задаваться целым числом или переменной целого типа. Например,

$$\text{ПОВ} - 17 - X = A - (0{,}5) \cdot 100 \begin{matrix}\nabla \\ \Delta\end{matrix}$$

Т. е. повторить участок программы, начиная с оператора с меткой 17, сто раз. При каждом новом повторении цикла будет изменяться величина параметра x начиная со значения A с шагом изменения этого параметра, равным 0,5.

$$\text{ПОВ} - 12 - I = 1 - (1) \,.N \begin{matrix}\nabla \\ \Delta\end{matrix}$$

Т. е. повторить участок программы, начиная с оператора с меткой 12, N раз. При этом индекс переменных I будет изменяться последовательно от 1 с шагом 1, т. е. принимать целочисленные значения 1, 2, 3, ..., N.

2. Конечным значением одной из простых переменных. Эта переменная записывается последней в списке изменяющихся параметров, после чего через пробел указывается условие, при выполнении которого цикл будет повторяться. Условие записывается с помощью знаков отношения, за которыми стоит либо действительное число, либо переменная действительного типа. Например,

$$\text{ПОВ} - 22 - K = 1 - (1).X = X\,0 - (H) - (= 100 \begin{matrix}\nabla \\ \Delta\end{matrix}$$

Т. е. повторить участок программы, начиная с оператора с меткой 22. При каждом повторении изменяется индекс K от 1 с шагом 1 и простая переменная X от X 0 с шагом H. Цикл будет повторяться до тех пор, пока значение переменной X будет оставаться меньшим или равным 100.

3. Конечным значением индекса переменной. Этот индекс записывается последним в списке изменяющихся параметров и после пробела указывается конечное значение индекса. Это значение может быть целым числом или переменной целого типа. Например,

$$\text{ПОВ} - 34 - X = T - (P)\ I = 1 - (H) - N \begin{matrix}\nabla \\ \Delta\end{matrix}$$

Т. е. повторить, начиная с оператора с меткой 34, участок циклической программы. При каждом повторении цикла изменяется простая переменная X от начального значения T с шагом P и индекс I от начального значения 1 с шагом H. Цикл будет повторяться до значения индекса I = N включительно.

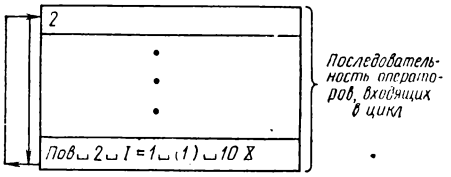
Если начальное значение параметра, шаг его изменения и условие окончания цикла задаются простыми переменными, то значения этих переменных должны быть определены в автокодовой программе до оператора ПОВТОРИТЬ ⌴.

Следует заметить, что если в операторе ПОВТОРИТЬ не задано условие окончания цикла, то для выхода из циклической программы следует использовать оператор ЕСЛИ. В этом случае оператор ПОВТОРИТЬ будет только задавать начальные значения и изменять параметры цикла.

Для организации цикла необходимо задавать начальные значения параметров, изменяющихся при каждом новом повторении цикла (простых переменных и индексов), правило их изменения и условие окончания цикла.

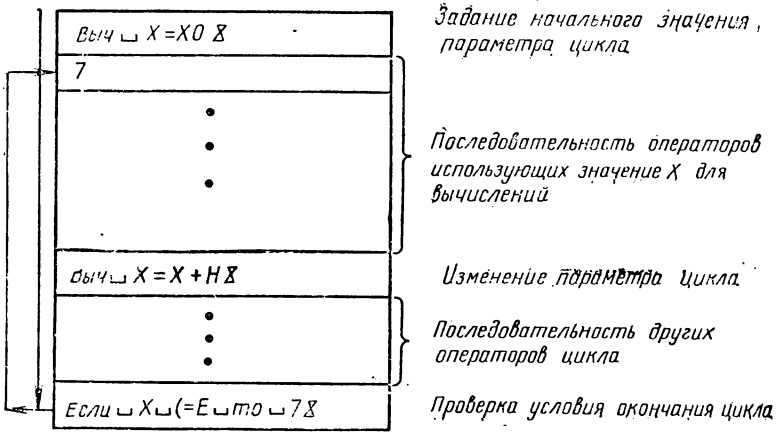
Рассмотрим способы организации простого цикла.

Способ 1. Цикл может быть организован с помощью оператора ПОВТОРИТЬ, который производит засылку начальных значений параметров, изменяет их указанным образом и проверяет условие окончания цикла. Этот оператор ставится в конце циклического участка программы, схема которой имеет вид:



(стрелками указана последовательность выполнения операторов). В этом случае 10 раз выполняется последовательность операторов, начиная с оператора с меткой 2 и кончая оператором ПОВТОРИТЬ. После каждого выполнения оператора ПОВ ␣ производится изменение параметра цикла I на единицу и осуществляется переход к оператору с меткой 2. После десятикратного выполнения этого участка программы выполняется оператор, следующий за оператором ПОВТОРИТЬ.

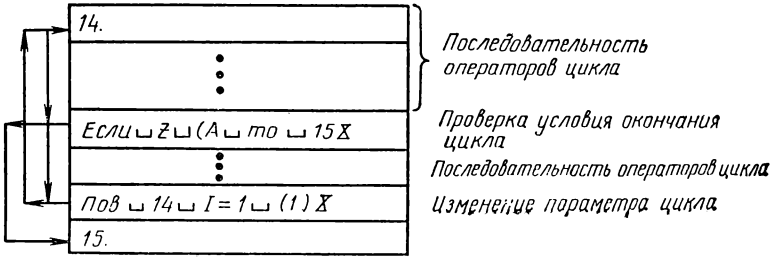
Способ 2. Часто цикл можно записать с помощью двух операторов ВЫЧИСЛИТЬ (один из которых задает начальные значения параметров цикла, а второй изменяет их) и оператора ЕСЛИ, который проверяет условие окончания цикла. Схема программы в этом случае имеет вид:



Проверка условия окончания цикла может осуществляться не только по параметру X, но и по какому-либо вычисленному результату.

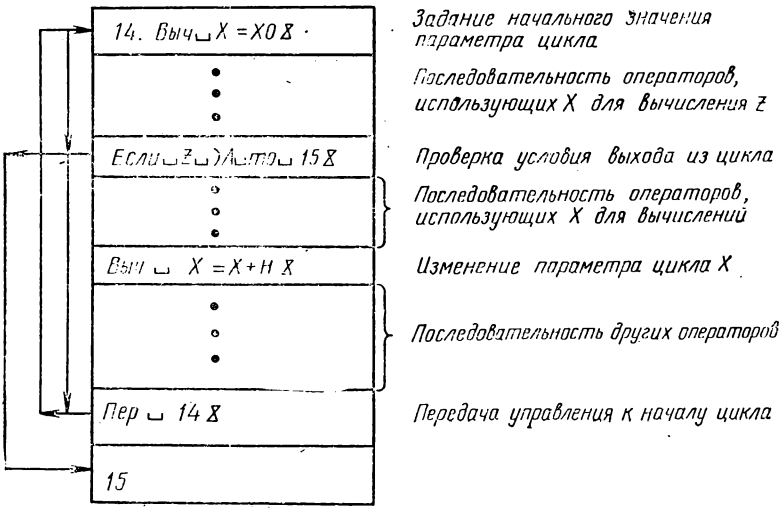
Параметрами цикла могут быть в этом случае лишь простые переменные, так как над индексами никакие операции не производятся.

Способ 3. Оператор ПОВТОРИТЬ используют лишь для изменения параметров цикла. Тогда выход из цикла можно осуществить с помощью оператора ЕСЛИ, записанного внутри цикла. Схема программы в этом случае имеет вид:



Здесь индекс не может быть параметром, по которому проверяется условие окончания цикла оператором ЕСЛИ.

Способ 4. Если параметрами цикла являются только простые переменные, то вместо оператора ПОВ ␣ можно использовать два оператора ВЫЧИСЛИТЬ ␣ для задания начального значения параметра и его изменения и оператор ПЕРЕЙТИ ␣ для передачи управления. Схема программы в этом случае имеет вид:



Нельзя входить внутрь цикла с помощью операторов (ПЕРЕИТИ, ЕСЛИ, ПОВТОРИТЬ), т. е. входить в цикл можно только через его начало.

Пример. Пусть необходимо вычислить и отпечатать на БПМ 100 значений функций $y_i = \sin(\alpha x_i)/x_i$, где $x_i = 1, 2, \ldots, 100$ (см. задачу 3.5).

Будем выполнять операторы вычисления значения функции и печати его. Эти операторы вместе с оператором ПОВТОРИТЬ образуют цикл. Поскольку каждое вычисленное значение переменной y сразу будет выводиться на печать, то целесообразно представить ее простой переменной, x будем считать простой переменной, а закон ее изменения зададим в операторе ПОВТОРИТЬ. С учетом сказанного программа имеет вид:

```
1. ВВОД — ALFA
2. ВЫЧ — Y = SIN (ALFA . X) ; X
   НАП — НА — БПМ — Y
   ПОВ — 2 — X = 1 — (1) — (= 100
   КОН —
   НАЧ — 1
```

Если необходимо сохранить в памяти машины все значения функции y, то необходимо, во-первых, выделить место для их хранения с помощью оператора МАССИВ и, во-вторых, вычислять функцию как переменную с индексом. Но поскольку в операторе печати нельзя записывать переменные с буквенными индексами, то перед этим оператором необходимо переименовать эту переменную в простую с помощью оператора ВЫЧИСЛИТЬ. В цикле кроме переменной x необходимо также изменять индекс i. Так как значения индекса изменяются одновременно с изменением переменной x, то закон его изменения указывается в том же операторе ПОВТОРИТЬ. С учетом сказанного программа имеет вид:

```
1. ВВОД — ALFA
   МАС — Y (100)
2. ВЫЧ — Y/I/ = SIN (ALFA. X) ; X
   ВЫЧ — M = Y / I /
   НАП — НА — БПМ — M
   ПОВ — 2 — I = 1 — (1) . X = 1 — (1) — (= 100
   КОН ( —
   НАЧ ( — 1
```

Пример. Вычислить значение функции $Z = \sum_{i=1}^{25} (\sin x_i + x_i^2)$, где x_i заданы массивом, результат отпечатать на БПМ.

Накопление суммы (см. задачу 3.8) осуществляется в цикле с помощью выражения $Z = Z + Y$, где $Y = \sin x_i + x_i^2$ — значение слагаемого.

Перед циклом необходимо очистить ячейку, отведенную для накопления суммы, т. е. вычислить $Z = 0$. Программа решения этой задачи имеет вид:

```
1. ВВОД — X (25)
   ВЫЧ — Z = 0
2. ВЫЧ — Y = SIN (X/I/) + X /I/▽2 — Z = Z + Y
   ПОВ — 2 — I = 1 — (1) — 25
   НАП — НА — БПМ — Z
   КОН —
   НАЧ — 1
```

Примеры иллюстрируют способ 1 организации цикла.

Пример. Вычислить сумму бесконечного ряда $S = 1 + x + \frac{x^2}{2!} + \frac{x^n}{n!} + \ldots$ с точностью ε (см. задачу 3.11).
Программа решения задачи имеет вид:

```
1. ВВОД — X, EPSILON
   ВЫЧ — S = 1 — H = 1 — N = 1
2. ВЫЧ — H = H. X : N — S = S + H — N = N + 1
   ЕСЛИ — H — ) = EPSILON — ТО — 2
   НАП — НА — БПМ — S
   КОН —
   НАЧ — 1
```

Здесь приняты следующие обозначения: S — текущее значение суммы; H — текущее значение члена ряда; N — номер члена ряда. Пример иллюстрирует способ 2 организации цикла.

Пример. Определить k, при котором $Z = x^{-2}/k$ становится меньше 1, где $k = 1, 2, 3, \ldots$.

Алгоритм решения этой задачи следующий.

1. Вычисляется Z при k, равном 1, 2, 3, и т. д.
2. После каждого вычисления проверяется выполнение условия $Z < 1$. Если условие выполняется, то необходимо выходить из цикла, в противном случае, необходимо изменить на единицу значение k и вернуться к началу цикла.

Поскольку букву k нельзя использовать для обозначения переменной, то заменим ее буквой N.

Программа решения задачи имеет вид:

```
1. ВВОД — X
2. ВЫЧ — Z = X ▽ (—2) : N
   ЕСЛИ — Z — (1 — ТО — 3
   ПОВ — 2 — N = 1 — (1)
3. НАП — НА — БПМ — N
   КОН —
   НАЧ — 1
```

Пример иллюстрирует способ 3 организации цикла.

Пример. Решить уравнение $\operatorname{arctg} x - x = 0$, используя метод итераций (последовательных приближений). Корень уравнения вычислить с точностью $1 \cdot 10^{-5}$. Начальное значение корня считать равным $x_0 = 4{,}7$.

Блок-схема алгоритма решения задачи приведена на рис. 10.7.

Обозначим предыдущее значение корня через X0, тогда следующее приближенное значение корня может быть определено с помощью выражения X1 = = ARCTG (X0).

Если разность между двумя смежными значениями корня $|X1 - X0| \leqslant 1 \cdot 10^{-5}$, то задача решена и надо выходить из цикла. В противном случае необходимо X1 считать предыдущим значением корня и переходить к вычислению нового значения X1.

Программа решения задачи имеет вид:

```
▽
△
1. ВЫЧ — X0 = 4, 7 ▽
                   △
2. ВЫЧ — X1 = ARCTG (X0) ▽
                         △
   ЕСЛИ — MOD (X1 — X0) — ( = Ю — 5 — ТО — 3 ▽
                                             △
   ВЫЧ — X0 = X1 ▽
                 △
   ПЕР — 2 ▽
           △
3. НАП — НА — БПМ — X1 ▽
                       △
   КОН — ▽
         △
   НАЧ — 1 ▽
           △
```

Пример иллюстрирует способ 4 организации цикла.

Принцип организации сложного цикла. Цикл (внешний), внутрь которого входит другой цикл (внутренний), называют *сложным циклом*. Правила организации как внешнего, так и внутреннего цикла те же, что и для простого цикла. В случаях, когда выход из внутреннего цикла, заканчивающегося оператором ПОВТОРИТЬ, осуществляется с помощью оператора ЕСЛИ, то параметром, изменяющимся в этом цикле, не может быть индекс.

В качестве примера сложного цикла рассмотрим задачу раскроя материала (см. задачу 3.2), блок-схема алгоритма решения которой представлена на рис. 10.39.

Программа решения этой задачи имеет вид:

```
▽
△
MAC — R (5) ▽
            △
1  ВЫЧ — MIN = Ю10 ▽
                   △
2. ВЫЧ — X = 500 — 0,75. Y — (7 : 12). Z — (1 : 3). U — V = 250 — 0,25.
   Y — 0,5. Z — 0,75. U ▽
                        △
ЕСЛИ — X — (0 — ТО — 8 ▽
                       △
ЕСЛИ — V — (0 — ТО — 8 ▽
                       △
ВЫЧ — C = (3:16). Y + 0,125. Z + (5 : 16) U + 0,75 V ▽
                                                    △
ЕСЛИ — C —) = MIN — ТО — 8 \
                           △
ВЫЧ — MIN = C — R / 1 / = X — R / 2 / = Y — R / 3 / = Z —
   R / 4 / = U — R / 5 / = V ▽
                             △
```

8. ПОВ — 2 — Y = 0 — (100) — (= 600) ▽△

ПОВ — 2 — Z = 0 — (100) — (= 500 ▽△

ПОВ — 2 — Z = 0 — (100) — (= 300▽△

НАП — НА — БПМ — MIN, R (5) ▽△

КОН — ▽△

НАЧ — 1 ▽△

Здесь для сохранения в памяти машины значений x, y, z, u, v, при которых был получен минимальный расход, в отличие от блок-схемы, выделен массив R. В этот массив записываются x, y, z, u, v всякий раз, как только получают C, меньше MIN.

Существует еще несколько операторов, которые не были рассмотрены:

1. Оператор КОД[ ]; с его помощью в автокодовую программу можно вводить участки программ, написанных в коде машины; 2. Группа корректировочных операторов ВСТАВИТЬ, ЗАМЕНИТЬ, УДАЛИТЬ; в них указываются номера строк программы, которые надо заменить или удалить, или после которых нужно вставить новые операторы. Строки с исправленными операторами записываются непосредственно за этими операторами.

ГЛАВА 8

АЛГОРИТМИЧЕСКИЙ ЯЗЫК КОБОЛ

Основным требованием, выдвинутым при разработке языка КОБОЛ, было облегчение описания разнообразных коммерческих, финансовых, учетно-статистических и других экономических задач с использованием терминов и логики, обычно применяемых в обработке экономических данных без помощи ЦВМ. Поэтому этот язык получил широкое распространение как средство привлечения к решению задач экономики лиц, не являющихся профессиональными программистами, но хорошо представляющих сущность финансовых или экономических задач. Очень важным свойством КОБОЛа служит то, что он в значительной мере «машинно-независим», в результате чего отлаженные КОБОЛ-программы можно с очень незначительными переделками использовать для других ЦВМ и осуществлять обмен программами между различными организациями. Необходимо также отметить, что использование КОБОЛа позволяет автоматизировать не только решение задач, но и ведение документации по задачам, что особенно важно в области экономики.

Исходная КОБОЛ-программа перфорируется на перфоленте или перфокартах, вводится в ЦВМ и посредством программы-транслятора переводится на язык машинных команд. Результатом трансляции является рабочая программа, которую и выполняет ЦВМ. В задачу программиста входит описать алгоритм решения своей задачи в виде исходной КОБОЛ-программы.

Для обработки экономических данных характерно многократное повторение однотипных, сравнительно несложных арифметических операций над последовательными группами данных большого объема. Вводимые данные представляют собой уже существующие *массивы* (*файлы*) определенной структуры, хранимые на магнитной ленте или перфокартах и содержащие, кроме числовых, различные текстовые данные (фамилии, адреса, названия предприятий и т. д.). Каждый массив данных, записанных на магнитной ленте, состоит из некоторого известного числа *записей*, которым, как правило, предшествуют *метки массива*, позволяющие отличить один массив от другого. Записи в массиве могут представлять собой строки ведомостей, наряды на работу, записи в балансах, квитанции и т. д. В свою очередь, записи состоят из отдельных *единиц данных* (*пунктов*), например, порядковый номер, фамилия, наименование, цена, количество, сумма, причем каждая единица имеет определенный формат. В процессе выполнения программы записи по одной считываются с магнитной ленты, помещаются в оперативную память и обрабатываются, после чего осуществляется вывод на печать, магнитную ленту или перфокарты записей, образующих выходной массив.

Набор символов КОБОЛа. Состав символов, входящих в набор (алфавит языка), определяется разработчиком, проектирующим программу-транслятор для определенной ЦВМ. В большинстве реализаций языка используют три основных типа символов: цифровые — арабские цифры от 0 до 9; алфавитные — латинские заглавные буквы или русские заглавные буквы (как, например, в реализации языка КОБОЛ в системе автоматической обработки данных (САОД) для ЦВМ Минск-22М и Минск-32); специальные символы, типовой набор которых приведен в табл. 8.1.

Таблица 8.1

Название	Символ	Название	Символ
Пробел	—	Точка с запятой	;
Знак плюс	+	Точка	.
Знак минус	–	Запятая	,
Дефис	–	Знак равенства	=
Знак умножения	*	Знак «больше»	>
Знак деления	/	Знак «меньше»	<
Левая скобка	(	Кавычка	"
Правая скобка	)		

Набор символов, приведенный в таблице, является общим для большинства реализаций языка. Далее при изложении будем ориентироваться на набор символов и ограничения, принятые при реализации КОБОЛа в САОД.

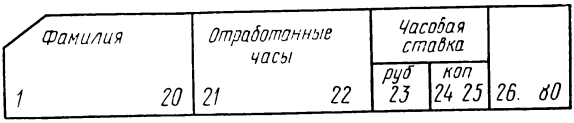
Структура КОБОЛ-программы. Как указывалось, использование ЦВМ для экономических расчетов связано со считыванием большого числа данных с входных носителей, их преобразованием в ЦВМ (арифметические операции, преобразование форматов) и выдачей результатов либо в виде записей на выходной носитель, либо в виде печатного документа. Структура КОБОЛ-программы отражает стремление облегчить описание такого рода обработки, при этом особое внимание уделяется проблемам представления массивов входных и выходных данных.

Описание процесса обработки включает в себя название процесса, описания оборудования, структуры исходных, промежуточных и получающихся в результате обработки данных, последовательности обработки. В соответствии с этим в исходной КОБОЛ-программе различаются четыре раздела, записываемых в следующем порядке:

РАЗДЕЛ ИДЕНТИФИКАЦИИ
РАЗДЕЛ ОБОРУДОВАНИЯ
РАЗДЕЛ ДАННЫХ
РАЗДЕЛ ПРОЦЕДУР.

Пример. Рассмотрим упрощенную программу решения следующей задачи. Автоматизируется ведение документации по расчетам заработной платы для внештатных сотрудников (почасовиков) некоторой организации. Для каждого

из них на перфокарте записаны сведения о числе часов, отработанных за месяц, и часовой ставке в таком формате:



Результаты расчета месячной заработной платы каждого сотрудника должны быть занесены на магнитную ленту в виде записей, содержащих данные входной перфокарты в том же формате плюс графа «Сумма за месяц», в которой две позиции отведено под рубли и две под копейки. Кроме того, надо подсчитать и напечатать общую сумму выплат всем внештатным сотрудникам за месяц.

Упрощенная КОБОЛ-программа, решающая эту задачу, будет следующей:

```
010 РАЗДЕЛ ИДЕНТИФИКАЦИИ.
020 ПРОГРАММА. РАСЧЕТ ЗАРПЛАТЫ.
030 РАЗДЕЛ ОБОРУДОВАНИЯ.
040 РАБОЧАЯ — МАШИНА. МИНСК-32.
050 УПРАВЛЕНИЕ — МАССИВАМИ. ДЛЯ МАС — ВХ — ДОКУМ ПРЕД-
    НАЗНАЧИТЬ ВК.
060 ДЛЯ МАС — ВЫХ — ДОКУМ ПРЕДНАЗНАЧИТЬ МЛ1.
070 РАЗДЕЛ ДАННЫХ.
080 ОМ МАС — ВХ — ДОКУМ МЕТКИ ОПУЩЕНЫ.
090 01 ВХ — ДОКУМ.
100     02 ФАМИЛИЯ                ШАБЛОН А (20).
110     02 ОТРАБ — ЧАСЫ           ШАБЛОН 9 (2).
120     02 ЧАСОВАЯ — СТАВКА       ШАБЛОН 9Т99.
130 ОМ МАС — ВЫХ — ДОКУМ МЕТКИ    ОПУЩЕНЫ.
140 01 ВЫХ — ДОКУМ.
150     02 ФАМ — ВЫХ              ШАБЛОН А (20).
160     02 ЧАСЫ — ВЫХ             ШАБЛОН 9 (2).
170     02 ЧАС — СТАВКА — ВЫХ     ШАБЛОН 9Т99.
180     02 СУММА — ЗА — МЕСЯЦ     ШАБЛОН 9 (3)Т99.
190 77 ОБЩ — СУМ ШАБЛОН 9 (5) Т99 ЗНАЧЕНИЕ НУЛЬ.
200 РАЗДЕЛ ПРОЦЕДУР.
210 П1. ЧИТАТЬ МАС — ВХ — ДОКУМ В КОНЦЕ ПЕРЕЙТИ К П2.
220     ПОМЕСТИТЬ ФАМИЛИЯ В ФАМ — ВЫХ. ПОМЕСТИТЬ ОТРАБ —
        ЧАСЫ.
230     В ЧАСЫ — ВЫХ. ПОМЕСТИТЬ ЧАСОВАЯ — СТАВКА В ЧАС —
        СТАВКА — ВЫХ.
240     ВЫЧИСЛИТЬ СУММА — ЗА — МЕСЯЦ = ЧАСОВАЯ — СТАВКА *
        ОТРАБ — ЧАСЫ.
250     ВЫЧИСЛИТЬ ОБЩ — СУМ = ОБЩ — СУМ + СУММА — ЗА —
        МЕСЯЦ.
260     ПИСАТЬ ВЫХ — ДОКУМ. ПЕРЕЙТИ К П1.
270 П2. ВЫДАТЬ ОБЩ — СУМ. КОНЕЦ — ПРОГРАММЫ.
```

В РАЗДЕЛЕ ИДЕНТИФИКАЦИИ дается название программы, а также может быть указана другая справочная информация, используемая для документации.

РАЗДЕЛ ОБОРУДОВАНИЯ содержит сведения об используемой ЦВМ и устройствах ввода — вывода и ориентирован на конкретную ЦВМ и именно он обычно изменяется при переходе на другую машину.

Информация в строке 050 сообщает ЦВМ, что массив данных, названный МАС — ВХ — ДОКУМ, следует вводить, используя устройство ввода с перфокарт (сокращенное обозначение этого устройства в САОД — ВК).

Информация строки 060 состоит в том, что массив, полученный в результате обработки информации (МАС — ВХ — ДОКУМ), следует записывать на магнитную ленту МЛ1.

РАЗДЕЛ ДАННЫХ сообщает ЦВМ, в каком виде появляются указанные выше массивы.

Кодовый символ ОМ (описание массива) в строке 080 обозначает начало статьи описания входного массива и предшествует названию массива.

Кодовый символ 01 в строке 090 перед названием ВХ — ДОКУМ указывает ЦВМ, что внутри массива МАС — ВХ — ДОКУМ надо найти записи, называемые ВХ — ДОКУМ.

Символы 02 в строках 100, 110, 120 указывают, что названные в них единицы данных ФАМИЛИЯ, ОТРАБ — ЧАСЫ, ЧАСОВАЯ — СТАВКА находятся внутри записи, называемой ВХ — ДОКУМ. За каждым названием данных идет фраза ШАБЛОН, указывающая ЦВМ, сколько и какие символы содержит каждое поле описываемых данных. В этом описании символ А означает любую букву или пробел, а цифра 20 в скобках говорит о том, что поле, отведенное для фамилии, занимает 20 позиций. Символ 9 соответствует любой десятичной цифре, а символ Т указывает на положение десятичной точки. Аналогично описан выходной массив, названный МАС — ВЫХ — ДОКУМ.

В строке 190 величина, названная ОБЩ — СУМ, имеет символ 77, который указывает на то, что эта величина не является массивом или элементом записи, а существует отдельно от других элементов данных. Она служит для подсчета общей суммы выплат за месяц. Перед началом вычислений (до обработки первой записи) значение этой величины должно быть равно нулю. Это обеспечивается фразой ЗНАЧЕНИЕ НУЛЬ в строке 190.

РАЗДЕЛ ПРОЦЕДУР в данном случае разделен на два параграфа П1 и П2. Информация в строке 210 (параграф П1) заставляет ЦВМ читать запись из массива входных документов (с помощью устройства ввода с перфокарт) с целью ее обработки в соответствии с инструкциями строк 220 ÷ 260. Записи читаются и обрабатываются последовательно одна за другой. Если в массиве не содержится больше записей, ЦВМ должна перейти к строке 270 (параграфу П2) для выполнения содержащихся в ней инструкций. Строки 220 ÷ 240 определяют процедуру получения выходного документа. При этом некоторые данные (ФАМ — ВЫХ, ЧАСЫ — ВЫХ, ЧАС — СТАВКА — ВЫХ) получаются за счет переписи информации из соответствующих полей входного документа, а другие данные (СУММА — ЗА — МЕСЯЦ) — в результате вычислений. ОБЩ — СУМ увеличивается при обработке документа, относящегося к следующему сотруднику, на величину суммы, которая должна быть выплачена этому сотруднику. После вывода на магнитную ленту обработанной записи ЦВМ возвращается к началу параграфа П1 для считывания следующей записи.

К параграфу П2 ЦВМ переходит, закончив обработку последней записи, содержащейся во входном массиве. В соответствии с инструкцией этого параграфа ЦВМ печатает общую сумму выплат и останавливается.

КОБОЛ-программа, записанная на бумаге или на специальном бланке, перфорируется затем на перфокарты (каждая строка на отдельную карту). Колонки 1÷6 карты содержат номер страницы и строки программы, что нужно в основном для удобств программиста. Номера строк могут идти не подряд, но обязательно в возрастающем порядке. Поэтому их обычно нумеруют так: 010, 020 и т. д., что допускает возможность внесения вставок. Запись операторов КОБОЛа начинается с колонки 8 (граница А) или с колонки 12 (граница В), правила этой записи оговариваются для каждого оператора.

КОБОЛ-программа состоит из *разделов*, которые в свою очередь подразделяются на *секции* и *параграфы*. Каждая секция и каждый параграф в разделах идентификации, оборудования и данных имеют строго определенное название и назначение. Структурой раздела процедур в значительной мере управляет программист, который сам

определяет объединение групп процедур в секции и параграфы и присваивает им названия. Все конструкции языка КОБОЛ строятся из *слов*, образованных из символов языка по определенным правилам. При этом некоторые слова в языке являются фиксированными и могут употребляться только в определенном, специальном смысле (например, ЗНАЧЕНИЕ, МЕТКИ, ЧИТАТЬ, НУЛЬ, ПОМЕСТИТЬ). Эти слова называют *резервированными*. Все остальные слова, называемые *словами пользователя*, представляют собой информацию, которую дает составитель программы. В основном это названия отдельных элементов данных, процедур и т. д. Правила употребления резервированных слов и записи слов пользователя указываются при рассмотрении конкретных разделов КОБОЛ-программы.

Далее при описании форматов каждого раздела используют следующую систему обозначений: резервированные слова записываются прописными буквами; слова, написанные строчными буквами, заменяются словами пользователя.

В КОБОЛ-программе все, что заключается в квадратные скобки, необязательно и может быть опущено. Фигурные скобки означают, что может быть выбрана любая из представленных в скобках альтернатив.

Приведенный ниже материал не является полным описанием языка КОБОЛ, но его достаточно для чтения, понимания и составления программ на этом языке.

§ 8.2. РАЗДЕЛЫ ИДЕНТИФИКАЦИИ И ОБОРУДОВАНИЯ

Раздел идентификации. Этот раздел используют для сообщения некоторой справочной информации о программе. Формат раздела следующий:

```
РАЗДЕЛ ИДЕНТИФИКАЦИИ.
ПРОГРАММА. Название программы.
[АВТОР. Предложение.]
[ДАТА — НАПИСАНИЯ. Предложение.]
[ЗАМЕЧАНИЯ. Предложение.]
```

В разделе идентификации только параграф ПРОГРАММА является обязательным и записывается всегда первым. Остальные параграфы, если они присутствуют, следуют за ним в любом порядке. Типичный раздел идентификации может иметь такой вид:

```
РАЗДЕЛ ИДЕНТИФИКАЦИИ.
ПРОГРАММА. РАСЧ — ЗАРПЛ.
АВТОР. ВАСИЛЬЕВ И. П.
ДАТА — НАПИСАНИЯ. ОКТ 72.
ЗАМЕЧАНИЯ. ПРОГРАММА СОСТАВЛЯЕТ ВЕДОМОСТЬ НА ВЫДАЧУ
ЗАРПЛАТЫ ОСНОВНЫМ СОТРУДНИКАМ.
```

Раздел оборудования. Этот раздел описывает состав используемого для решения задачи оборудования (ЦВМ, устройства ввода—вывода) и сопоставляет конкретные устройства используемым в задаче входным и выходным массивам. Раздел оборудования состоит из

секций конфигурации и ввода — вывода и имеет следующую структуру:

```
РАЗДЕЛ ОБОРУДОВАНИЯ.
[СЕКЦИЯ КОНФИГУРАЦИИ.
РАБОЧАЯ — МАШИНА. ...]
[СЕКЦИЯ ВВОДА — ВЫВОДА.
УПРАВЛЕНИЕ — МАССИВАМИ. ...
[УПРАВЛЕНИЕ — ВВОДОМ — ВЫВОДОМ. ...]]
```

(многоточие стоит вместо содержимого параграфов).

Секции конфигурации и ввода—вывода необязательны в КОБОЛ-программе.

Параграф УПРАВЛЕНИЕ—МАССИВАМИ с помощью фразы ДЛЯ...ПРЕДНАЗНАЧИТЬ... описывает, посредством каких устройств будут вводиться или выводиться определенные массивы, участвующие в задаче. Состав устройств и их обозначения в каждой реализации языка свои. Так, в САОД используют обозначения:

ВЛ	устройство ввода с перфоленты
ВК	устройство ввода с перфокарт
ЫШ	устройство вывода на широкую печать
ЫЛ 1	выходной ленточный перфоратор 1
ЫЛ 2	выходной ленточный перфоратор 2
ЫК	выходной перфоратор карт
МЛ целое (от 2 до 15)	магнитная лента

Например,

```
УПРАВЛЕНИЕ — МАССИВАМИ.
    ДЛЯ ПЛАТЕЖ — БАЛАНС ПРЕДНАЗНАЧИТЬ МЛ1.
    ДЛЯ ПОСТУПЛЕНИЯ ПРЕДНАЗНАЧИТЬ МЛ4.
    ДЛЯ ПЛАТЕЖИ ПРЕДНАЗНАЧИТЬ МЛ2.
    ДЛЯ ИТОГИ ПРЕДНАЗНАЧИТЬ ЫШ.
```

Параграф УПРАВЛЕНИЕ—ВВОДОМ—ВЫВОДОМ с помощью фразы ОБЩАЯ ЗОНА ДЛЯ... позволяет более эффективно использовать память при выполнении рабочей программы.

Пример раздела оборудования.

```
РАЗДЕЛ ОБОРУДОВАНИЯ.
СЕКЦИЯ КОНФИГУРАЦИИ.
РАБОЧАЯ — МАШИНА. МИНСК-22М.
СЕКЦИЯ ВВОДА — ВЫВОДА.
УПРАВЛЕНИЕ — МАССИВАМИ.
    ДЛЯ КРЕДИТНЫЕ — ССУДЫ ПРЕДНАЗНАЧИТЬ МЛ2.
    ДЛЯ ПЛАН — ПОГАШЕНИЯ ПРЕДНАЗНАЧИТЬ МЛ4.
```

§ 8.3. ОРГАНИЗАЦИЯ ДАННЫХ В ЯЗЫКЕ КОБОЛ

Одно из наиболее важных мест в КОБОЛе занимают средства описания данных, в частности, данных сложной иерархической структуры.

Прежде чем давать формат записи раздела данных, следует пояснить структуру данных, которые описывает КОБОЛ-программа.

В качестве основного входного и выходного объекта для системы обработки данных, использующей язык КОБОЛ, выступает *массив* (*файл*), представляющий собой совокупность родственных по содержанию данных. Примерами массивов являются: платежная ведомость, закодированная на перфокартах; множество счетов, записанных на магнитной ленте; совокупность анкетных данных о сотрудниках и т. д. Массив в КОБОЛе соответствует папке с документами в обычном, неавтоматизированном делопроизводстве.

Каждый массив состоит из *записей*, которые соответствуют отдельным документам или отдельным строкам большого документа. Например, запись могут составлять анкетные данные одного человека, информация одной строки платежной ведомости и т. д. Массивы могут состоять также и из неоднородных записей, например, документов различной формы.

Каждая запись в свою очередь подразделяется на отдельные *единицы* (пункты) данных, каждый из которых занимает определенное *поле*. В КОБОЛе различают единицы данных, содержащие в себе более мелкие подразделения — *групповые единицы данных*, и единицы, не содержащие подразделений, — *элементарные единицы данных*. Например, групповая единица данных АДРЕС может подразделяться на элементарные единицы ГОРОД, УЛИЦА, ДОМ, КОРПУС, КВАРТИРА.

Таким образом, иерархия данных в КОБОЛе строится так: массивы содержат записи, записи состоят из единиц данных, которые описываются как групповые, если они далее подразделяются, и как элементарные, если не подразделяются. Массивы, записи, групповые и элементарные единицы данных являются в КОБОЛе группировками данных, которым присваиваются имена-данных. *Имя-данного* — обозначение, даваемое программистом для определенной группировки данных. При выборе имени-данного необходимо учитывать, что оно 1) может включать в себя от 1 до 30 символов; 2) может содержать только буквы, цифры и дефисы; 3) не может начинаться или кончаться дефисом; 4) должно содержать, по крайней мере, одну букву; 5) не может быть резервированным словом КОБОЛа.

В табл. 8.2 приводятся примеры ошибочных имен-данных.

Типы данных. При описании в разделе данных массивов, записей и полей для них резервируют место в памяти. Содержимое же полей

Таблица 8.2

Имя-данного	Ошибка	Следует писать
ФИО СОТРУДНИКА	Присутствует пробел	ФИО—СОТРУДНИКА
МАС—'ПЛАТЕЖН—	Присутствует кавычка	МАС—ПЛАТЕЖН—
БАЛАНС'		БАЛАНС
БЛОК	Резервированное слово	БЛОК—Х
ПОРЯД—НОМ.	Присутствует точка	ПОРЯД—НОМ

получают в результате ввода или вычислений. Это так называемые *переменные данные*. Формой данных, не зависящих от ввода или вычислений, являются *константы*, которые могут быть определены прямо в разделе процедур. Различают следующие типы констант.

Числовые литералы — константы, определяемые в разделе процедур и используемые для арифметических операций. Например:

2.98; —876.04; +0.65; 978; 3.1415926.

Нечисловые литералы — константы, используемые в разделе процедур для всех операций, кроме арифметических. Эти константы могут содержать любые символы алфавита, кроме кавычек, и должны быть заключены в кавычки. Например, набор символов 'ОШИБКА В ЗАПИСИ' рассматривается в КОБОЛе как константа и может использоваться для выдачи на печать сообщения о том, что какая-то из записей массива содержит ошибку. При печати кавычки опускаются. Примерами нечисловых литералов являются: 'АБВ 123' '1,000' 'СООБЩЕНИЕ' 'ДАТА СОСТАВЛЕНИЯ ПРОГРАММЫ (ДЕНЬ, МЕСЯЦ, ГОД)'.

Фигуративные константы — резервированные слова КОБОЛа, имеющие определенное значение. В языке допускаются следующие фигуративные константы: НУЛЬ — представляет величину нуль или последовательность нулей; ПРОБЕЛ — представляет последовательность пробелов; КАВЫЧКА — представляет последовательность кавычек; ВСЕ нечисловой-литерал — последовательность символов, указанных односимвольным литералом.

Например, ВСЕ * — последовательность звездочек (длина ее определяется размером поля, заполняемого этой константой).

Литералы и фигуративные константы могут использоваться в разделе данных для указания значений *именуемых констант*, т. е. констант, которым присваивается имя в разделе данных.

Единицей ввода в ЦВМ является запись. Для размещения ее в оперативной памяти выделяется *зона ввода*, соответствующая размеру записи. Все массивы, участвующие в задаче, могут обрабатываться только последовательно — запись за записью. Для обработки каждая очередная запись переносится из зоны ввода в специальную зону обработки записи — *зону записи*. Иногда используется группировка записей на машинном носителе (магнитной ленте, перфоленте и т. д.) в *блоки записей*. В этом случае физической порцией ввода массива с внешних носителей в оперативную память является блок, а зона ввода равна размеру блока. Для отыскания определенного массива на внешнем носителе и указания его границ используются *метки*. Размещение информации на магнитной ленте может выглядеть так:

Метка начала ленты	*Метка начала массива*	*Записи*	*Метка конца массива*	*Метка начала массива*	*Записи*	*Метка конца массива*

В разделе данных программист описывает физические характеристики каждого массива, участвующего в обработке: 1) метод представления массива на внешнем носителе; 2) группирование записей в блоки; 3) средства идентификации массива (метки).

Кроме физических характеристик массива, в разделе данных дается детальное описание вида и структуры данных, входящих в состав записей массива, а также отдельных переменных и именуемых констант.

Структура раздела данных. Описание обрабатываемых входных и выходных массивов, а также промежуточных результатов и констант дается в разделе данных с помощью *статей описания*, состоящих в свою очередь из *фраз описания*.

Раздел данных состоит из *секции массивов* и *секции рабочей памяти*.

Секция массивов состоит из статей описания массивов, за каждой из которых следуют статьи описания записей, входящих в массив.

В секции рабочей памяти описываются области оперативной памяти, в которых хранятся промежуточные результаты и константы во время выполнения программы. Секция рабочей памяти может быть опущена, если надобность в рабочих областях памяти не возникает.

Секция массивов. Эта секция имеет следующую структуру:

СЕКЦИЯ МАССИВОВ.
статья описания массива—1.
статьи описания записи 1-го типа массива—1.
. . . . . .
статьи описания записи k-го типа массива—1.
статья описания массива—2.
статьи описания записи 1-го типа массива—2.
. . . . . .

Статья описания массива имеет формат:

ОМ имя — массива [МЕТОД ...] [В БЛОКЕ ... ЗАПИСЕЙ]

МЕТКИ { СТАНДАРТНЫ, ШИФР МАССИВА / ОПУЩЕНЫ / имя — данного }

Сочетание ОМ является условным обозначением начала статьи описания массива.

Фраза МЕТОД ... используется для указания способа представления записи на носителе, в частности, способа кодирования информации. Например, в САОД МЕТОД РМ2ПЛ означает, что массив подготовлен на перфоленте в международном телеграфном коде МТК-2 и данные отделяются одно от другого специальным символом (разделителем). Если фраза опущена, то это означает, что применен стандартный метод представления информации для конкретного носителя.

Вместо многоточия в фразе в БЛОКЕ ... ЗАПИСЕЙ должно стоять целое число. Эта фраза обеспечивает более эффективный ввод—

вывод данных, при котором обмен информацией между оперативной памятью и внешними носителями осуществляется целыми блоками. Продуманное формирование записей в блоки позволяет сократить число обращений к ленте, что ускоряет выполнение программы. Следует учитывать при этом, что в случае слишком больших блоков рабочая программа может не уместиться в оперативную память. Если фраза отсутствует, то размер блока принимается равным размеру наибольшей записи.

Метки служат для идентификации массивов на внешних носителях. Фраза МЕТКИ СТАНДАРТНЫ обеспечивает при выполнении рабочей программы проверку меток для входных массивов и снабжения метками выходных массивов. Формат меток зависит от типа машины и детально описывается в инструкциях, прилагаемых к машине. Фраза МЕТКИ ОПУЩЕНЫ используется для массивов на устройствах единичных записей или магнитных лент без меток. Для выводимых массивов тип записи меток может быть указан самим программистом. Это осуществляется определенным набором инструкций в разделе процедур, которому присвоено имя, указанное во фразе МЕТКИ имя данных. Примерами статей описания массива являются:

```
ОМ ИНВЕНТ — ВЕДОМОСТЬ В БЛОКЕ 5 ЗАПИСЕЙ МЕТКИ СТАНДАРТНЫ
    ШИФР МАССИВА ИВ1.
ОМ ПЛАТЕЖНЫЙ — БАЛАНС МЕТОД РМ2ПЛ МЕТКИ ОПУЩЕНЫ.
ОМ КВАРТАЛЬНЫЙ — ОТЧЕТ МЕТКИ ПОЛУЧ — СПЕЦ — МЕТОК.
```

В последнем случае в разделе процедур должен быть параграф, имеющий название ПОЛУЧ—СПЕЦ—МЕТОК, в котором описана структура и содержание меток к массиву КВАРТАЛЬНЫЙ—ОТЧЕТ.

Статьи описания записей каждого массива следуют непосредственно за статьей описания данного массива. Каждая единица данных (элементарная или групповая) должна быть описана в отдельной статье.

Иерархия данных в записи определяется *номерами уровня*, обозначаемыми целыми числами, стоящими перед названиями данных. Номера уровня присваиваются отдельным единицам данных по определенным правилам.

Высший уровень иерархии данных внутри массива (запись) обозначается числом 01. Любая единица данных, входящая в состав другой единицы, имеет соответственно численно больший номер уровня.

Пример. Документ «Список студентов» учебного заведения может иметь такую структуру:

Номер п.п	Личные данные									Факультет	Группа
	Ф	И	О	Дата рождения			Домашний адрес				
				Число	Месяц	Год	Город	Улица	Дом	Квартира	

и храниться в виде массива на магнитной ленте. Каждая строка этого документа соответствует одной записи. В разделе данных КОБОЛ-программы такой массив может быть описан следующим образом:

```
ОМ СПИСОК — СТУДЕНТОВ МЕТКИ ОПУЩЕНЫ.
01 СТРОКА — СПИСКА.
  02 ПОР — НОМЕР ШАБЛОН 9 (4).
  02 ЛИЧНЫЕ — ДАННЫЕ.
    03 ФАМИЛИЯ ШАБЛОН А (20).
    03 ИМЯ ШАБЛОН А (20).
    03 ОТЧЕСТВО ШАБЛОН А (20).
    03 ДАТА — РОЖДЕНИЯ.
      04 ЧИСЛО ШАБЛОН 99.
      04 МЕСЯЦ ШАБЛОН ААА.
      04 ГОД ШАБЛОН 9 (4).
    03 ДОМ — АДРЕС.
      04 ГОРОД ШАБЛОН А (20).
      04 УЛИЦА ШАБЛОН Х (30).
      04 ДОМ ШАБЛОН 9 (3).
      04 КВАРТИРА ШАБЛОН 9 (3).
  02 ФАКУЛЬТЕТ ШАБЛОН А (20).
  02 ГРУППА ШАБЛОН Х (10).
```

Из приведенного примера видно, что описание элементарных единиц данных заканчивается фразой ШАБЛОН ... . С помощью этой фразы дается полная характеристика каждой единицы данных: сколько символов эта единица содержит, какие это могут быть символы (буквы, цифры и т. д.), где должна находиться десятичная точка, если это единица числовых данных, и т. п. Буква А в записи шаблона соответствует любой букве или пробелу. Число в скобках показывает, сколько позиций отведено под символы. Таким образом, ФАМИЛИЯ, ИМЯ и ОТЧЕСТВО в записи занимают по 20 буквенных позиций, МЕСЯЦ занимает три буквенных позиции, т. е. используют, например, сокращения ОКТ, ДЕК, МАР. Цифра 9 в записи шаблона соответствует любой цифре от 0 до 9, буква Х соответствует любому символу языка.

Кроме буквенного, буквенно-цифрового и цифрового видов шаблонов в КОБОЛе существует шаблон отчета, служащий для редактирования выходных данных при выдаче их на печать с возможностью включения точек и запятых, подавления нулей в старших разрядах и т. д., а также шаблон с плавающей десятичной точкой.

При описании записей вместо имени данного может использоваться слово ЗАПОЛНИТЕЛЬ, если в разделе процедур нет ссылок на эту единицу данных. Это экономит использование памяти, так как программа-транслятор составляет и хранит в памяти таблицу всех имен данных. Например, программа обрабатывает документы, содержащие в числе прочих данных групповую единицу данных ДАТА, подразделяющуюся на ДЕНЬ, МЕСЯЦ, ГОД. При этом раздел процедур не использует имени данного ГОД. Тогда эта единица данных может быть описана следующим образом:

```
02 ДАТА.
  03 ЗАПОЛНИТЕЛЬ ШАБЛОН 9 (4).
  03 МЕСЯЦ ШАБЛОН А (3).
  03 ДЕНЬ ШАБЛОН 99.
```

При использовании слова ЗАПОЛНИТЕЛЬ в описании выходного массива можно управлять форматом выдачи, например, описывать интервал между графами таблицы:

```
01 СПИСОК.
   02 ФИО ШАБЛОН А (20).
   02 ЗАПОЛНИТЕЛЬ ШАБЛОН А (20) ЗНАЧЕНИЕ ПРОБЕЛ.
   02 ВОЗРАСТ ШАБЛОН 99.
```

В выходном документе между графами ФИО и ВОЗРАСТ обеспечивается интервал в 20 пробелов.

Фраза ЗНАЧЕНИЕ ... может употребляться в КОБОЛ-программе для указания начальных значений отдельных полей данных, чаще в секции рабочей памяти. В качестве возможных значений данных могут использоваться литералы или фигуративные константы. Фразы ШАБЛОН и ЗНАЧЕНИЕ должны быть непротиворечивы. Например,

```
02 ЗАРПЛАТА ШАБЛОН 9 (3) Т99 ЗНАЧЕНИЕ 120.50.
```

В поле ЗАРПЛАТА будет занесено число 12050, интерпретируемое при обращении к нему как 120.50 (буква Т в описании шаблона соответствует подразумеваемой десятичной точке);

```
03 ПРЕДПРИЯТИЕ ШАБЛОН Х (10) ЗНАЧЕНИЕ 'ФИРМА ЗАРЯ'.
```

В начале выполнения рабочей программы комбинация букв ФИРМА ЗАРЯ будет помещена в поле ПРЕДПРИЯТИЕ.

Имеется также ряд фраз, которые не являются обязательными при описании записей:

Фраза ПОВТОРЯЕТСЯ ... может использоваться на уровне группового данного (кроме уровня 01), указывая, что описание применимо к повторяющейся группе данных. Это сокращает описание однородных данных. Например, для описания записи (не массива!) ВЕДОМОСТЬ, состоящей из 20 одинаковых строк, содержащих единицы данных ТАБ—НОМЕР, ФИО, ОКЛАД, можно использовать такой способ:

```
01 ВЕДОМОСТЬ.
   02 СТРОКА — ВЕД ПОВТОРЯЕТСЯ 20 РАЗ.
      03 ТАБ — НОМЕР ШАБЛОН 9 (4).
      03 ФИО ШАБЛОН А (12).
      03 ОКЛАД ШАБЛОН П99.99.
```

Вид шаблона в последней строке используется при выводе данных на печать. Буква П означает, что если в этом разделе стоит нуль, то при выводе он не напечатается. Точка (вместо Т) означает, что при хранении данного в оперативной памяти точка не подразумевается, а занимает отдельный разряд. В качестве возможных значений данного ОКЛАД при выводе их на печать могут быть, например, числа 175.00; 78.50; 60.00 (в последних двух значениях отсутствует нуль в старшем разряде).

Фраза { ДЛЯ ВЫЧИСЛЕНИЙ / ДЛЯ ВЫВОДА } влияет на форму внутреннего представления данных (в оперативной памяти).

Если фраза опущена, то предполагается использование ДЛЯ ВЫВОДА. При этом над данными нельзя производить арифметических операций. В случае же их выполнения данные необходимо переслать в поле вычислительного данного.

Если фраза написана на уровне группового данного, то она относится ко всем элементарным единицам, входящим в состав этого группового данного. Например,

```
02 ФАМИЛИЯ ШАБЛОН А (12).
02 ВЫРАБОТКА ДЛЯ ВЫЧИСЛЕНИЙ.
   03 ЦЕНА — ИЗД ШАБЛОН 9Т99.
   03 КОЛИЧ — ИЗД ШАБЛОН 9 (4).
```

Секция рабочей памяти. Эта секция используется для описания полей, в которых во время выполнения рабочей программы хранятся промежуточные результаты или константы, которым присвоены имена. Поля, описываемые в секции рабочей памяти, могут быть связанные и несвязанные. *Связанные поля* образуют запись, описываемую в той же форме, как и в секции массивов. *Несвязанные поля* — одиночные поля, не имеющие подразделений и сами не являющиеся подразделениями других полей. Для их описания используется номер уровня 77.

Пример раздела данных.

```
РАЗДЕЛ ДАННЫХ.
СЕКЦИЯ МАССИВОВ.
ОМ РАСЦЕНКИ МЕТКИ СТАНДАРТНЫ ШИФР Р1.
01 ТАБЛ — РАСЦЕНОК.
   02 СТРОКА — ТАБЛИЦЫ ПОВТОРЯЕТСЯ 15 РАЗ.
      03 НАЗВ — РАБ ШАБЛОН Х (10).
      03 ЦЕНА — ИЗД ШАБЛОН 99Т99 ДЛЯ ВЫЧИСЛЕНИЙ.
ОМ НАРЯДЫ.
01 ЗАПИСЬ — НАРЯДА.
   02 ФИО ШАБЛОН А (12).
   02 НАЗВ — РАБ — 1 ШАБЛОН Х (10).
   02 ВЫРАБОТКА ШАБЛОН 9 (4) ДЛЯ ВЫЧИСЛЕНИЙ.
ОМ ПЛАТЕЖНАЯ — ВЕДОМОСТЬ МЕТКИ ОПУЩЕНЫ.
01 СТРОКА — ВЕД.
   02 ФИО — ВЫХ ШАБЛОНА (12).
   02 НАЧИСЛЕНИЯ.
      03 НАЗВ — РАБ — ВЫХ ШАБЛОН Х (10).
      03 ВЫРАБ — ВЫХ ШАБЛОН 9 (4) ДЛЯ ВЫЧИСЛЕНИЙ.
      03 СУММА ШАБЛОН П (4). 99 ДЛЯ ВЫЧИСЛЕНИЙ.
   02 УДЕРЖАНИЯ ДЛЯ ВЫЧИСЛЕНИЙ.
      03 АВАНС ШАБЛОН П (3) Т99.
      03 КРЕДИТ ШАБЛОН П (3) Т99.
      03 НАЛОГ ШАБЛОН П (3) Т99.
   02 К — ВЫДАЧЕ ШАБЛОН П (4).99.
СЕКЦИЯ РАБОЧЕЙ — ПАМЯТИ.
77 ИТОГ ШАБЛОН 9 (6). 99 ЗНАЧЕНИЕ НУЛЬ.
```

§ 8.4. РАЗДЕЛ ПРОЦЕДУР

В этом разделе описываются операции, выполняемые в процессе обработки данных: ввод и вывод записей массива, арифметические действия над величинами, перемещение данных из одних документов

в другие, логические действия над данными (сравнение,выбор данных, обладающих определенными признаками, и т. д.). Основу раздела процедур составляют *глаголы*, определяющие действия, которые необходимо выполнить. Глаголы делятся на несколько основных групп.

Арифметический глагол ВЫЧИСЛИТЬ (в некоторых версиях языка используются еще глаголы СЛОЖИТЬ, ВЫЧЕСТЬ, УМНОЖИТЬ РАЗДЕЛИТЬ).

Глаголы перемещения данных ПОМЕСТИТЬ, ПРОСМОТРЕТЬ.

Глаголы ввода — вывода данных ОТКРЫТЬ, ЧИТАТЬ, ПИСАТЬ, ЗАКРЫТЬ, ПРИНЯТЬ, ВЫДАТЬ.

Глаголы управления последовательностью процедур ПЕРЕЙТИ, ЕСЛИ, ВЫПОЛНИТЬ, ВЫЙТИ, ОСТАНОВИТЬ.

Особое место в разделе процедур занимает ПРИМЕЧАНИЕ. С помощью ПРИМЕЧАНИЯ в программу можно ввести необходимые пояснения (на выполнение программы эти пояснения влияния не оказывают). После слова ПРИМЕЧАНИЕ ставится точка, за которой может следовать любая комбинация символов языка, ограниченная точкой.

Комбинация глагола и его операндов составляет *высказывание*. Одно или несколько высказываний образуют *предложение*, в конце которого ставится точка. Например,

```
П1. ЧИТАТЬ АССОРТИМЕНТ В КОНЦЕ ПЕРЕЙТИ К ПАРАГР—2.
    ПОМЕСТИТЬ НАИМ В НАИМ — ВЫХ.
    ВЫЧИСЛИТЬ СУМ — ВКЛАДОВ = СУМ — ВКЛАДОВ + ВКЛАД.
    ЕСЛИ ВРЕМЯ БОЛЬШЕ НОРМ — ВРЕМЯ ВЫЧИСЛИТЬ ПЕРЕРАБ =
         ВРЕМЯ — НОРМ — ВРЕМЯ ВЫЧИСЛИТЬ СВЕРХУР = ПЕРЕ-
         РАБ * ЧАС — СТАВКА. ИНАЧЕ ПЕРЕЙТИ К П3.
```

Одно или несколько предложений могут объединяться в параграф. Перед первым предложением параграфа ставится название, заканчивающееся точкой. Один или несколько параграфов могут группироваться в секцию, являющуюся самой крупной конструкцией раздела процедур. Для этого перед названием первого параграфа секции ставится слово СЕКЦИЯ, за которым через пробел следует название секции, заканчивающееся точкой. В секцию входят все параграфы до следующего названия секции. При выборе названий для секций и параграфов программист должен учитывать те же требования, что и при выборе имен данных. Названия секций и параграфов служат для управления последовательностью выполнения процедур с помощью соответствующих глаголов, ссылающихся на эти названия.

В конце раздела процедур записывается КОНЕЦ—ПРОГРАММЫ.

Арифметический глагол.

```
ВЫЧИСЛИТЬ ВОЗРАСТ = 1973 — ГОД — РОЖД.
ВЫЧИСЛИТЬ СРЕДНЕКВ — ОТКЛ = (СУМ2/1000 — СУМ1 ** 2) ** (1/2)
```

Последнее предложение вычисляет величину

$$\text{СРЕДНЕКВ} - \text{ОТКЛ} = \sqrt{\frac{\text{СУМ2}}{1000} - \text{СУМ1}^2}$$

Порядок действий в арифметических выражениях определяется круглыми скобками, а при их отсутствии — старшинством операций. Вначале выполняется действие возведения в степень (**), далее — умножения (*) и деления (/), а затем — сложения (+) и вычитания (—).

Глагол ВЫЧИСЛИТЬ может использоваться с вариантом ОКРУГЛЯЯ, при этом последняя цифра, определяемая шаблоном результата, увеличивается на единицу, если старшая отбрасываемая цифра больше или равна пяти.

Пример. В разделе данных величины А, Б, В описаны таким образом:

```
01 А ШАБЛОН 99Т9.
01 Б ШАБЛОН 9Т99.
01 В ШАБЛОН 9 (3) Т99.
```

Если поля А и Б занимают соответственно числа 13.9 и 7.23, то после выполнения предложения

ВЫЧИСЛИТЬ В = А * Б

в поле В находится число 100.49, а после процедуры

ВЫЧИСЛИТЬ В = А * Б ОКРУГЛЯЯ

число 100. 50 (точный результат В = 100. 497).

Если количество разрядов в целой части результата превышает количество разрядов целой части, определяемое шаблоном результата, то такая ситуация рассматривается как переполнение. Можно предусмотреть в исходной программе разветвление хода выполнения программы в зависимости от наличия переполнения. При этом результату не будет присвоено вычисленное значение. Например,

```
ВЫЧИСЛИТЬ СУММА = ЦЕНА — ИЗД * КОЛ — ИЗД ПРИ ПЕРЕПОЛНЕНИИ
        ВЫДАТЬ ЦЕНА — ИЗД, КОЛ — ИЗД.
```

В случае переполнения на телетайп будет выдано значение указанных величин.

Глаголы перемещения данных. Глагол ПОМЕСТИТЬ служит для переноса данных из одного поля в другое. Например,

```
ПОМЕСТИТЬ ИМЯ В ИМЯ — 1
```

(значение поля ИМЯ передается в поле ИМЯ—1).

```
ПОМЕСТИТЬ АРТИКУЛ В АРТ — 1, АРТ — 2, АРТ — 3
```

(значение поля АРТИКУЛ передается в поля АРТ—1, АРТ—2, АРТ—3).

```
ПОМЕСТИТЬ НУЛЬ В ОБЩ — СУМ
```

(поле ОБЩ—СУМ заполняется нулями).

```
ПОМЕСТИТЬ 'ОШИБКА В ДАННЫХ' В ВЫХ — ЗАПИСЬ
```

(в поле ВЫХ—ЗАПИСЬ записывается комбинация символов ОШИБКА В ДАННЫХ).

```
ПОМЕСТИТЬ 1973 В ТЕК — ГОД
```

(в поле ТЕК—ГОД записывается число 1973).

При перемещении данных происходит их выравнивание по положению десятичной точки в соответствии с шаблоном.

Пример. В разделе данных величины СУММА, СУММА — 1, СУММА — 2, СУММА — 3 были описаны таким образом:

02 СУММА ШАБЛОН 9 (5)Т9 (3).
02 СУММА — 1 ШАБЛОН 9Т9 (5).
02 СУММА — 2 ШАБЛОН 9 (5).
02 СУММА — 3 ШАБЛОН 9 (6) Т99.

Пусть значение данного СУММА равно 23456.789. Тогда после выполнения предложения

ПОМЕСТИТЬ СУММА В СУММА — 1, СУММА — 2, СУММА — 3.

значения соответствующих данных будут:

СУММА — 1 6.78900
СУММА — 2 23456
СУММА — 3 023456.78

Глаголы ввода—вывода данных. Эти глаголы используются для ввода данных с внешних носителей в оперативную память и вывода из оперативной памяти на внешние носители.

Глагол ОТКРЫТЬ.

ОТКРЫТЬ { ВХОДНОЙ / ВЫХОДНОЙ } название-массива.

Глагол ОТКРЫТЬ подготавливает к обработке входные и выходные массивы (обеспечивает ввод и проверку или запись начальной метки, проверяет правильность использования внешних устройств и зон памяти для массивов и т. п.). Выполнение ОТКРЫТЬ для массива должно предшествовать выполнению ЧИТАТЬ (ПИСАТЬ) и ЗАКРЫТЬ для этого массива.

Глагол ЧИТАТЬ.

ЧИТАТЬ название массива В КОНЦЕ повелительное высказывание. Глагол помещает в зону входной записи очередную запись указанного массива. Массив предварительно должен быть открыт. Если при чтении обнаруживается, что массив не содержит больше записей, то выполняется повелительное высказывание.

Глагол ПИСАТЬ.

ПИСАТЬ название-записи. В результате выполнения этого глагола запись помещается в выходной массив.

Глагол ЗАКРЫТЬ.

Глагол выполняет необходимые операции по завершению обработки массива, например, запись или проверку конечных меток, если требуется, перемотку лент и пр.

Глагол ПРИНЯТЬ.

ПРИНЯТЬ имя данного [с { ТТ / ПУЛЬТА }].

Глагол используется для ввода данных небольшого объема с телетайпа (ТТ) или с центрального пульта управления ЦВМ. Если устройство не указано, подразумевается ввод с ТТ. Принимаемое данное должно быть описано в разделе данных.

Глагол ВЫДАТЬ.

Глагол используется для выдачи на телетайп или широкую печать небольшого объема данных. Если устройство явно не указано, то подразумевается вывод на телетайп. Например,

ВЫДАТЬ ЦЕНА — ИЗД, КОЛИЧ, ИТОГ НА ТТ

(содержимое полей, названных ЦЕНА—ИЗД, КОЛИЧ, ИТОГ, выдается на телетайп).

ВЫДАТЬ 'ОШИБКА В ДАННЫХ' НА ЫШ

(фраза ОШИБКА В ДАННЫХ печатается посредством устройства широкой печати).

ВЫДАТЬ ОБЩ — СУМ, 'РУБ'

(на телетайп выдается содержимое поля ОБЩ—СУМ, после чего печатается слово РУБ).

ВЫДАТЬ, 'ИТОГ = ' ИТОГ

(на телетайпе печатается ИТОГ =, после чего идет значение поля, названного ИТОГ). Например,

```
П1. ОТКРЫТЬ ВХОДНОЙ ВЕДОМОСТЬ. ОТКРЫТЬ ВЫХОДНОЙ МАС —
    ВЫПИСОК.
    ЧИТАТЬ ВЕДОМОСТЬ В КОНЦЕ ПЕРЕЙТИ К П2.
    ПОМЕСТИТЬ ФИО В ФИО — 1, ПОМЕСТИТЬ ДОЛЖНОСТЬ
    В ДОЛЖН — 1.
    ВЫЧИСЛИТЬ ЗАРПЛАТА = ВРЕМЯ * ЧАС — СТАВКА. ВЫЧИСЛИТЬ
    ИТОГ = ИТОГ + ЗАРПЛАТА ПРИ ПЕРЕПОЛНЕНИИ ВЫДАТЬ 'ПЕ-
    РЕПОЛНЕНИЕ'. ПИСАТЬ ВЫПИСКА. ПЕРЕЙТИ К П1.
П2. ВЫДАТЬ 'ИТОГ =', ИТОГ, 'РУБ'. ЗАКРЫТЬ ВЕДОМОСТЬ. ЗАКРЫТЬ
    МАС — ВЫПИСОК. КОНЕЦ — ПРОГРАММЫ.
```

Глаголы управления последовательностью процедур.

Глагол ПЕРЕЙТИ.

Вариант 1.

ПЕРЕЙТИ К П1;

Вариант 2.

ПЕРЕЙТИ К БЛОК — 1, А1Б, ОШИБКА В ЗАВИСИМОСТИ ОТ УКАЗАТ.

Величина УКАЗАТ должна быть описана в разделе данных и иметь целое значение 1, 2 или 3. Если УКАЗАТ = 1, осуществляется переход к параграфу (или секции) БЛОК—1, если УКАЗАТ = 2, то к параграфу (секции) А1Б, если УКАЗАТ = 3, то к параграфу (секции), названного ОШИБКА.

Для модификации действия предложения ПЕРЕЙТИ К ..., его можно выделить в отдельный параграф (дать ему название) и использовать глагол ИЗМЕНИТЬ ... ДЛЯ ПЕРЕХОДА. Например,

```
П1. ПЕРЕЙТИ К НОРМ — ОПЛАТА.
    . . . . . .
    ИЗМЕНИТЬ П1 ДЛЯ ПЕРЕХОДА К СВЕРХУР — ОПЛАТА.
```

При первом выполнении параграфа, названного П1, происходит переход к параграфу, имеющему название НОРМ—ОПЛАТА. После выполнения предложения ИЗМЕНИТЬ ... выполнение параграфа П1 будет вызывать переход к параграфу СВЕРХУР—ОПЛАТА.

Глагол ЕСЛИ.

ЕСЛИ условие { повелительное высказывание 1 / СЛЕДУЮЩЕЕ ПРЕДЛОЖЕНИЕ }

[ИНАЧЕ { повелительное высказывание 2 / СЛЕДУЮЩЕЕ ПРЕДЛОЖЕНИЕ }].

На месте условия в предложении может быть либо проверка отношения [НЕ] БОЛЬШЕ, [НЕ] МЕНЬШЕ, [НЕ] РАВНО, либо проверка класса данного [НЕ] ЦИФРОВОЕ, [НЕ] БУКВЕННОЕ. Другие типы условий, допускаемые КОБОЛом, рассматривать не будем. Если предложение состоит из нескольких высказываний, то высказывание ЕСЛИ ... должно быть последним в предложении. Например,

```
ЕСЛИ РАБ БОЛЬШЕ РАБ — НОРМ ВЫЧИСЛИТЬ ПЕРЕРАБ = РАБ —
РАБ — НОРМ ИНАЧЕ ПОМЕСТИТЬ НУЛЬ В ПЕРЕРАБ.
ЕСЛИ ГОД — РОЖД ЦИФРОВОЕ СЛЕДУЮЩЕЕ ПРЕДЛОЖЕНИЕ ИНАЧЕ
ПЕРЕЙТИ К ОШИБКА — В — ДАННЫХ.
ЕСЛИ ВРЕМЯ МЕНЬШЕ 7 ПЕРЕЙТИ К НОЧН — РАБ.
ЕСЛИ ВРЕМЯ МЕНЬШЕ 18 ПЕРЕЙТИ К ДНЕВ — РАБ.
ЕСЛИ ВРЕМЯ МЕНЬШЕ 24 ПЕРЕЙТИ К ВЕЧ — РАБ ИНАЧЕ
ПЕРЕЙТИ К ОШИБКА.
```

Сравнивать на БОЛЬШЕ, МЕНЬШЕ, РАВНО можно не только цифровые величины. Для нецифровых величин сравнение ведется посимвольно, начиная с самого левого символа данного, в соответствии с расположением символов в так называемой основной последовательности.

Для реализации КОБОЛа в САОД основная последовательность имеет вид:

0 1 2 ... 9 + — /, . $\_{10}$ ↑ ( ) X = ; [ ] * · ' ≠ < >:

А Б В ... Ю Я.

В соответствии с этой последовательностью наименьшей величиной является нуль, пробел больше любой цифры, но меньше любой буквы. Буква Я больше каждого из символов. Например, если значение данного Д1 СТОЛ, а данного Д2—СТУЛ, то Д1 < Д2, так как О < У.

Глагол ВЫПОЛНИТЬ.

Глагол ВЫПОЛНИТЬ позволяет использовать отдельные параграфы или секции КОБОЛ-программы в виде подпрограмм, т. е. передавать управление этим параграфам, а после их выполнения возвращаться к месту программы, откуда производилась передача управления.

Вариант 1.

ВЫПОЛНИТЬ название — процедуры — 1 [ПО название — процедуры — 2]

[{ целое / имя данного } {РАЗ / РАЗА}].

Примеры. ВЫПОЛНИТЬ ПОДПР — 1 ПО П4.
(Осуществляется передача управления параграфу, названному ПОДПР — 1, после которого выполняются все следующие за ним параграфы, включая параграф П4. После выполнения последнего высказывания параграфа П4 происходит возврат к предложению, следующему за ВЫПОЛНИТЬ...).

ВЫПОЛНИТЬ СВЕРХУР — РАБ.
(Если в данной КОБОЛ-программе СВЕРХУР — РАБ является названием параграфа, то возврат осуществляется после выполнения последнего высказывания этого параграфа, если же СВЕРХУР — РАБ является названием секции, то возврат происходит после выполнения последнего высказывания последнего параграфа в этой секции.)

ВЫПОЛНИТЬ РАСЧЕТ — ЗАРПЛ ПО УЧЕТ — ПРЕМИЙ 25 РАЗ.
(Осуществляется передача управления параграфу с названием РАСЧЕТ — ЗАРПЛ, после которого выполняются все следующие за ним параграфы, вплоть до параграфа с названием УЧЕТ — ПРЕМИЙ. После выполнения последнего высказывания параграфа УЧЕТ — ПРЕМИЙ управление вновь передается параграфу РАСЧЕТ — ЗАРПЛ. После 25-кратного повторения этого участка программы происходит возврат к предложению, следующему за ВЫПОЛНИТЬ ...)

Вариант 2.

ВЫПОЛНИТЬ название — процедуры — 1 [ПО название — процедуры — 2]

МЕНЯЯ имя — данного — 1 ОТ $\left\{\begin{array}{l}\text{числовой — литерал — 1}\\ \text{имя — данного — 2}\end{array}\right\}$

НА $\left\{\begin{array}{l}\text{числовой — литерал — 2}\\ \text{имя — данного — 3}\end{array}\right\}$ ДО отношение.

Пример. ВЫПОЛНИТЬ П1 ПО П3 МЕНЯЯ ЧАС — СТАВКА ОТ ЧАС — СТАВКА — МИНИМ НА ИЗМ — СТАВКИ ДО ЧАС — СТАВКА БОЛЬШЕ ЧАС — СТАВКА — МАКС.
(Осуществляется передача управления параграфу П1, после которого выполняются все следующие за ним параграфы вплоть до параграфа с названием П3. При этом переменная ЧАС — СТАВКА имеет значение ЧАС — СТАВКА — МИНИМ. После выполнения последнего высказывания параграфа П3 производится прибавление к значению переменной ЧАС — СТАВКА значения величины ИЗМ —СТАВКИ. Если полученное значение не превышает значения переменной ЧАС — СТАВКА — МАКС, происходит передача управления параграфу П1, и выполнение всех параграфов от П1 до П3 повторяется при полученном новом значении переменной ЧАС — СТАВКА. После этого повторяется процесс изменения переменной ЧАС — СТАВКА и проверка условия ЧАС — СТАВКА БОЛЬШЕ ЧАС — СТАВКА — МАКС. Когда условие окажется выполненным, происходит возврат к предложению, следующему за ВЫПОЛНИТЬ...).

Глагол ВЫЙТИ.

Глаголом ВЫЙТИ пользуются как одним из предложений, исполняемых после предложения ВЫПОЛНИТЬ. Его всегда выделяют в отдельный параграф. Глагол обеспечивает возврат после окончания выполнения предписания ВЫПОЛНИТЬ в случае, когда выполняемая процедура имеет несколько разветвлений и конечную точку, общую для всех разветвлений. Например,

```
ВЫПОЛНИТЬ КОНТРОЛЬ — УРОВНЯ.
. . . . . .
КОНТРОЛЬ — УРОВНЯ. ЕСЛИ УРОВЕНЬ РАВЕН НАЛИЧИЕ ПЕРЕЙТИ
К ОКОНЧ. ЕСЛИ УРОВЕНЬ МЕНЬШЕ НАЛИЧИЕ ВЫЧИСЛИТЬ РАЗН =
НАЛИЧИЕ — УРОВЕНЬ ЗАТЕМ ПЕРЕЙТИ К ОКОНЧ. ВЫПОЛНИТЬ
ОФОРМЛ.
ОКОНЧ. ВЫЙТИ.
```

Глагол ОСТАНОВИТЬ.

ОСТАНОВИТЬ {литерал / РАБОТУ}.

Этот глагол вызывает остановку машины. Указанный литерал выводится на телетайп, что используется для определения места в программе, где произошел останов. После этого программа может быть пущена дальше с пульта управления ЦВМ. Часто такую остановку машины используют для смены магнитных лент, установки новой колоды перфокарт и т. п.

Индексация. Применение индексных величин облегчает пользование табличными данными.

Индексом называют целое, значение которого определяет элемент, на который ссылаются. В качестве целого можно воспользоваться также именем данного. Индекс заключается в скобки и ставится непосредственно за пробелом, который следует за наименованием индексированной величины. Эта величина обычно входит в состав группового данного, описанного в разделе данных с фразой ПОВТОРЯЕТСЯ.

Пример. На магнитной ленте записан массив сведений о результатах сдачи экзаменационной сессии в вузе. Массив состоит из записей, каждая из которых содержит сведения об одной группе. Запись может быть следующей:

ФИО	Физика	Математика	Начертательная геометрия	История КПСС
Антонов Р.М.	3	4	3	5
Архипов С.М	4	4	4	4
Бутенко В.А.	4	5	5	5
Васильева Э.С.	5	5	4	4

В разделе данных такая таблица может быть описана следующим образом:

```
01 ИТОГИ — СЕССИИ.
   02 СТРОКА — ИТОГОВ ПОВТОРЯЕТСЯ 25 РАЗ.
      03 ФИО ШАБЛОН А (15).
      03 ФИЗИКА ШАБЛОН 9.
      03 МАТЕМАТ ШАБЛОН 9.
      03 НАЧ — ГЕОМ ШАБЛОН 9.
      03 ИСТ — КПСС ШАБЛОН 9.
```

Тогда ФИО (3) означает ссылку на фамилию Бутенко В. А., ФИЗИКА (4) имеет значение 5 и т. д.

Пример раздела процедур. Пусть на магнитной ленте записан массив КРЕДИТНЫЕ — ССУДЫ, содержащий записи, состоящий из двух элементарных единиц данных НАЧ — ДОЛГ и ВЗНОС. Каждая кредитная ссуда, составляющая вначале сумму НАЧ — ДОЛГ, должна погашаться фиксированными ежемесячными взносами (кроме последнего месяца, где обычно сумма меньше). Взнос (ВЗНОС) состоит из суммы ПОГАШЕНИЕ и ПРОЦЕНТ. Для всякого значения НАЧ — ДОЛГ должен быть построен план погашения, причем для каждого месяца должны быть выписаны пять элементарных пунктов: МЕСЯЦ, ВХ — ДОЛГ, ПОГАШЕНИЕ, ПРОЦЕНТ и ВЫХ — ДОЛГ, образуя вместе запись ПЛАТЕЖ в выходном массиве ПЛАН — ПОГАШЕНИЯ. Норма процента одинакова для всех ссуд и вводится в память ЦВМ с телетайпа глаголом ПРИНЯТЬ. Раздел процедур этой программы имеет вид:

```
РАЗДЕЛ ПРОЦЕДУР.
П1. ПРИНЯТЬ НОРМА — ПРОЦ.
    ОТКРЫТЬ ВХОДНОЙ КРЕДИТНЫЕ — ССУДЫ.
    ОТКРЫТЬ ВЫХОДНОЙ ПЛАН — ПОГАШЕНИЯ.
СЧИТЫВ. ЧИТАТЬ КРЕДИТНЫЕ — ССУДЫ В КОНЦЕ ПЕРЕЙТИ К П2.
    ПОМЕСТИТЬ НАЧ — ДОЛГ В ВХ — ДОЛГ.
    ВЫПОЛНИТЬ МЕС — ПЛАТЕЖ МЕНЯЯ М ОТ1 НА 1 ДО
    ВХ — ДОЛГ НЕ БОЛЬШЕ НУЛЯ. ПЕРЕЙТИ К СЧИТЫВ.
МЕС — ПЛАТЕЖ. ПОМЕСТИТЬ М В МЕСЯЦ.
    ВЫЧИСЛИТЬ ПРОЦЕНТ = ВХ — ДОЛГ * НОРМА — ПРОЦ/100.
    ВЫЧИСЛИТЬ ВЫХ — ДОЛГ = ВХ — ДОЛГ + ПРОЦЕНТ — ВЗНОС.
    ПОМЕСТИТЬ ВЫХ — ДОЛГ В ПРОМЕЖ — ДОЛГ.
    ЕСЛИ ВЫХ — ДОЛГ МЕНЬШЕ НУЛЯ ПОМЕСТИТЬ НУЛЬ В ВЫХ —
    ДОЛГ.
    ВЫЧИСЛИТЬ ПОГАШЕНИЕ = ВХ—ДОЛГ — ВЫХ—ДОЛГ.
    ПИСАТЬ ПЛАТЕЖ.
    ПОМЕСТИТЬ ПРОМЕЖ — ДОЛГ В ВХ — ДОЛГ.
П2. ЗАКРЫТЬ КРЕДИТНЫЕ — ССУДЫ. ЗАКРЫТЬ ПЛАН — ПОГАШЕ-
НИЯ. КОНЕЦ — ПРОГРАММЫ.
```

Раздел процедур содержит четыре параграфа: П1, СЧИТЫВ, МЕС—ПЛАТЕЖ и П2.

В параграфе П1 норма процента вводится с телетайпа, массивы открываются для чтения и записи.

В параграфе СЧИТЫВ по одной считываются записи из массива КРЕДИТНЫЕ — ССУДЫ.

Когда считана одна запись, значение поля НАЧ — ДОЛГ передается в поле ВХ — ДОЛГ и выполняется параграф МЕС — ПЛАТЕЖ, в котором вычисляется и записывается на внешний носитель платеж для одного месяца в виде записи ПЛАТЕЖ, состоящей из пяти полей. Выполнение этого параграфа повторяется столько раз, сколько требуется для того, чтобы начальный долг оказался погашенным. При каждом выполнении параграфа МЕС — ПЛАТЕЖ переменная М, обозначающая месяц, возрастает на 1. Если вычисленное значение ВЫХ — ДОЛГ окажется отрицательным, то оставшийся долг меньше взноса, поэтому в поле ВЫХ—ДОЛГ передается нуль. Вспомогательная переменная ПРОМЕЖ—ДОЛГ используется для того, чтобы получить значение ВХ — ДОЛГ для следующего месяца.

Когда окажется, что входной массив не содержит больше записей, осуществляется переход к параграфу П2 для проведения заключительных операций — закрытия массивов и остановки.

§ 8.5. ПРИМЕР СОСТАВЛЕНИЯ КОБОЛ-ПРОГРАММЫ

Пусть на магнитной ленте имеется массив, каждая запись которого содержит заголовок и каталоговый номер некоторой научно-технической статьи. Определенным словам (*ключевым словам*) в заголовке, характеризующим основное направление содержания статьи, предшествует звездочка (*). Каждое слово оканчивается пробелом.

Задача состоит в том, чтобы просмотреть заголовки и для каждого найденного ключевого слова сформировать новую запись, в которой ключевое слово занимает определенное поле. Кроме того, эта новая запись должна содержать заголовок (уже без звездочек) и каталоговый номер. Такие записи образуют выходной массив. Это делается с целью обеспечения возможности классификации статей в соответствии с ключевыми словами, которая может быть выполнена посредством стандартных программ сортировки.

Длина заголовка ограничена 150 символами, неиспользованные позиции заполнены пробелами. Каталоговый номер содержит 6 разрядов. Для ключевого слова в новой записи отводится 25 символьных позиций, более длинные слова обрезаются.

КОБОЛ-программа имеет следующий вид:

```
РАЗДЕЛ ИДЕНТИФИКАЦИИ.
ПРОГРАММА. СОСТАВЛЕНИЕ УКАЗАТЕЛЯ СТАТЕЙ.
ЗАМЕЧАНИЯ. ПРОГРАММА ВЫДЕЛЯЕТ КЛЮЧЕВЫЕ СЛОВА,ОТМЕЧЕН-
          НЫЕ ЗВЕЗДОЧКОЙ, ИЗ ЗАГОЛОВКОВ СТАТЕЙ.
РАЗДЕЛ  ОБОРУДОВАНИЯ.
СЕКЦИЯ  КОНФИГУРАЦИИ.
РАБОЧАЯ — МАШИНА.  Р — 30.
СЕКЦИЯ ВВОДА — ВЫВОДА.
УПРАВЛЕНИЕ — МАССИВАМИ.
    ДЛЯ ВХ — МАС ПРЕДНАЗНАЧИТЬ МЛ1.
    ДЛЯ ВЫХ — МАС ПРЕДНАЗНАЧИТЬ МЛ2.
РАЗДЕЛ ДАННЫХ.
СЕКЦИЯ МАССИВОВ.
ОМ ВХ — МАС МЕТКИ ОПУЩЕНЫ.
01 СТАТЬЯ.
    02 ЗАГОЛОВОК ШАБЛОН Х ПОВТОРЯЕТСЯ 150 РАЗ.
    02 КАТАЛОГ — НОМ ШАБЛОН 9 (6).
ОМ ВЫХ — МАС МЕТКИ ОПУЩЕНЫ.
01 СТАТЬЯ — С — УКАЗАТЕЛЕМ.
    02 КЛЮЧЕВОЕ — СЛОВО.
     03 СИМВОЛ — КЛ — СЛОВА  ШАБЛОН  Х ПОВТОРЯЕТСЯ 24 РАЗА.
    02 ЗАГОЛ — ВЫХ ШАБЛОН Х ПОВТОРЯЕТСЯ 150 РАЗ.
    02 КАТ — НОМ — ВЫХ ШАБЛОН 9 (6).
СЕКЦИЯ РАБОЧЕЙ — ПАМЯТИ.
77 И1 ШАБЛОН 999 ДЛЯ ВЫЧИСЛЕНИЙ.
77 И2 ШАБЛОН 999 ДЛЯ ВЫЧИСЛЕНИЙ.
77 И3 ШАБЛОН 999 ДЛЯ ВЫЧИСЛЕНИЙ.
77 Т ШАБЛОН Х.
РАЗДЕЛ ПРОЦЕДУР.
ПОДГ. ОТКРЫТЬ ВХОДНОЙ ВХ — МАС. ОТКРЫТЬ ВЫХОДНОЙ ВЫХ —
      МАС.
ЧТ — ВХ — М.  ЧИТАТЬ  ВХ — МАС В КОНЦЕ ПЕРЕЙТИ К ЗАКЛ.
                ПОМЕСТИТЬ 1 В И3.
ПРОВ — КЛ — СЛОВ.  ЕСЛИ ЗАГОЛОВОК(И3) РАВЕН '*' ПЕРЕЙТИ
                К ЗАПОМ — КЛ — СЛОВА.
ПКС2. ВЫЧИСЛИТЬ И3 = И3 + 1. ЕСЛИ И3 БОЛЬШЕ 150 ПЕРЕЙТИ
      К ЧТ — ВХ — М. ПЕРЕЙТИ К ПРОВ — КЛ — СЛОВ.
ЗАПОМ — КЛ — СЛОВА. ПОМЕСТИТЬ ПРОБЕЛ В КЛЮЧЕВОЕ — СЛОВО.
                ПОМЕСТИТЬ 1 В И2.
ЗКС2. ВЫЧИСЛИТЬ И3 = И3 + 1. ЕСЛИ И3 БОЛЬШЕ 150 ПЕРЕЙТИ
      К ФОРМ — ЗАГОЛ.  ПОМЕСТИТЬ ЗАГОЛОВОК (И3) В Т.
      ЕСЛИ Т РАВНО ПРОБЕЛ ПЕРЕЙТИ К ФОРМ — ЗАГОЛ.
      ПОМЕСТИТЬ Т В СИМВОЛ — КЛ — СЛОВА(И2).
      ЕСЛИ И2 БОЛЬШЕ 25 ПЕРЕЙТИ К ФОРМ — ЗАГОЛ.
      ВЫЧИСЛИТЬ И2 = И2 + 1. ПЕРЕЙТИ К ЗКС2.
ФОРМ — ЗАГОЛ. ПОМЕСТИТЬ 1 В И1, И2.
Ф32. ПОМЕСТИТЬ ЗАГОЛОВОК (И1) В Т.
     ЕСЛИ Т РАВНО '*' ПЕРЕЙТИ К Ф33.
     ПОМЕСТИТЬ Т В ЗАГОЛ — ВЫХ (И2). ВЫЧИСЛИТЬ И2 = И2 + 1.
     ЕСЛИ И1 РАВНО 150 ПЕРЕЙТИ К Ф34.
Ф33. ВЫЧИСЛИТЬ И1 = И1 + 1. ПЕРЕЙТИ К Ф32.
Ф34. ЕСЛИ И2 БОЛЬШЕ 150 ПЕРЕЙТИ К ЗАП — СТ — С — УКАЗ.
     ПОМЕСТИТЬ ПРОБЕЛ В ЗАГОЛ — ВЫХ (И2). ВЫЧИСЛИТЬ И2 =
     = И2 + 1. ПЕРЕЙТИ К Ф34.
ЗАП — СТ—С — УКАЗ. ПОМЕСТИТЬ КАТАЛОГ— НОМ В КАТ—НОМ—ВЫХ.
                ПИСАТЬ СТАТЬЯ —С— УКАЗАТЕЛЕМ. ПЕРЕЙТИ
                К ПКС2.
     ЗАКЛ. ЗАКРЫТЬ ВХ — МАС. ЗАКРЫТЬ ВЫХ — МАС.
          ВЫДАТЬ 'ФОРМИРОВАНИЕ ВЫХОДНОГО МАССИВА ЗАКОН-
          ЧЕНО'. КОНЕЦ — ПРОГРАММЫ.
```

РАЗДЕЛ ИДЕНТИФИКАЦИИ И РАЗДЕЛ ОБОРУДОВАНИЯ понятны и не требуют пояснений.

В РАЗДЕЛЕ ДАННЫХ описываются массивы ,ВХ — МАС и ВЫХ — МАС, содержащие каждый по одному типу записей, СТАТЬЯ и СТАТЬЯ — С — УКАЗАТЕЛЕМ соответственно. Поле ЗАГОЛОВОК в записи СТАТЬЯ интерпретируется как таблица отдельных символов, поскольку необходимо анализировать каждый символ в отдельности.

В РАЗДЕЛЕ ПРОЦЕДУР КЛЮЧЕВОЕ — СЛОВО в записи СТАТЬЯ — С — УКАЗАТЕЛЕМ обрабатывается в одном случае как единое целое (когда оно заполняется пробелами), а в другом случае — как таблица символов, поэтому в РАЗДЕЛЕ ДАННЫХ оно описано как групповое данное, отдельные элементы которого являются элементами таблицы. В секции рабочей памяти описаны три целых переменных И1, И2, И3, которые служат в качестве индексов при просмотре и перемещении таблиц. Т — вспомогательная переменная.

В РАЗДЕЛЕ ПРОЦЕДУР в параграфе ПОДГ открываются массивы. Каждая запись считывается в параграфе ЧТ — ВХ — М и обрабатывается в остальной части программы.

Когда все записи обработаны, выполняется параграф ЗАКЛ, закрывающий массивы, выдающий на телетайп заключительное сообщение и останавливающий ЦВМ. Обработка записей начинается с последовательного анализа символов заголовка до тех пор, пока не будет найдена звездочка. Этот анализ использует переменную И3 как счетчик. Если текущим символом будет звездочка, то управление передается параграфу ЗАПОМ — КЛ — СЛОВА. После возврата анализ продолжается. С окончанием анализа последнего символа заголовка считывается новая запись и обрабатывается таким же образом.

Запоминание ключевых слов начинается заполнением символьных позиций, зарезервированных для данного с именем КЛЮЧЕВОЕ — СЛОВО, пробелами, чтобы стереть всю предыдущую информацию. Затем все символы после звездочки до пробела (но не более 25) переносятся в поле КЛЮЧЕВОЕ — СЛОВО, после чего управление передается параграфу ФОРМ — ЗАГОЛ, в котором содержимое поле ЗАГОЛОВОК, кроме звездочек, передается в поле ЗАГОЛ — ВЫХ записи СТАТЬЯ — С — УКАЗАТЕЛЕМ. После этого в параграфе ЗАП — СТ — С — УКАЗ данное КАТАЛОГ — НОМ входной записи помещается в поле КАТ — НОМ—ВЫХ, выходная запись выводится на магнитную ленту и осуществляется переход к параграфу ПКС2, где начинается поиск другого ключевого слова в том же заголовке.

ГЛАВА 9

ПОРЯДОК РЕШЕНИЯ ЗАДАЧ НА ЦВМ И ОБРАБОТКА РЕЗУЛЬТАТОВ

Рассмотренные в предыдущих главах этапы подготовки задач для решения на машине заканчиваются составлением программы на одном из алгоритмических языков. Программы записываются на программных бланках с помощью алфавитно-цифровых и специальных символов. Ввод программ непосредственно с бланков в память машины в настоящее время невозможен, поэтому перед вводом они должны преобразовываться к виду, понятному машине. В гл. 1 уже рассматривались устройства подготовки данных, используемые для этих целей, и способы представления информации на носителях.

Программа, записанная на алгоритмическом языке, перед выполнением должна быть переведена на язык машины (язык команд). Функции перевода программ, записанных на алгоритмическом языке, в программы, записанные в коде машины, выполняются специальной программой-транслятором.

Для каждой машины, использующей тот или иной алгоритмический язык, заранее составляются программы-трансляторы, которые хранятся в памяти машины. После ввода программы в память машины вызывается программа-транслятор, осуществляющая перевод программы в код машины. Во время трансляции попутно с переводом проверяется соответствие конструкций программы, записанной в алгоритмическом языке, грамматическим правилам этого языка.

Поскольку программирование является очень сложным и трудоемким процессом, то неизбежны ошибки, допускаемые на различных этапах. Выявление и исправление этих ошибок происходит во время отладки программы, которая предшествует непосредственному решению задачи на машине.

§ 9.1. ОТЛАДКА ПРОГРАММ

Цель отладки — устранить ошибки, допущенные на всех предыдущих этапах подготовки задач для решения на ЦВМ и получить программу, дающую правильные результаты. Ошибки, не позволяющие получить правильные результаты решения задачи на машине, следующие: 1) неправильно нанесена программа и исходные данные на носитель информации; 2) ошибки в написании команд и операторов программы; 3) ошибки в алгоритме решения задачи.

Обнаружение и устранение ошибок неправильного нанесения программы и исходных данных на носители информации. Ошибки этого типа могут быть обнаружены путем внешнего осмотра носителя информации, распечаткой программ и исходных данных на печатающих устройствах и последующим сравнением с программой и исходными данными, записанными на бланке. Обнаружение таких ошибок является делом достаточно простым и подробно рассматриваться не будет.

Устраняются ошибки этого типа путем повторного нанесения всей программы или отдельных ее участков и исходных данных на носители информации. Если в качестве носителя информации используют перфокарты, то заменяют перфокарты с ошибками на новые, при использовании перфоленты, необходимо между отдельными блоками программы оставлять пропуски, которые позволяют вырезать неправильно отперфорированный участок и заменить его на новый.

Устранение ошибок этого типа осуществляется либо до выхода на машину, либо сразу после ввода программы и исходных данных в машину.

Обнаружение и устранение ошибок в написании команд и операторов программы. Ошибки этого типа обнаруживаются частично при переводе программы в код машины (в случае использования алгоритмических языков) или в процессе решения задачи на машине.

Алгоритмические языки имеют набор формальных правил записи операторов, обеспечивающих однозначный перевод программ в код машины. Отступление от этих правил приводит к выявлению самой машиной причины ошибки и выдаче из машины типа этой ошибки с указанием оператора программы и места в операторе, где она обнаружена.

В частности, при нарушении синтаксических правил записи в языке АКИ в трансляторе предусмотрены специальные остановы, перечень которых приведен в табл. 9.1. После останова на телетайп можно вывести оператор, содержащий ошибку. При этом будут отпечатаны номера останова, страницы, строки программы, сам оператор и знак вопроса в том месте, где допущена ошибка. Например,

```
Ост. 30 Лист 01
03
1 ВЫЧ?Z = ...
```

Согласно таблице останов 30 говорит о том, что есть ошибка в названии оператора. Место ошибки указывается знаком вопроса «?». В языке АКИ названия операторов записываются русскими буквами. Здесь оператор начинается с цифры. Цифрами, стоящими перед оператором, обозначаются метки, но после них должны быть обязательно точки.

Другие типы остановов рассматриваться не будут, так как они достаточно хорошо иллюстрируются содержанием таблицы.

В случае использования алгоритмического языка АЛГОЛ ошибки такого рода также обнаруживаются при трансляции. При этом на печать выводится следующая информация: 1) текст, характеризующий тип допущенной ошибки или указывающий способ ее устранения; 2) часть алгольной программы, в которой обнаружена ошибка; 3) после ошибки печатается текст: «Трансляция не закончена» и строка черточек. Например,

```
Ошибка в записи выражения с условием
                p1041 (g); gd : = if j = 0 then g [4];
Трансляция не закончена.
— — — — — — — — — — — — — — — —
```

Таблица 9.1

Номер останова	Причина останова
01	Неверная запись числа (есть не цифра) или названия переменной (массива)
02	Число выходит за пределы допустимого значения
03	Количество констант превосходит допустимое (191)
04	Пропущен один из разделителей
05	Число простых переменных превосходит 192
06	Подготовка цикла требует слишком много команд
07	Количество рабочих ячеек превосходит максимальное
10	Переменная с индексом относится к массиву, который не был описан
11	Одновременно описано более 32 массивов
12	В операторе ПОВТОРИТЬ — недостаточно информации для построения цикла
13	Число повторений цикла в операторе ПОВТОРИТЬ — получается не целым. Например, ПОВ — 2 — 1 = 1 — (2) — 6^{∇}_{Δ}
14	При переименовании массивов новый массив не помещается на месте старого
15	В операторе НАЗВАТЬ — есть ошибка
16	Число символов в каком-либо операторе больше допустимого (768)
17	Ошибка в операторе ПОВТОРИТЬ
20	Ошибка в операторе ВВОД — , нарушена последовательность описание простых переменных и массивов
21	Не хватает индексных ячеек
22	Неверная запись знака действия
23 и 24	Необходимо упростить запись арифметического выражения (формулы), расчленив его на несколько частей
25	Обнаружены лишние знаки действия или лишние открывающие скобки
26	Указана функция, отсутствующая в АКИ, например, LG(X) и т. д.
27	Число закрывающих скобок превышает число открывающих
30	Ошибка в названии оператора или служебного слова
31	Допущена ошибка в использовании меток: а) пропущена метка, б) одна и та же метка присвоена нескольким операторам, в) общее число меток превышает 127
32	Производится неразрешенное в АКИ деление целых чисел, например, ВЫЧ — : Z=N:M $^{\nabla}_{\Delta}$
33	Не задано число повторений цикла, входящего в другой цикл
34	Неверно распределен формат в операторе НАПЕЧАТАТЬ — ТАБЛИЦУ —
100	На пульте управления не включен тумблер, определяющий вид ввода
101	Велик объем автокодовой программы с дополнениями к ней
102	Дополнения начинаются не с корректировочного оператора или последний записан неверно
103	Строка бланка не заканчивается одним из следующих символов: $^{\nabla}_{\Delta}$ или $\equiv$. Нет пяти пробелов в конце автокодовой программы
104	Нарушена последовательность записи номеров корректирующих строк
105	Есть ссылка на несуществующую метку
106	Рабочая программа не помещается в МОЗУ
107	Нарушены правила построения цикла
110	Плохо работает магнитная лента при вызове стандартных и библиотечных программ
111	В автокодовой программе есть обращение к библиотечной программе, не включенной в АКИ

После печати этой информации наступает останов. Нажатие кнопки «Пуск» также приводит к останову.

Обнаруженные при трансляции ошибки исправляются с помощью корректировочных операторов или путем замены отдельных участков программы на исправленные. После этого программа вновь транслируется; в итоге получают программу в коде машины.

Поскольку при трансляции могут быть обнаружены не все возможные типы ошибок, то при решении задачи проводится дальнейшая отладка программы, полученной после трансляции, а также программы, составленной в коде машины.

Перед решением задачи на машине вручную просчитывается один из ее вариантов, называемый *контрольным* или *отладочным*. Отладочный вариант нужно просчитывать при тех значениях исходных данных, которые позволяют выявить по возможности большее число ошибок. Результаты, полученные при ручном просчете, сравниваются затем с результатами машинного счета. Однако результаты машинного счета удается получить сразу не всегда, т. е. в программе могут быть ошибки в записи команд. Это в особенности относится к программам, составленным в коде машины. Наличие таких ошибок может привести к останову машины, переполнению разрядной сетки машины (аварийный останов), зацикливанию программы.

Останов машины. Останов машины может произойти в случае, если код операции одной из команд программы является не используемым. Например, у машин серии «Минск» неиспользуемыми кодами будут коды «—21», «—22», «—23» и др. Наличие в старших разрядах команды этих кодов приведет к останову. Такие остановы возможны из-за неправильной записи команд программы, ошибок при нанесении программы на носители информации, неправильной передачи управления в программе, приводящей к передаче управления тем ячейкам, в которых находятся не команды, а какая-либо числовая информация (например, исходные данные). Способы обнаружения и исправления ошибок первых двух типов уже рассматривались выше.

Переполнение разрядной сетки машины. Переполнение разрядной сетки машины возникает в случае, когда какие-либо промежуточные или окончательные результаты по абсолютной величине получаются большими, чем предельно допустимые для машины данного типа. Обнаружить ошибки такого типа можно с помощью контрольного просчета, а устранить их — с помощью масштабирования, т. е. умножения либо исходных данных, либо промежуточных результатов на масштабные коэффициенты, меньшие единицы, с последующим их восстановлением. Например, если возникает переполнение при вычислении $Z = (x^2 - y^2)/c$ из-за того, что x^2 или y^2 больше предельно допустимого числа, то можно перед вычислением каждого из этих чисел умножить их на масштабный коэффициент, например, равный 0,1 (т. е. вычислить сначала $r = 0{,}1x$ и $g = 0{,}1y$, затем $S = (r^2 - g^2)/c$ и, наконец, $Z = 100S$).

Устранить переполнение часто удается путем преобразования исходной формулы, а следовательно, и порядка выполнения действий

машиной. В данном примере Z можно было бы вычислять по следующей формуле: $Z = [(x + y)/c]\ (x - y)$.

Данная формула не содержит x^2 и y^2, поэтому уменьшается возможность переполнения.

Если окончательный результат больше предельно допустимого числа, то устранить переполнение можно масштабированием, а результат восстановить вручную после окончания решения задачи.

Причиной, вызывающей переполнение разрядной сетки машины, может быть также неправильная запись адресов команд или неправильная передача управления. При этом в операциях могут участвовать не исходные числа, а какая-либо вспомогательная информация или даже команды.

Наиболее часто переполнение возникает при выполнении операции деления, когда в качестве адреса делителя указывается адрес пустой ячейки, что равносильно делению на нуль.

Зацикливание программы. Зацикливание программы происходит в случае, когда машина совершает вычисления по некоторому замкнутому циклу, не останавливаясь.

Причины, вызывающие зацикливание, могут быть различные и все они приводят либо к неправильному изменению счетчика циклов, либо к отсутствию команд, осуществляющих проверку условия окончания цикла или выхода из него.

Обнаружить зацикливание можно по изменению содержимого счетчика циклов. Чтобы устранить зацикливание, необходимо проверить и исправить команды, организующие цикл, или, в случае программирования на алгоритмических языках, проверить и исправить операторы, используемые для его организации. Но иногда зацикливание может привести к переполнению, из-за чего его обнаружение затрудняется.

Ошибки в передаче управления и записи команд, не вызвав останов, переполнение или зацикливание, приводят к получению неправильных результатов. Ошибки в программе можно обнаружить путем сравнения полученных на машине результатов с результатами контрольного просчета. Значительно труднее найти место в программе, где допущена ошибка, и определить тип последней. Для этих целей используют следующие способы обнаружения ошибок.

Проверка правильности выполнения команд программы в однотактном режиме. В данном случае последовательно выполняются все команды программы и путем сравнения с результатами контрольного просчета проверяется правильность полученных результатов. При этом легко проверить правильность выполнения команд передачи управления и команд, организующих цикл. Однако такой метод неэффективен, так как требует очень больших затрат машинного времени. Поэтому его применяют лишь для проверки отдельных небольших участков программы, на которых ожидается обнаружение ошибки.

Метод отладки программ в режиме контрольных остановов. Сущность этого метода состоит в том, что решение задачи разбивается на ряд отдельных этапов. После выполнения каждого этапа машина останавливается и выдает на печать информацию для анализа правиль-

ности программы. Правильность программы проверяется путем сравнения выведенной из машины информации с соответствующими результатами ручного контрольного просчета.

В зависимости от типа машины контрольные остановы осуществляются либо программным путем (введение специальных команд, пометка контрольными знаками), либо путем включения специальных тумблеров на пульте управления машины.

Использование отладочных программ. Наибольший эффект дает автоматизация процесса отладки с помощью специальных отладочных программ. Сущность работы отладочной программы заключается в том, что автоматически после выполнения каждой команды выводится на печать вся необходимая информация, используемая для контроля правильности ее выполнения.

Для различных машин и алгоритмических языков программирования существует много отладочных программ, отличающихся друг от друга как способами их использования, так и количеством выводимой на печать информации. По этой причине в данной книге эти программы рассматриваться не будут.

Ошибки в алгоритме решения задачи. Сущность этих ошибок заключается в том, что программа реализует алгоритм, не соответствующий решаемой задаче. Причиной этого может быть либо неправильно составленный алгоритм, либо неправильная его реализация. Наличие ошибки в алгоритме можно установить путем сравнения результатов, полученных на машине, с результатами контрольного просчета. Однако процесс обнаружения самой ошибки является достаточно сложным.

При неправильно составленном алгоритме машинные методы обнаружения ошибок неприемлемы. Для проверки правильности алгоритма целесообразно вручную проверить его выполнение для упрощенного варианта, например уменьшив количество рассматриваемых вариантов до двух-трех и взяв удобные для вычислений числа.

В случае неправильной реализации алгоритма можно также проверить программу путем упрощенного просчета или путем использования способов отладки, рассмотренных выше. При проверке правильности алгоритма и его реализации целесообразно вручную имитировать работу машины, осуществляя соответствующие передачи управления и запись результатов в определенные ячейки памяти машины.

§ 9.2. КОНТРОЛЬ ПРАВИЛЬНОСТИ ВЫЧИСЛЕНИЙ

Причиной получения неправильного результата могут быть не только ошибки в программе, которые устраняются в процессе отладки, но и ошибки, вызванные неисправностью самой машины. В процессе вычислений машина может допускать как систематические, так и случайные ошибки.

Систематические ошибки возникают при неисправности отдельных устройств и элементов машины. В этом случае многократное решение задачи на машине будет приводить к получению одного и того же

неправильного результата. Другими словами, причиной систематической ошибки является неисправность вычислительной машины. Обнаружение систематических ошибок и устранение причин, их порождающих, осуществляется персоналом, обслуживающим машину.

Случайные ошибки или *сбои* могут возникать вследствие влияния случайных факторов таких, как изменение напряжения питающей сети, перепадов температур, резких толчков и т. д.

Способы обнаружения случайных ошибок. Для обнаружения случайных ошибок используют программные методы, сущность которых состоит в многократном повторении вычислений с последующим сравнением их результатов и использование контрольных соотношений.

Двойной счет. Этот способ контроля производится во время решения задачи следующим образом. В основную программу включают специальные процедуры, повторяющие вычисление по всей программе или ее участку дважды. В случае совпадения результатов обоих просчетов можно судить об их правильности.

Блок-схема алгоритма, включающего двойной счет, приведена на рис. 9.1.

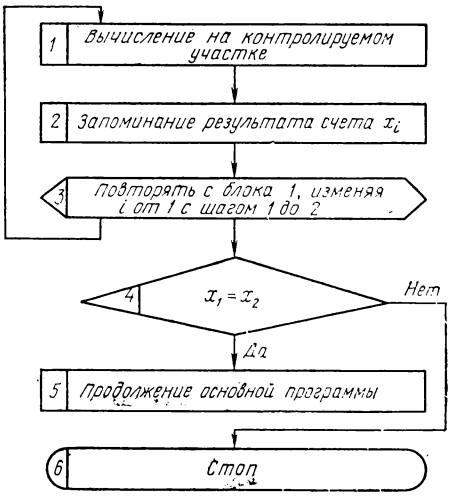


Рис. 9.1. Блок-схема алгоритма двойного счета

Поскольку двойной счет требует больших затрат машинного времени для его реализации, то им проверяют правильность выполнения не всей программы, а только наиболее важных ее участков.

Способ двойного счета дает возможность установить правильность вычислений только при совпадении результатов обоих просчетов. При несовпадении результатов нельзя сказать, какой из двух результатов является правильным. Ответ на этот вопрос можно получить, используя метод двойного-тройного счета.

Двойной-тройной счет. Сущность этого метода заключается в том, что правильный результат определяется при совпадении результатов двух любых просчетов из трех. Блок-схема алгоритма, реализующего этот способ, приведена на рис. 9.2.

Здесь сначала осуществляется контроль способом двойного счета. Если результаты обеих просчетов совпадают, то результат признается правильным, и третьего просчета не производится. Если результаты первого и второго счетов не совпадают, то производится третий счет. Затем сравниваются результаты третьего счета с первыми двумя. Совпадение любых двух результатов позволяет судить об их правильности. Только в случае несовпадения всех трех результатов не удается обнаружить правильный ответ.

Способ контрольных соотношений. При решении многих задач вычисляемые функции могут быть связаны соотношением, которое не используется для получения результатов. С помощью таких соотношений удобно контролировать правильность вычислений. Например, если в ходе решения задачи вычисляются $\sin x$ и $\cos x$, то для контроля правильности определения этих функций можно использовать контрольное выражение $\sin^2 x + \cos^2 x = 1$. Но поскольку машина производит вычисления с некоторой, хотя и очень малой, погрешностью ε, отличной от нуля, то контрольное выражение должно быть преобразовано к виду $|1 - \sin^2 x - \cos^2 x| \leqslant \varepsilon$.

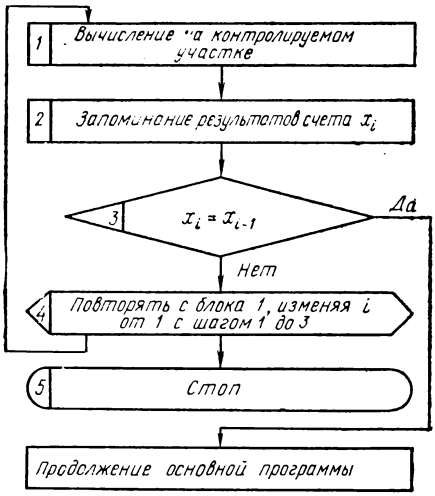


Рис. 9.2. Блок-схема алгоритма двойного-тройного счета

Часто в качестве контрольных соотношений используют метод подстановки. Например, если решают уравнение вида $f(x) = 0$ и в качестве корней получают значения a_i, то, подставляя значения корней в исходное уравнение, можно проверить правильность вычисления с помощью выражений $|f(a_i)| \leqslant \varepsilon$, где ε — допустимое значение погрешности.

Способ контрольных соотношений обеспечивает высокую надежность контроля и требует малых затрат машинного времени. Однако он может быть использован не всегда.

Все рассмотренные способы контроля правильности вычислений являются программными.

Наряду с программными способами контроля существуют также аппаратные, которые в данной книге рассматриваться не будут.

§ 9.3. ОБРАБОТКА РЕЗУЛЬТАТОВ РЕШЕНИЯ ЗАДАЧИ

После отладки программы и проверки правильности работы машины приступают к решению задачи. Результаты вычислений выводятся из машины, как правило, на перфораторы и различные печатающие устройства (см. гл. 1). На перфораторы результаты вычислений выводятся чаще всего в тех случаях, когда в дальнейшем они будут использоваться в качестве исходных данных для продолжения решения задачи, или для вывода буквенно-цифровой информации, если отсутствует алфавитно-цифровые печатающие устройства. Выходным документом в этом случае является перфоноситель с закодированной на нем в виде системы отверстий выходной информацией. В дальнейшем информация с перфоносителей может быть распечатана на печатающих устройствах, например на телетайпе, для непосредственного использо-

вания результатов вычислений. Вывод результатов на перфораторы иногда используется в случаях, когда печатающие устройства имеют очень малую скорость работы. При этом распечатка результатов производится автономно, что позволяет увеличить загрузку машины решением задач.

Печатающие устройства различного типа печатают выходную информацию на широкую (например, АЦПУ) или узкую (например БПМ) бумажные ленты.

На широкой бумажной ленте печатается числовая, текстовая или любая алфавитно-цифровая информация. Т. е. могут печататься не только числа, являющиеся результатами вычислений, но и текстовая информация (например, списки фамилий с указанием занимаемой должности и домашнего адреса), а также смешанная алфавитно-цифровая информация (например, наименование переменной и вычисленное ее числовое значение). Поскольку выходная информация при этом печатается в виде, близком к естественному, то ее расшифровка не представляет трудности. Для облегчения процесса обработки результаты могут быть выведены на печать в виде таблиц с указанием наименования переменных, записанных в каждой графе.

На узкой бумажной ленте печатаются, как правило, только числовые результаты, которые располагаются один под другим. Но отпечатанные таким образом результаты требуют дополнительной расшифровки и последующей обработки (построение таблиц, графиков и т. д.).

Результаты вычислений могут быть как отдельными числами, так и массивами чисел. Поскольку на узкую бумажную ленту выводятся только числа, то необходимо установить, прежде всего, каким переменным они соответствуют.

Числа печатаются в том порядке, в котором записаны соответствующие им переменные в операторе печати. Например, если на печать необходимо вывести три числа, то операторы печати имеют вид на языках АКИ и АЛГОЛ соответственно:

НАП — НА — БПМ — X, Y, A1 ∇ Δ

печать (*x*, *y*, *A*1);

Такая запись означает, что первое число является значением переменной *x*, второе — переменной *y*, третье — переменной A1.

Если оператор «*печать*» выполняется в цикле многократно, то первая тройка чисел дает значения *x*, *y* и *A*1 при первом выполнении цикла, вторая тройка чисел — значения этих же переменных при втором выполнении цикла, и т. д.

Если на печать выводятся массивы, то печатаются числа, соответствующие элементам массива, начиная с первого и кончая последним. Для двумерных массивов на печать выводятся элементы по строкам, т. е. сначала элементы первой строки массива, затем второй и т.д. Для удобства чтения после каждой строки делают пропуск. Напри-

мер, если выводится на печать массив

$$\begin{Vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \vdots & \vdots & \vdots & \vdots \\ a_{51} & a_{52} & a_{53} & a_{54} \end{Vmatrix},$$

то соответствующие ему числа будут расположены на бумажной ленте следующим образом:

a_{11}
a_{12}
a_{13}
a_{14} } Элементы первой строки массива.

Пропуск

a_{21}
a_{22}
a_{23}
a_{21} } Элементы второй строки массива.

$\vdots$

a_{51}
a_{52}
a_{53}
a_{54} } Элементы пятой строки массива.

Если в программе имеется несколько операторов печати, то указанные в них переменные выводятся на печать в том порядке, в котором выполняются эти операторы. Иногда заранее неизвестно, в каком порядке будут выполняться операторы печати, например в разветвленных программах трудно установить, каким переменным соответствуют напечатанные числа. В этом случае целесообразно выводить на печать какое-либо дополнительное число, которое будет являться признаком того, какой ветви вычислительного процесса соответствуют отпечатанные за ним результаты. Так, при решении квадратного уравнения на печать могут выводиться значения корней x_1 и x_2 этого уравнения, если $D = b^2 - 4ac > 0$, или действительная часть e и коэффициент при мнимой части f, если $D < 0$ (см. рис. 3.4).

Перед решением данного уравнения неизвестно D больше нуля или меньше нуля (при заданных значениях коэффициентов a, b, c), поэтому трудно сказать, чему соответствуют результаты вычислений — корням уравнения x_1 или x_2 или действительной и мнимой частям e и f. Чтобы ответить на этот вопрос, можно выводить на печать не только значения соответствующих переменных, но и некоторое вспомогательное число, которое может указывать номер ветви. Тогда оператор «*печать*», стоящий в одной ветви, будет выводить на печать N_1, x_1, x_2, а оператор «*печать*», стоящий во второй ветви, — N_2, e, f.

Здесь N_1 и N_2 несут признак того, что отпечатано. Если, например, $N_1 = 1$, а $N_2 = 2$, то на печать будет выведено число 1 и далее еще два числа, и можно сказать, что это корни уравнения; если же на печать будет выведено число 2, то следующие за ним числа будут соответствовать действительной e и мнимой f частям.

Целые числа на узкую бумажную ленту печатаются в виде целых чисел со знаком, а действительные числа в нормальной форме в виде мантиссы со знаком и порядка со знаком.

Для целых чисел на печать выводятся только значащие цифры, т. е. нули перед целым числом не печатаются. В крайнем правом разряде печатаются единицы, в следующем разряде — десятки и т. д. Знак числа печатается в крайнем левом разряде, отведенном для знака мантиссы.

Для чисел в нормальной форме на печать выводятся сначала мантисса со знаком, а затем знак порядка и двухразрядный порядок. Мантисса печатается в нормализованном виде, начиная с первой значащей цифры, т. е. следом за знаком печатается первая значащая цифра мантиссы (нуль целых и десятичная запятая опускаются). Порядок печатается обязательно в виде двухразрядного десятичного числа. Если величина порядка меньше десяти, то первая цифра нуль. Например,

$$+9702500 - 02$$
$$- \qquad 121.$$

Эти числа соответствуют действительному числу $0{,}9702500 \cdot 10^{-2}$ и целому числу —121.

После расшифровки результатов приступают к их обработке. Способы обработки результатов зависят от типа конкретной задачи, поэтому здесь рассматриваться не будут (наиболее часто по результатам решения задачи строят таблицы и графики, что можно выполнить вручную, или автоматически с использованием соответствующих печатающих устройств и графопостроителей).

ГЛАВА 10

ЧИСЛЕННЫЕ МЕТОДЫ И АЛГОРИТМЫ

В самых разнообразных областях техники приходится встречаться с математическими задачами, точное решение которых невозможно получить классическими методами или оно может быть получено в таком сложном виде, что станет совершенно неприемлемым для практического применения. Разрабатываемые вычислительной математикой численные методы носят в основном приближенный характер, позволяя тем не менее получить окончательный числовой результат с приемлемой для практических целей точностью.

В данной главе рассмотрены численные методы и принципы составления алгоритмов решения на ЦВМ некоторых математических задач, наиболее часто встречающихся в инженерной практике. К их числу относятся нелинейные алгебраические и трансцендентные уравнения, системы линейных и нелинейных уравнений, определенные интегралы, дифференциальные и разностные уравнения, задачи оптимизации. Оценка точности решения задач на ЦВМ имеет большое значение, поскольку объем вычислений приближенными методами велик и ошибки могут существенно исказить результат. В некоторых пределах увеличить точность можно за счет увеличения количества вычислений, а уменьшение временных затрат при решении задачи связано, до некоторых пределов, с уменьшением точности получаемого решения.

§ 10.1. ЧИСЛЕННОЕ РЕШЕНИЕ АЛГЕБРАИЧЕСКИХ И ТРАНСЦЕНДЕНТНЫХ УРАВНЕНИЙ

Необходимость отыскания корней алгебраических и трансцендентных уравнений встречается и в практике расчетов линейных систем автоматического регулирования, и при расчетах собственных колебаний машин и конструкций со многими степенями свободы, и при других технических расчетах.

Разные постановки задач и формы задания уравнений обусловили применение на практике нескольких методов отыскания их корней.

Общая форма задания таких уравнений такая: $f(x) = 0$.

Алгебраические уравнения n-й степени $a_0 x^n + a_1 x^{n-1} + a_2 x^{n-2} + \ldots + a_n = 0$ имеют n корней, коэффициенты a_i ($i = 0, 1, 2, \ldots, n$) могут быть и действительными, и комплексными числами.

Трансцендентные уравнения включают степенные алгебраические, тригонометрические и экспоненциальные функции от некоторого аргумента x, например, $\operatorname{arctg} x - x + 1 = 0$. Такие уравнения обычно имеют бесконечное множество корней.

Для решения нелинейных уравнений используют различные методы, из которых наиболее пригодными для реализации на ЦВМ являются методы последовательных приближений (итерационные), сво-

дящиеся к последовательному уточнению начального приближения для корня. Поскольку и для алгебраических, и для трансцендентных уравнений пригодны одни и те же методы уточнения приближенных значений действительных корней, далее будут совместно рассматриваться оба типа уравнений.

Отыскание начального приближения корней составляет проблему отделения корней.

Для алгебраического уравнения с действительными коэффициентами $a_0, a_1, \ldots, a_n$ $(a_0 > 0)$ верхняя граница положительных действительных корней $R_в^+$ определяется, по Лагранжу [19], формулой:

$$R_в^+ = 1 + \sqrt[k]{B/a_0}, \qquad (10.1)$$

где k $(k \geqslant 1)$ — номер первого из отрицательных коэффициентов полинома $f(x) = a_0x^n + a_1x^{n-1} + \ldots + a_n$; B — наибольшая из абсолютных величин отрицательных коэффициентов $f(x)$.

Все положительные корни x^+ уравнения $f(x) = 0$ удовлетворяют неравенству $x^+ \leqslant R_в^+$.

Нижнюю границу положительных действительных корней $R_н^+$, верхнюю и нижнюю границы отрицательных корней $R_в^-$, $R_н^-$ уравнения $f(x) = 0$ можно определить по формуле (10.1) для вспомогательных уравнений:

$$f_1(x) = x^n f(1/x) = 0;$$
$$f_2(x) = f(-x) = 0;$$
$$f_3(x) = x^n f(-1/x) = 0.$$

Если R_1, R_2, R_3 — верхние границы действительных положительных корней соответствующих вспомогательных уравнений, то для исходного уравнения выполняются неравенства:

$$(1/R_1) \leqslant x^+ \leqslant R_в^+;$$
$$-R_2 \leqslant x^- \leqslant -(1/R_3).$$

В трансцендентных и, в частности, тригонометрических уравнениях интервал отыскания корней определен заранее постановкой задачи, поскольку чаще всего речь идет о главных значениях аргументов. Чтобы отделить действительные корни таких уравнений, часто достаточно построить графики изменения функции $f(x)$ в выбранных границах изменения аргумента. Для уменьшения затрат времени шаг h изменения аргумента x при построении графика функции $f(x)$ можно выбрать первоначально достаточно большим, а затем, с целью уточнения формы графика в окрестностях корней $f(x)$, вычислять значения функции с дробным шагом h/k $(k > 1)$.

Часто, когда ставится задача отыскания начального приближения наибольшего (наименьшего) действительного корня, алгоритм формулируют так: изменять аргумент от верхней (нижней) границы действительных корней и вычислять значения функции $f(x)$ до тех пор, пока не будет обнаружено изменение знака функции $f(x)$.

Задача 10.1. Составить блок-схему алгоритма для отыскания начального приближения максимального положительного действительного корня уравнения

$$f(x) = 3x^8 - 5x^7 - 6x^3 - x - 9 = 0 \qquad (10.2)$$

с точностью 0, 1.

Верхнюю границу положительных действительных корней определяем по формуле Лагранжа (10.1) ($k = 1$, $B = 9$, $a_0 = 3$):

$$R_в^+ = 1 + 9/3 = 4.$$

Нижнюю границу положительных корней отыскиваем для вспомогательного уравнения: $9x^8 + x^7 + 6x^3 + 5x - 3 = 0$, где $k = 8$, $B = 3$, $a_0 = 9$,

$$R_1 = R_н^+ = 1 + \sqrt[8]{3/9} = 1{,}87.$$

Для решения поставленной задачи начнем вычислять значения $f(x)$ от значения $x = R_в^+$, последовательно изменяя значения аргумента с шагом h, равным по величине допустимой ошибке отделения корня: $h = -0{,}1$.

Блок-схема алгоритма приведена на рис. 10.1.

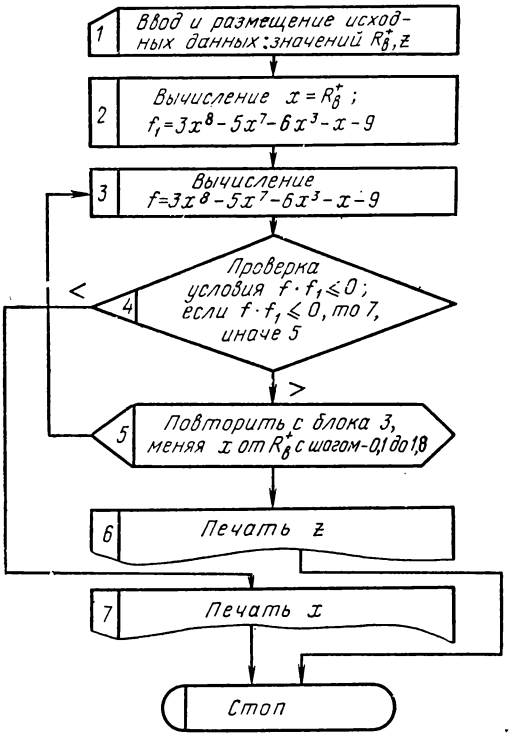


Рис. 10.1. Алгоритм отделения максимального действительного положительного корня алгебраического уравнения

Собственно алгоритм вычисления графика функции $f(x)$ и проверки условия изменения знака функции $f(R_в^+) \cdot f(x) \leqslant 0$, являющегося признаком существования корня, представляет собой цикл (блоки *3* ÷ *5*). Повторные вычисления в этом цикле выполняются с изменением аргумента x на величину шага $h = -0{,}1$. Повторения заканчиваются либо при выполнении условия $f(R_в^+) \cdot f(x) \leqslant 0$ (выход из цикла при отделении корня), с последующей печатью x, либо при достижении x значения $R_н^+ = 1{,}8$.

Несмотря на то что по формуле Лагранжа определяются границы действительных корней, в интервале от $R_в^+$ до $R_н^+$ корня может и не оказаться.

В качестве результата выполнения алгоритма в случае отсутствия действительных положительных корней в его схеме (рис. 10.1) предусматривается печать некоторого целого числа z, служащего в данном случае признаком отсутствия искомого корня. Если же искомый корень существует, то будет отпечатано его приближенное значение с недостатком, отличающееся от истинного значения корня не более чем на 0,1.

Соответствующая такому алгоритму АЛГОЛ-программа имеет вид:

```
begin real Rb, x, f1, f; integer z;
ввод (Rb, z);
x := Rb; f1 := 3 × x ↑ 8 — 5 × x ↑ 7 — 6 × x ↑ 3 — x — 9;
for x := Rb step — 0.1 until 1.7 do
```

```
begin f := 3×x↑8 — 5×x↑7 — 6×x↑3 — x — 9;
if f×f1 ⩽ 0 then go to l end;
печать (z); go to l1; l : печать (x);
l1 : end.
```

После отделения корня можно уточнить его методом последовательных приближений. Рассмотрим алгоритмы, соответствующие наиболее распространенным численным методам: последовательного деления отрезка, содержащего корень, пополам (метод половинного деления); касательных (метод Ньютона); итерации.

Уточнение корня, например уравнения $3x^8 - 5x^7 - 6x^3 - x - 9 = 0$, по методу половинного деления производится следующим образом.

Предположим, что найденная с помощью программы задачи 10.1 левая граница интервала, заключающего корень, будет $a = x$, а правая граница интервала будет $b = x + 0{,}1$. Последовательными уточнениями могут быть найдены теперь приближенные, более точные значения корня $x^{[1]}$, $x^{[2]}$, ..., $x^{[i]}$, ..., причем $x^{[1]}$ — середина отрезка (a, b), а $x^{[2]}$ — середина отрезка $(x^{[1]}, b)$ (рис. 10.2) и т. д.

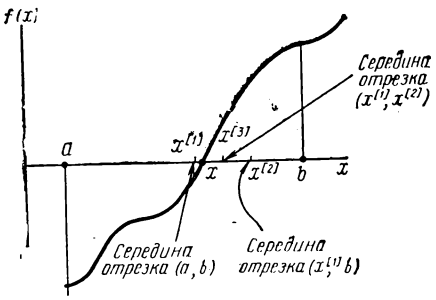


Рис. 10.2. Уточнение корня методом половинного деления

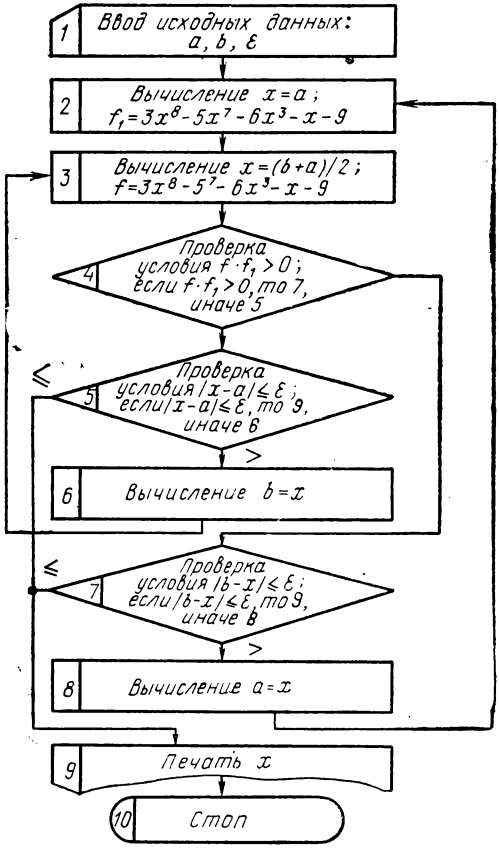


Рис. 10.3. Блок-схема алгоритма уточнения корня методом половинного деления

Задача 10.2. Составить блок-схему алгоритма для уточнения корня уравнения

$$3x^8 - 5x^7 - 6x^3 - x - 9 = 0$$

по методу половинного деления.

Блок-схема алгоритма (рис. 10.3) предусматривает циклическое вычисление значений функции $f(a)$ и $f(b)$ соответственно для левой и правой границ интер-

вала, заключающего корень (блоки 2 ÷ 8), до тех пор, пока не будет достигнута заданная точность ε.

После определения половины текущего интервала, содержащей корень (блок 4), переменным a или b придаются значения середины текущего интервала $x^{[i]}$ (блоки 8, 6), причем значение одной из них в каждом цикле сохраняется.

Исходными данными для алгоритма являются значения действительных переменных a, b, ε — соответственно первоначальные границы интервала существования корня и значение заданной погрешности уточнения корня. Промежуточные переменные f и f_1 принимают в процессе вычислений текущие значения функции $f(x)$ на концах интервала существования корня. Переменные a и b в процессе уточнения корня принимают значения текущих абсцисс границ интервала нахождения корня $x^{[1]}$, $x^{[2]}$, ..., $x^{[i]}$, ... Результатом реализации ал-

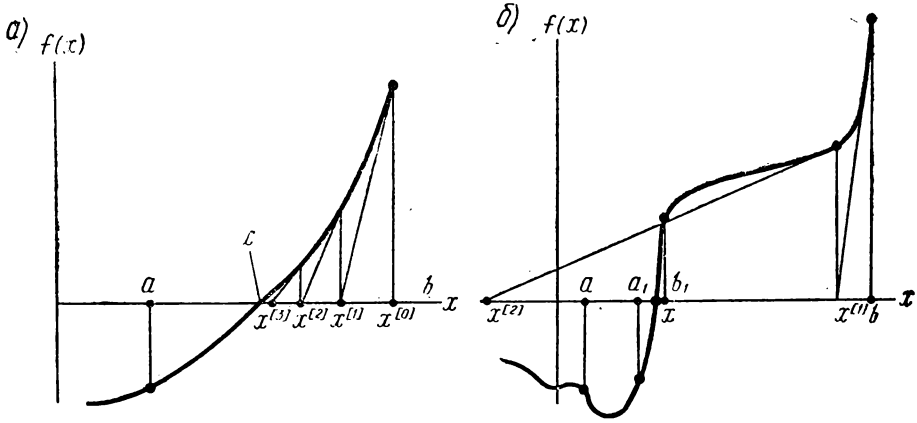


Рис. 10.4. Уточнение корня $f(x)$ методом Ньютона:
а — последовательное построение касательной; *б* — влияние формы $f(x)$ на сходимость

горитма является печать приближенного значения корня x, отличающегося от истинного его значения не более чем на величину ε. В том случае, когда корень будет точно в середине интервала, приводимый алгоритм также позволяет вычислить его с заданной погрешностью.

АЛГОЛ-программа уточнения корня методом половинного деления приведена ниже:

```
begin real a, b, epsilon, x, f, f1;
ввод (a, b, epsilon);
l : x : = a; f1 : = 3 × x ↑ 8 — 5 × x ↑ 7 — 6 × x ↑ 3 — x — 9;
l1 : x : = (b + a)/2; f : = 3 × x ↑ 8 — 5 × x ↑ 7 — 6 × x ↑ 3 — x — 9;
if f × f1 > 0 then go to l2 else if abs (x — a) ≤
epsilon then go to l3 else begin b : = x; go to l1 end;
l2 : if abs (b — x) ≤ epsilon then go to l3 else
begin a : = x; go to l end; l3: печать (x) end.
```

Уточнение корня уравнения методом касательных (методом Ньютона) производится с меньшими затратами времени следующим образом.

В известном интервале существования действительного корня функции (a, b) (рис. 10.4, *а*) выбирается начальное приближение корня $x^{[0]}$, и в точке $x^{[0]}$ проводится касательная к функции $f(x)$, точка $x^{[1]}$ пересечения касательной с осью абсцисс принимается за уточненное значение корня. Повторяя построение касательных в точках $x^{[1]}$, $x^{[2]}$,

..., $x^{[n]}$, ..., получим последовательное уточнение корня. Уточнение выполняется по формуле

$$x^{[n+1]} = x^{[n]} - f(x^{[n]})/f'(x^{[n]}).$$

При последовательных уточнениях

$$\lim_{n\to\infty} x^{[n+1]} = x.$$

Метод касательных в отличие от метода половинного деления использует информацию о текущей форме функции, которая задается в виде производных $f'(x^{[n]})$ в каждой точке последовательного уточнения $x^{[n]}$; это ускоряет процесс уточнения, но и ограничивает применимость формулы, поскольку для функций с пологими или параллельными оси абсцисс участками точка пересечения касательной с осью абсцисс может выйти за пределы участка, содержащего корень, и тогда уточнения не получится. Это показано на рис. 10.4, *б*, где в интервале (a, b) отделения корня начальное приближение $x^{[0]}$ выбрано неудачно, например, $x^{[0]} = a$.

Рекомендуется: 1) выбирать достаточно тесные границы корня (например, a_1, b_1); 2) проводить первую касательную всегда в той граничной точке интервала, заключающего корень, где знаки функции $f(x)$ и ее кривизны $f''(x)$ совпадают, т. е. где выполнено условие: $f(x)\cdot f''(x) > 0$, например в точке b (см. рис. 10.4, *а*).

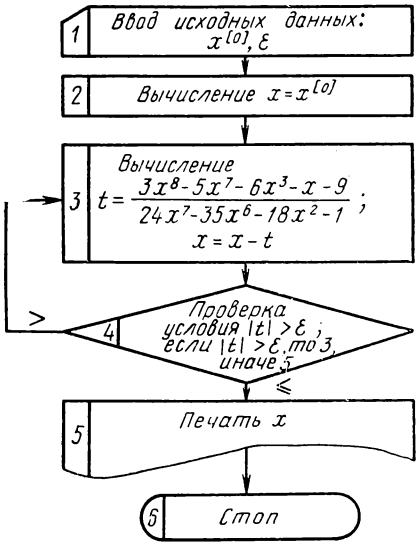


Рис. 10.5. Блок-схема алгоритма уточнения корня методом касательных

Задача 10.3. Составить блок-схему алгоритма для уточнения корня уравнения

$$3x^8 - 5x^7 - 6x^3 - x - 9 = 0$$

по методу касательных.

Алгоритм (рис. 10.5) представляет собой цикл итерационного типа (цикл типа пересчета), в котором число повторений заранее неизвестно: вычисления (блок *3*) и проверка условия $|x^{[n+1]} - x^{[n]}| > \varepsilon$ выполняются до тех пор, пока абсолютная величина приращения $|t| = |x^{[n+1]} - x^{[n]}|$ не станет меньше заданной погрешности ε.

Приращение t вычисляется по формуле

$$t = f(x)/f'(x)$$

и для уравнения (10.2).

$$t = \frac{3x^8 - 5x^7 - 6x^3 - x - 9}{24x^7 - 35x^6 - 18x^2 - 1}.$$

Переменной x первоначально присваивается значение начального приближения $x^{[0]}$ (блок 2), при дальнейших вычислениях эта переменная принимает уточняемые значения корня $x = x^{[1]}, x^{[2]}, \ldots, x^{[n]}, \ldots$. По окончании повторений x имеет значение корня, вычисленного с точностью ε, которое и печатается.

В соответствующей АЛГОЛ-программе обозначение $x^{[0]}$ заменено обозначением простой переменной $x0$:

```
begin real x0, x, t, epsilon;
ввод (x 0, epsilon);
x : = x0;
t : t : = (3 × x ↑ 8 — 5 × x ↑ 7 — 6 × x ↑ 3 — x — 9)/
(24 × x ↑ 7 — 35 × x ↑ 6 — 18 × x ↑ 2 — 1); x : = x — t;
if abs (t) > epsilon then go to t;
печать (x) end
```

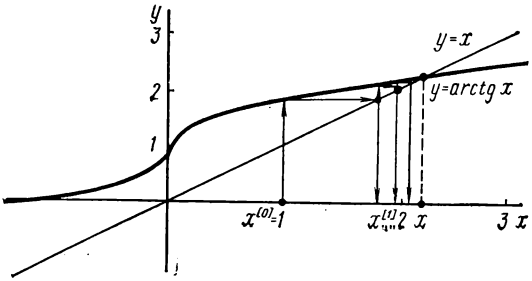


Рис. 10.6. Отделение и уточнение корня трансцендентного уравнения методом итерации

Метод простой итерации (последовательных приближений) для решения нелинейных уравнений заключается в следующем.

Уравнение

$$f(x) = 0$$

заменяется равносильным ему уравнением:

$$x = \varphi(x).$$

Выбрав начальное приближение для корня $x^{[0]}$ и подставив его в правую часть преобразованного уравнения, вычислим уточненное значение $x^{[1]}$ по формуле

$$x^{[1]} = \varphi(x^{[0]}).$$

Аналогично, $x^{[2]} = \varphi(x^{[1]})$, $x^{[3]} = \varphi(x^{[2]})$, ..., $x^{[i+1]} = \varphi(x^{[i]})$, ...

Установлено, что предел последовательности $x^{[0]}, x^{[1]}, \ldots, x^{[i+1]}, \ldots$, если он существует, является корнем x уравнения $f(x) = 0$:

$$\lim_{i \to \infty} x^{[i+1]} = \lim_{i \to \infty} \varphi(x^{[i]}) = x. \qquad (10.3)$$

Условием сходимости процесса итерации, т. е. условием существования предела (10.3), является соблюдение неравенства

$$|\varphi'(x)| < 1,$$

где $\varphi'(x) = d\varphi(x)/dx$.

Сходимость будет тем более быстрой, чем меньше величина $|\varphi'(x)|$. На рис. 10.6 поясняется геометрический смысл процесса итерации.

Для $(i-1)$-го значения $x^{[i-1]}$ вычисляется $\varphi(x^{[i-1]})$ и приравнивается $x^{[i]}$. Из рисунка видно, что при надлежащем выборе значения начального приближения $x^{[0]}$ значения $x^{[i]}$, $x^{[i+1]}$ монотонно приближаются к истинному значению корня. При некоторых условиях можно считать, что если

$$|x^{[i+1]} - x^{[i]}| \leqslant \varepsilon, \qquad (10.4)$$

то

$$|x - x^{[i+1]}| \leqslant \varepsilon,$$

т. е. для того чтобы вычислить значение действительного корня уравнения с точностью ε, достаточно добиться выполнения неравенства (10.4).

Задача 10.4. С точностью ε найти значение аргумента x, удовлетворяющего уравнению

$$x = \operatorname{arctg} x + 1.$$

Построим графики функций $y = x$ и $y = \operatorname{arctg} x + 1$, позволяющие отделить корень этого уравнения (см. рис. 10.6) и выбрать его начальное приближение. Из графика видно, что уравнение имеет один действительный корень.

Анализ производной

$$\varphi'(x) = \frac{d(\operatorname{arctg} x + 1)}{dx} = \frac{1}{1+x^2}$$

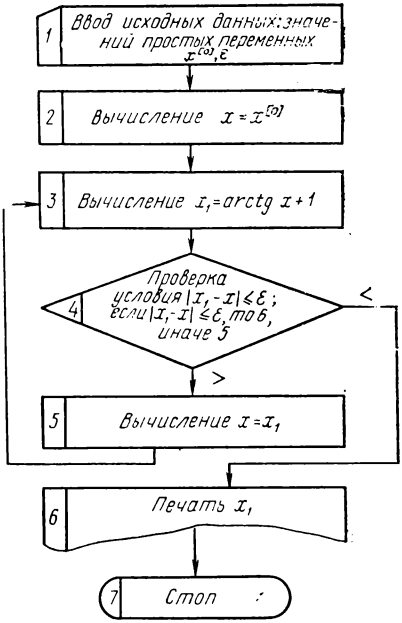


Рис. 10.7. Блок-схема алгоритма уточнения корня методом итерации

показывает, что простая итерация сходится для всех $x \neq 0$. Но в интервале $(-\infty, 0)$ сходимость не будет монотонной, т. е. величина $|x^{[i+1]} - x^{[i]}|$ не будет монотонно уменьшаться. Поэтому начальное приближение корня желательно выбрать в интервале $0 < x < \infty$ вблизи корня, например, $x^{[0]} = 1$.

Блок-схема алгоритма уточнения корня методом простой итерации содержит цикл уточнения корня по формуле: $x_1 = \operatorname{arctg} x + 1$, где x и x_1 — предыдущее и последующее приближение соответственно (рис. 10.7).

Окончание повторений (выход из цикла) определяет выполнение условия $|x_1 - x| \leqslant \varepsilon$.

Так же, как и в предыдущей АЛГОЛ-программе, обозначение $x^{[0]}$ заменено обозначением $x0$:

```
begin real x, x1, x0, epsilon;
ввод (x 0, epsilon);
x := x0;
l : x1 := arctan (x) + 1;
if abs (x1 — x) ⩽ epsilon then go to l1;
x := x1; go to l;
l1 : печать (x1) end
```

Во всех итерационных методах уточнения корней уравнений, представленных в данном параграфе, в качестве критерия окончания процесса вычислений выбрано следующее условие — модуль разности

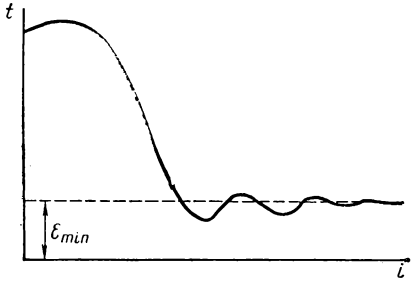


Рис. 10.8. Зависимость погрешности итерационного процесса от числа шагов итерации i

между двумя последовательными уточненными значениями корня становится меньше заданной погрешности ε:

$$t = |x^{[i+1]} - x^{[i]}| \leqslant \varepsilon.$$

При этом предполагается, что чем больше проделано уточнений, тем выше точность определения корня. Однако погрешности округления не имеют большого значения только для начальных шагов уточнения, после чего приближенные значения корня перестают приближаться к его истинному значению в среднем меньше, чем на некоторую величину ε_{min}.

Поведение величины t в процессе итерации (рис. 10.8) необходимо иметь в виду при выборе допустимой погрешности ε, которая не должна быть слишком малой, так как приблизиться к корню на величину $\varepsilon < \varepsilon_{min}$ не удается.

§ 10.2. РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ И НЕЛИНЕЙНЫХ УРАВНЕНИЙ

При решении систем линейных алгебраических уравнений на ЦВМ используют точные и приближенные методы.

Известный в алгебре точный метод решения систем n линейных алгебраических уравнений с n неизвестными вида

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1; \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n = b_2; \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \ldots + a_{3n}x_n = b_3; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_n = b_n \end{cases}$$

заключается в отыскании определителя системы Δ и определителей Δ_j ($j = 1, 2, \ldots, n$), получающихся из определителя Δ заменой его j-го столбца столбцом свободных членов системы, при этом $x_j = \Delta_j/\Delta$ (метод Крамера).

На ЦВМ метод Крамера используют редко, поскольку при больших n увеличивается число элементов определителей, место, отводимое в памяти ЦВМ для хранения значений их элементов, и время, необходимое для их вычисления. Не менее важным обстоятельством является и то, что вычисление корней системы x_j связано с делением на величину Δ, а это может привести к переполнению разрядной сетки ЦВМ и уменьшению точности вычислений корней при малых Δ. Поэтому при расчетах на ЦВМ чаще используют точные методы, основанные на исключении неизвестных (метод Гаусса).

Алгоритм отыскания решения системы n линейных алгебраических уравнений методом Гаусса состоит из следующих основных этапов.

1. Коэффициенты a_{kj} и свободные члены b_k системы размещаются в памяти ЦВМ в форме массива — прямоугольной матрицы вида

$$\mathbf{A} = \left\| \begin{array}{cccccc} a_{11} & a_{12} & a_{13} & \dots & a_{1n} & a_{1,\,n+1} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} & a_{2,\,n+1} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} & a_{n,\,n+1} \end{array} \right\|,$$

где $a_{1,\,n+1} = b_1$; $a_{2,\,n+1} = b_2$; ...; $a_{n\ n+1} = b_n$.

2. Первая строка матрицы делится на a_{11}, затем умножается на a_{k1} ($k = 2, 3, \dots, n$) и вычитается из k-й строки (последовательно из 2-, 3-, ..., n-й строк); элементы первого столбца матрицы, начиная с a_{21} и до a_{n1}, становятся нулями:

$$\mathbf{A}^{[1]} = \left\| \begin{array}{cccccc} 1 & a_{12}^{[1]} & a_{13}^{[1]} & \dots & a_{1n}^{[1]} & a_{1,\,n+1}^{[1]} \\ 0 & a_{22}^{[1]} & a_{23}^{[1]} & \dots & a_{2n}^{[1]} & a_{2,\,n+1}^{[1]} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & a_{n2}^{[1]} & a_{n3}^{[1]} & \dots & a_{nn}^{[1]} & a_{n,\,n+1}^{[1]} \end{array} \right\|,$$

где обозначение $\mathbf{A}^{[1]}$ принято для преобразованной матрицы.

3. Вторая строка матрицы $\mathbf{A}^{[1]}$ делится на $a_2^{[1]}$, а затем умножается на $a_{k,2}^{[1]}$ и вычитается из всех строк с $k = 3, 4, \dots, n$.

4. Действия, аналогичные перечисленным в п. 2, 3, повторяются до тех пор, пока подобная процедура не будет проделана с $(n-1)$-й строкой матрицы; при этом матрица будет приведена к виду:

$$\mathbf{A}^{[n-1]} = \left\| \begin{array}{cccccc} 1 & a_{12}^{[n-1]} & a_{13}^{[n-1]} & \dots & a_{1n}^{[n-1]} & a_{1,\,n+1}^{[n-1]} \\ 0 & a_{22}^{[n-1]} & a_{23}^{[n-1]} & \dots & a_{2n}^{[n-1]} & a_{2,\,n+1}^{[n-1]} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 & a_{n-1,\,n+1}^{[n-1]} \\ 0 & 0 & 0 & \dots & 0 & a_{n,\,n+1}^{[n-1]} \end{array} \right\|.$$

5. Элементы последней строки полученной матрицы позволяют вычислить значение

$$x_n = \frac{a_{n,\,n+1}^{[n-1]}}{a_{n,\,n}^{[n-1]}}.$$

6. Значение корня x_n используется для отыскания x_{n-1} при подстановке в $(n-1)$-ю строку треугольной матрицы $\mathbf{A}^{[n-1]}$, а затем последовательно вычисляются корни x_{n-2}, ..., x_1 по формулам

$$x_k = a_{k,\,n+1}^{[n-1]} - \sum_{j=k+1}^{n} a_{kj}^{[n-1]}\, x_j.$$

Метод Гаусса позволяет построить экономичный алгоритм, поскольку элементы матриц преобразуются по достаточно простым формулам и значения преобразованных элементов представляют собой текущие значения тех же переменных, что и значения элементов исходной матрицы.

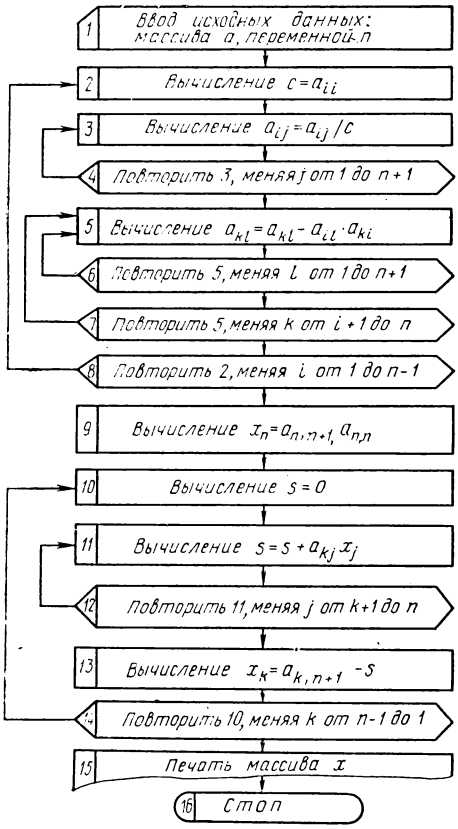


Рис. 10.9. Блок-схема алгоритма Гаусса

Блок-схема алгоритма, реализующего метод Гаусса, приведена на рис. 10.9. Исходными данными при составлении алгоритма является массив действительных переменных (коэффициентов и свободных членов системы) a, состоящий из n строк и $n+1$ столбца (порядок системы n).

Элементы прямоугольной матрицы A размещаются в одном массиве a, причем свободные члены b_k находятся в конце каждой k-й строки, т. е. обозначаются в массиве a индексами $a_{k,n+1}$. Промежуточные и окончательные значения элементов в процессе преобразования матрицы располагаются в том же массиве. Очередные значения диагональных элементов перед началом преобразования строк присваиваются промежуточной переменной c, с тем чтобы сохранить эти значения до окончания преобразования очередной строки матрицы.

Результаты вычисления неизвестных x_k накапливаются в одномерном массиве переменных x, содержащем n переменных. Нахождение сумм, необходимых для подсчета x_k, производится так же, как и в задаче 3.8, способом «накопления суммы». Значения частных и окончательной сумм присваиваются переменной s. При этом АЛГОЛ-программа имеет вид:

```
begin integer n; ввод (n);
begin real c, s; array a[1 : n, 1 : n+1], x[1 : n];
integer i, j, k, l; ввод (a);
for i := 1 step 1 until n-1 do begin c := a[i, i];
for j := 1 step 1 until n+1 do a[i, j] := a[i, j]/c;
for k := i+1 step 1 until n do for l := 1 step 1
until n+1 do a[k, l] := a[k, l] - a[i, l] × a[k, i] end;
```

```
x[n] := a[n, n+1]/a[n, n]; for k := n−1 step −1
until 1 do begin s := 0; for j := k+1 step 1
until n do s := s+a[k, j]×x[j]; x[k] := a[k, n+1]−s;
end; печать (x) end end
```

Сокращения времени решения на 20—30% по сравнению с «точными» методами получения решений систем линейных алгебраических уравнений для больших n добиваются путем применения итерационных методов (методов последовательных приближений).

Если итерационный метод сходится, то он дает следующие преимущества по сравнению с «точными» методами.

1. Если итерации сходятся достаточно быстро, то затраты времени на получение решения уменьшаются, так как общее число арифметических действий для алгоритма Гаусса пропорционально n^3, тогда как для каждой итерации число действий пропорционально n^2.

2. Погрешности округления при решении по методу итераций сказываются значительно меньше, чем при решении по методу Гаусса. Кроме того, случайные ошибки в процессе итераций самоисправляются на следующем шаге уточнения решений.

3. Метод итераций становится особенно выгодным при решении систем, значительное число коэффициентов которых равно 0; такие системы появляются, например, при решении краевых задач методом конечно-разностных уравнений (см. § 10.5).

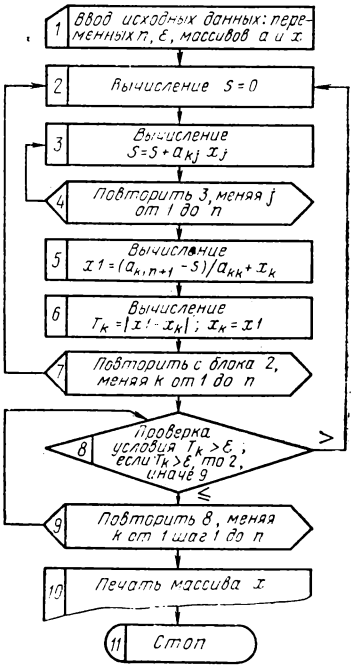


Рис. 10.10. Блок-схема алгоритма решения системы линейных алгебраических уравнений методом итераций Зейделя

Одним из наиболее распространенных итерационных методов решения систем линейных алгебраических уравнений является итерационный метод Зейделя, отличающийся от простой итерации (см. § 10.1) лишь тем, что каждое уточнение для x_k, вычисляемое по итерационной формуле,

$$x_k^{[l+1]} = \frac{b_k - \sum_{j=k}^{n} a_{kj} x_j^{[l]} - \sum_{j=1}^{k-1} a_{kj} x_j^{[l+1]}}{a_{k,k}} + x_k^{[l]}$$

использует уже уточненные значения $x_j^{[l+1]}$, $(j = 1, 2, \ldots, k-1)$.

Блок-схема алгоритма, реализующего метод Зейделя, приведена на рис. 10.10. При выполнении условий сходимости начальные приближения решения системы могут быть любыми. Исходными данными

для алгоритма будут: массив a коэффициентов и свободных членов системы, массив x начальных приближений корней $x_k^{[0]}$, заданная погрешность ε уточнения решения.

По структуре алгоритм представляет собой многоступенчатый циклический процесс (сложный цикл), во внешнем цикле итерационного типа (блоки 2÷8) размещены внутренние циклы вычисления сумм $\sum_{j=1}^{n} a_{kj}x_j$ для всех k (блоки 3, 4), вне этого цикла — цикл анализа невязки $|x_k^{[l+1]} - x_k^{[l]}|$, $(k = 1, \ldots, n)$, включающий проверку условия $|x_k^{[l+1]} - x_k^{[l]}| > \varepsilon$ (блоки 8, 9).

Записанная символами языка АЛГОЛ-60 программа отыскания приближенного решения системы линейных уравнений по Зейделю описывает ту же последовательность действий:

```
begin integer n; ввод (n);
begin real s, epsilon, x1; array a[1 : n, 1: n+1],
x [1 : n], T [1 : n]; integer j, k; ввод (a, x);
l: for k := 1 step 1 until n do begin s := 0;
for j = 1 step 1 until n do s := s+a [k, j] × x [j];
x1 := (a [k, n+1] — s)/a [k, k]+x [k]; T [k] := abs (x1 — x [k]);
x [k] := x1 end; for k := 1 step 1 until n do
if T [k] > epsilon then go to l; печать (x) end end
```

Перед решением системы линейных алгебраических уравнений на ЦВМ необходимо провести анализ существования и ожидаемой точности решения. О существовании и ожидаемой точности решения системы линейных алгебраических уравнений можно судить по величине Δ определителя системы:

$$\Delta = \begin{vmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{vmatrix},$$

где a_{ij} — коэффициенты системы.

Если $\Delta \neq 0$, то система имеет единственное решение. Вместе с тем, если абсолютная величина $|\Delta|$ мала, то полученное решение будет очень неточным (такие системы в алгебре называют *сильно косоугольными*, поскольку на графиках прямые, представляющие уравнения системы, почти параллельны друг другу). Устранить потерю точности в этом случае возможно только при вычислениях с большим числом значащих цифр, что сопряжено с трудностями при составлении алгоритмов.

Численное решение систем нелинейных алгебраических уравнений вида

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0; \\ f_2(x_1, x_2, \ldots, x_n) = 0; \\ \ldots \ldots \ldots \ldots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \end{cases}$$

(где $f_1, f_2, \ldots, f_n$ — любая нелинейная алгебраическая или трансцендентная функция) осуществляется в два этапа: 1) отделение корней (отыскание области существования корня), 2) уточнение корней с помощью методов последовательных приближений (методом Ньютона или методом итераций).

Отделение корней системы аналогично отделению корней нелинейных уравнений и может быть выполнено путем построения графиков функций $f_1, f_2, \ldots, f_n$. Однако при уточнении корней возникают трудности, связанные со сходимостью методов итерации.

Алгоритм уточнения корней системы нелинейных уравнений методом простой итерации приведен здесь для системы двух нелинейных уравнений.

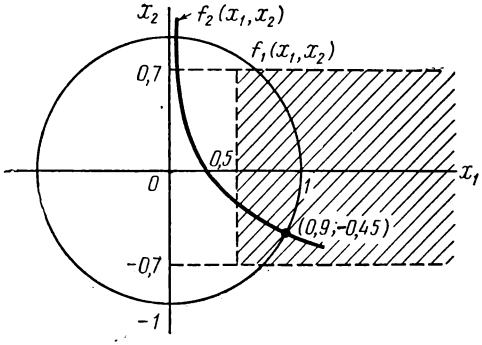


Рис. 10.11. Графическое решение системы нелинейных уравнений

Задача 10.5. Дана система уравнений:

$$\begin{cases} x_1^2 + x_2^2 = 1; \\ \ln x_1 + 2x_2 = -1, \end{cases}$$

графическое решение которых показано на рис. 10.11.

Начальные приближения для уточняемого корня $x_1^{[0]} = 0{,}9$, а $x_2^{[0]} = -0{,}45$.

Требуется, чтобы погрешность уточнения не превышала величины ε.

Выделяя x_1 и x_2 из уравнений, преобразуем систему к следующему виду:

$$x_1 = \sqrt{1 - x_2^2} = F(x_1, x_2);$$

$$x_2 = -1/2 - \ln x_1/2 = \Phi(x_1, x_2).$$

Последовательные приближения к точному значению корня вычисляются по формулам:

$$x_1^{[l+1]} = \sqrt{1 - (x_2^{[l]})^2};$$

$$x_2^{[l+1]} = -1/2 - \ln(x_2^{[l]})/2.$$

Условием сходимости итерационного процесса в данном случае будет:

$$\left|\frac{\partial F}{\partial x_1}\right| + \left|\frac{\partial \Phi}{\partial x_1}\right| < 1;$$

$$\left|\frac{\partial F}{\partial x_2}\right| + \left|\frac{\partial \Phi}{\partial x_2}\right| < 1$$

Здесь

$$\frac{\partial F}{\partial x_1} = 0; \quad \frac{\partial \Phi}{\partial x_1} = -\frac{1}{2x_1};$$

$$\frac{\partial F}{\partial x_2} = -\frac{x_2}{\sqrt{1 - x_2^2}}; \quad \frac{\partial \Phi}{\partial x_2} = 0.$$

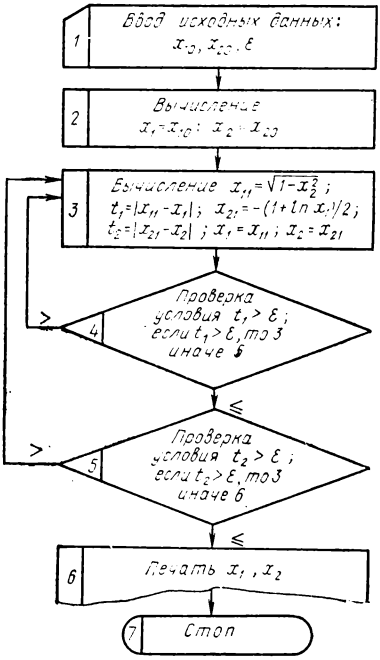


Рис. 10.12. Блок-схема алгоритма уточнения корней системы нелинейных уравнений

Исследование частных производных позволяет утверждать, что условие сходимости выполняется для всех точек, расположенных внутри области $0{,}5 < x_1 < \infty$; $-0{,}7 \leqslant x_2 \leqslant 0{,}7$ (на рис. 10.11 заштрихована). Поскольку выбранное начальное приближение корня расположено внутри этой области, то итерационный процесс уточнения корня сходится, т. е.

$$\lim_{l \to \infty} (x_1^{[l]}, x_2^{[l]}) = (x_1, x_2).$$

Блок-схема алгоритма приведена на рис. 10.12.

Перед очередным анализом близости двух последовательных уточнений (блоки *4* и *5*) предыдущие значения x_1 и x_2 заменяются только что уточненными значениями, располагавшимися во вспомогательных ячейках x_{11}, x_{2i} ОЗУ.

В рассматриваемой задаче удалось сразу найти форму представления уравнений системы, для которой итерационный процесс сходится достаточно быстро.

Представление уравнений в форме $x = f(x)$ неоднозначно, поэтому приходится проводить анализ с целью получения пригодной для итерации формы уравнений.

АЛГОЛ-программа уточнения корней системы нелинейных уравнений имеет вид:

```
begin real x 10, x 20, epsilon, t1, t2, x1,
x2, x11, x21; ввод (x 10, x 20, epsilon);
x1 := x10; x2 := x20;
l : x11 := sqrt (1 − x2 ↑ 2); t1 := abs (x11 − x1);
x21 := − (1 + ln (x1))/2; t2 := abs (x21 − x2);
x1 := x11; x2 := x21; if t1 > epsilon ∨ t2 > epsilon;
then go to l; печать (x1, x2) end
```

§ 10.3. ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ

Численные методы вычисления определенных интегралов не используют переходного этапа, формулирующего вид неопределенного интеграла, в который должны быть подставлены пределы интегрирования. Основу машинных алгоритмов составляет геометрический смысл определенного интеграла.

Определенным интегралом $y = \int_a^b f(x)\,dx$ называют площадь, ограниченную подынтегральной кривой, осью абсцисс и ординатами $f(a)$ и $f(b)$.

Широко известными алгоритмами такого рода, использующими для приближенного подсчета определенных интегралов их замену конеч-

ной суммой, являются методы прямоугольников, трапеций, Симпсона (парабол).

Все эти методы обеспечивают приближенное вычисление определенных интегралов, поскольку вместо кривой подынтегральной функции используются заменяющие ее кривые или ломаные линии, для которых вычисление ограниченной ими площади производится в соответствии с достаточно простыми формулами.

Метод прямоугольников. По методу прямоугольников кривая подынтегральной функции заменяется ломаной линией, отрезки которой параллельны оси абсцисс (рис. 10.13).

Формула прямоугольников имеет вид

$$y = \int_a^b f(x)\,dx \cong \frac{b-a}{n} \sum_{i=1}^{n} f(x_i),$$

где $f(x_i)$ — значения $f(x)$ в начале каждого i-го отрезка кусочной непрерывности интерполирующей ломаной $\varphi(x)$ (в этих точках значения $\varphi(x_i)$ и $f(x_i)$ одинаковы); n — число отрезков, на которые разбит интервал интегрирования (a, b).

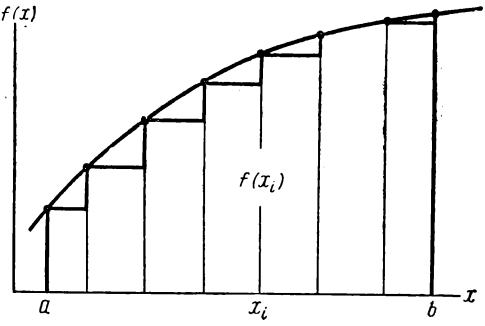


Рис. 10.13. Численное интегрирование методом прямоугольников

Исходными данными для составления алгоритма являются пределы интегрирования a, b и число отрезков n, а промежуточными результатами — $f = f(x_i)$, $(i = 1, 2, \ldots, n)$ и значения частных сумм y_i. Поскольку все промежуточные результаты могут потребоваться только на протяжении одного шага интегрирования, то удобно их значения рассматривать как значения простых переменных f, y; последовательные значения простой переменной x — абсциссы x_i.

Простая переменная h используется для обозначения ячейки для хранения значения шага интегрирования, равного $(b - a)/h$.

Схема алгоритма представлена на рис. 10.14.

Арифметические действия блока *2* отделены от арифметических действий блока *3*, поскольку блок *2* должен быть выполнен однократно, а блок *3* — многократно в цикле.

Для интеграла вида

$$y = \int_a^b \frac{e^x}{x}\,dx \qquad (10.5)$$

программа вычисления определенного интеграла методом прямоугольников с постоянным шагом имеет следующий вид:

```
begin real a, b, x, y, h; integer n;
ввод (a, b, n); h := (b — a)/n; y := 0;
for x := a step h until b do y := y + exp (x)/x;
y := y × h; печать (y) end
```

При вычислении определенного интеграла с переменным шагом в качестве исходных данных задаются величины приращения аргумента h_i в виде набора чисел. Значения h_i размещаются в массиве переменных h (здесь обязательно вычисление площади каждого элементарного прямоугольника).

Блок-схема алгоритма дана на рис. 10.15.

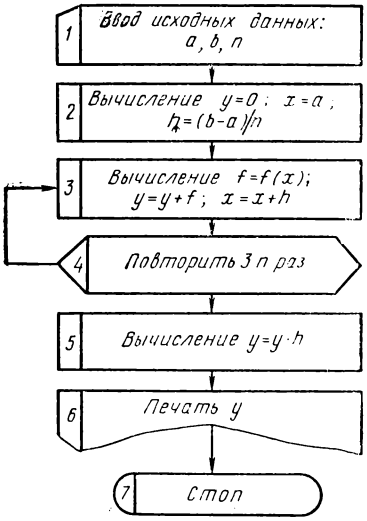


Рис. 10.14. Блок-схема алгоритма вычисления интеграла методом прямоугольников с постоянным шагом

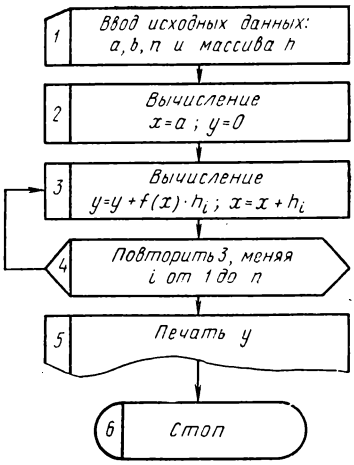


Рис. 10.15. Алгоритм интегрирования методом прямоугольников с переменным шагом

Расчетная формула имеет вид

$$y = \int_a^b f(x)\,dx \cong \sum_{i=1}^{n} h_i f(x_i)$$

АЛГОЛ-программа вычисления интеграла вида (10.5) методом прямоугольников с переменным шагом имеет вид:

```
begin integer n; ввод (n);
begin real a, b, x, y; array h [1 : n]; integer i;
ввод (a, b); y := 0; x := a;
for i := 1 step 1 until n do begin
y := y + h[i] × exp (x)/x; x := x + h [i] end;
печать (y) end end
```

Точность интегрирования ε методом прямоугольников с постоянным шагом h связана с величиной шага соотношением:

$$\varepsilon \approx h \quad (h \ll 1).$$

Метод Симпсона. Метод Симпсона обеспечивает большую точность вычисления определенного интеграла без увеличения шага h. Этот метод заключается в следующем: кривая подынтегральной функции заменяется кусочно-непрерывной линией, состоящей из отрезков квадратичных парабол; весь интервал интегрирования при этом разбивается на $n = 2m$ отрезков, число которых должно быть четным (рис. 10.16). Формула Симпсона для численного интегрирования с постоянным шагом имеет вид:

$$y = \int_a^b f(x)\,dx \cong \frac{b-a}{3n}\,[f_0 + f_{2m} + 4(f_1 + f_3 + \ldots + f_{2m-1}) + 2(f_2 + f_4 + \ldots + f_{2m-2})],$$

где $f_0 = f(a);\ f_{2m} = f(b);\ f_i = f(x_i);\ i = 1, 2, \ldots, 2_{2m-1}.$

Точность интегрирования по методу Симпсона для гладких функций связана с шагом интегрирования соотношением:

$$\varepsilon \approx h^3 \div h^4.$$

Время интегрирования пропорционально $\sqrt[5]{\varepsilon}$.

Схема алгоритма интегрирования по методу Симпсона сложнее схемы интегрирования по методу прямоугольников.

Оценить зависимость точности вычисления определенного интеграла от величины шага интегрирования для подынтегральных функций любой формы достаточно сложно. Ошибка в выборе шага интегрирования либо не обеспечит нужной точности, либо приведет к необоснованным затратам машинного времени. Поэтому в алгоритмах интегрирования предусматривается автоматический выбор шага интегрирования. В качестве исходного значения шага интегрирования h берется значение определяемое для метода Симпсона по формуле:

$$h \approx \sqrt[4]{\varepsilon}.$$

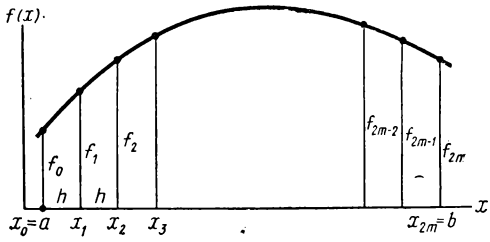


Рис. 10.16. Разбиение интервала при интегрировании по методу Симпсона

После подсчета значения $y^{[h]}$ интеграла с шагом h значение шага делится пополам, и вычисление значения $y^{[h/2]}$ интеграла ведется для шага $h/2$.

Если разность вычисленных значений интеграла будет $|y^{[h]} - y^{[h/2]}| \leqslant \varepsilon$, то $y^{[h/2]}$ считают значением $\int_a^b f(x)\,dx$, вычисленным с погрешностью ε.

Если $|y^{[h]} - y^{[h/2]}| > \varepsilon$, то продолжают «дробление» шага и вычисляют значения $y^{[h/4]}$, $y^{[h/8]}$, ...

Блок-схема алгоритма интегрирования по методу Симпсона с автоматическим выбором шага приведена на рис. 10.17. Она предусматривает вычисление ординат $f(a)$, $f(b)$ и ординат $f(x_i)$ по подпрограмме $f(x)$, дробление шага h, вычисление двух сумм s и INT, представ-

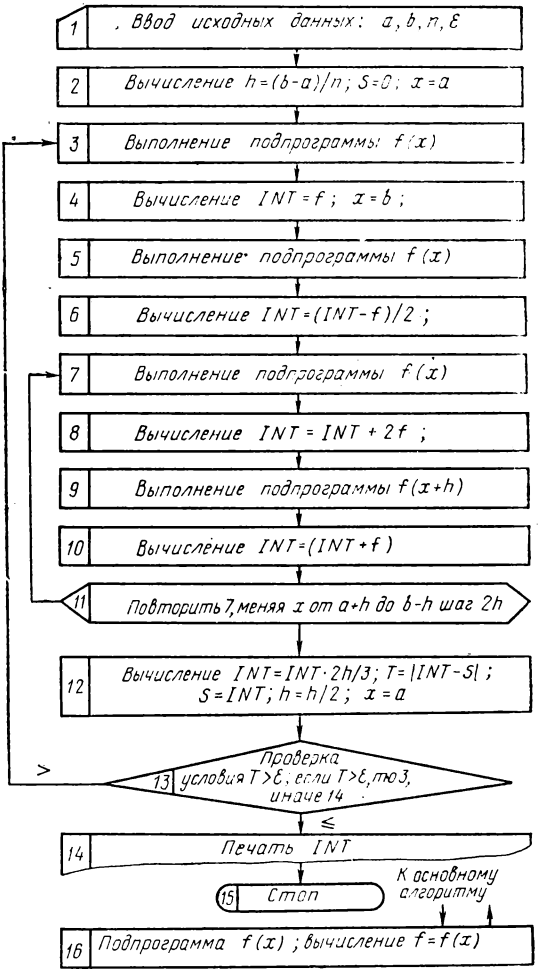


Рис. 10.17. Интегрирование по методу Симпсона с автоматическим выбором шага

ляющих собой соответственно значения интеграла y для полного и половинного шага, а также анализ точности вычисления интеграла по критерию $|INT - s| \leqslant \varepsilon$.

Для сокращения записи алгоритма и программы формула Симпсона эквивалентным преобразованием приводится к виду:

$$y \approx 2\frac{b-a}{3n}\left[\frac{f_0 - f_{2m}}{2} + 2(f_1 + f_3 + \ldots + f_{2m-1}) + f_2 + f_4 + \ldots + f_{2m}\right].$$

Это позволяет реализовать вычисление сумм ординат с нечетными (1, 3, ..., $2m-1$) и четными (2, 4, ..., $2m$) номерами в одном цикле (блоки $7 \div 11$). Параметр цикла (аргумент x) изменяется циклическим блоком, или оператором цикла соответствующей АЛГОЛ-программы, от значения $a+h$ до значения $b-h$ с шагом $2h$ (по нечетным значениям аргумента); нахождение четных ординат обеспечивается дополнительным увеличением аргумента на величину шага h внутри цикла.

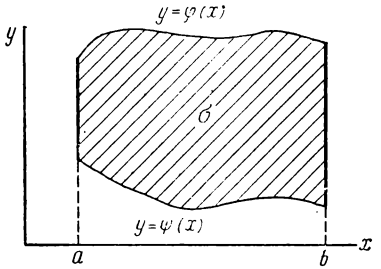


Рис. 10.18. Вид области интегрирования

Поскольку величина $x+h$ непосредственно определяется в каждом цикле и используется только при вычислении четной ординаты без записи в ячейку x, то простая переменная x сохраняет предыдущие значения, и добавление к этому значению в следующем цикле приращения $2h$ создает возможность вычисления $f(x)$ как значения нечетной ординаты.

Программа интегрирования методом Симпсона с автоматическим выбором шага приведена для интеграла вида (10.5):

```
begin real a, b, epsilon, s, INT, x, h, T;
integer n; real procedure f(x); real x;
f:=exp(x)/x; ввод (a, b, epsilon, n);
h:=(b-a)/n; s:=0, x:=a;
l: INT:=(f(a)-f(b))/2;
for x:=a+h step 2×h until b do
INT:=INT+2×f(x)+f(x+h);
INT:=INT×2×h/3; T:=abs(INT-s); s:=INT;
h:=h/2; x:=a; if T>epsilon then go to l;
печать (INT) end
```

При вычислении двойных интегралов

$$\iint_{\sigma} f(x, y)\, dx\, dy$$

пользуются кубатурными формулами. Для области интегрирования σ, ограниченной непрерывными однозначными кривыми $y=\varphi(x)$, $y=\psi(x)$ и вертикалями $x=a$, $x=b$ (рис. 10.18), $\iint_{\sigma} f(x, y)\, dy dx = \int_a^b dx \int_{\psi(x)}^{\varphi(x)} f(x, y)\, dy$, достаточно пользоваться только квадратурными формулами.

Применяя квадратурные формулы типа прямоугольников, получают

$$\iint\limits_{\sigma} f(x, y)\, dx\, dy \cong \sum_{i=1}^{n} \sum_{j=1}^{m_i} A_i B_{ij} f(x_i, y_j),$$

где A_i — шаг интегрирования по x; B_{ij} — шаг интегрирования по y для $x = x_i$; n — количество дискретных элементарных интервалов интегрирования по x в пределах от a до b; m_i — количество элементарных интервалов интегрирования по y для каждого сечения области σ при $x = x_i$.

Схема алгоритма приведена на рис. 10.19, где x, y, A, B — простые переменные, принимающие текущие значения x_i, y_j, A_i, B_{ij}, $m = \text{const}$.

Приведем программу на языке АЛГОЛ-60 вычисления двойного интеграла

$$s = \int_{a}^{b} \int_{y_1=\psi(x)}^{y_2=\varphi(x)} (3x^2 y + \sin y)\, dy\, dx$$

методом прямоугольников.

Функции $\psi(x)$ и $\varphi(x)$ заданы формулами:

$$\psi(x) = 1/x; \quad \varphi(x) = 3 + \cos x$$

и вычисляются с помощью процедур, описание которых дается в программе.

```
begin real a, b, s, A, y, x, B; integer n, m, i, j;
real procedure psi(x); real x; psi:=1/x;
real procedure fi(x); real x; fi:=3+cos(x);
real procedure f(x, y); real x, y; f:=3×x↑2×y+sin(y);
ввод (a, b, m, n); s:=0; A:=(b-a)/n; x:=a;
for i:=1 step 1 until n do begin B:=(fi(x)-psi(x))
/m; y:=psi(x); for j:=1 step 1 until m do begin
s:=s+A×B×f(x, y); y:=y+B end; x:=x+A end; печать(s) end
```

Для вычисления k-кратных интегралов (при больших k) рационально применение метода Монте—Карло, обеспечивающего малый рост затрат машинного времени с увеличением размерности задачи (см. § 10.7).

При вычислении несобственных интегралов с бесконечным промежутком интегрирования, если интеграл сходящийся, то его представляют в виде суммы интеграла с конечным пределом интегрирования b и несобственного интеграла:

$$\int_{a}^{\infty} f(x)\, dx = \int_{a}^{b} f(x)\, dx + \int_{b}^{\infty} f(x)\, dx$$

и выбирают число b столь большим, чтобы выполнялось неравенство

$$\left| \int_b^{\infty} f(x)\,dx \right| < \frac{\varepsilon}{2},$$

где $f(x)$ непрерывна при $a \leqslant x < \infty$.

Тогда определенный интеграл $\int_a^b f(x)\,dx$ нужно вычислять (по одной из квадратурных формул) с точностью до $\varepsilon/2$.

Обобщенная блок-схема алгоритма приведена на рис. 10.20.

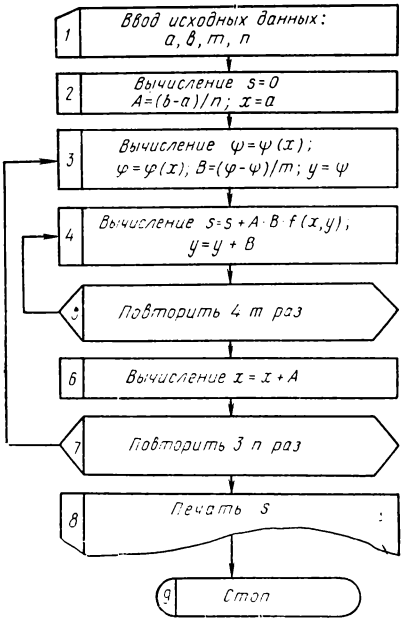


Рис. 10.19. Блок-схема алгоритма вычисления двойного интеграла методом прямоугольников

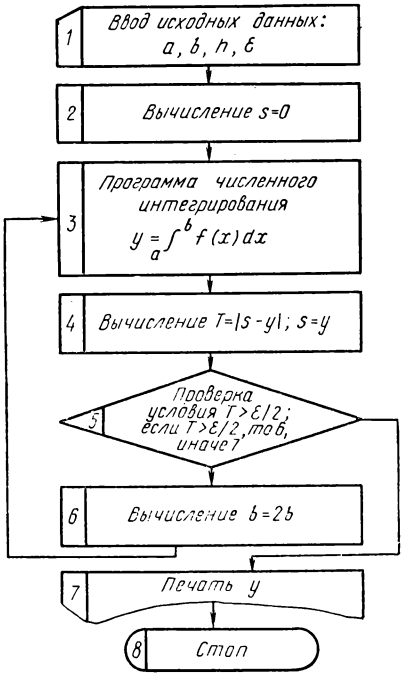


Рис. 10.20. Вычисление несобственного интеграла

Здесь переменная y принимает значения вычисленных по программе численного интегрирования определенных интегралов $\int_a^{b_i} f(x)dx$, а переменной s приписываются предыдущие значения интегралов с верхним пределом b_{i-1}. Переменной b придают значения увеличивающегося верхнего предела интегрирования, его увеличение получают удваиванием: $b_i = 2b_{i-1}$. В соответствии с приведенным алгоритмом и рассматривая процедуру вычисления определенного интеграла с автоматическим выбором шага как стандартную, к которой можно обра-

титься с помощью оператора процедуры *int* (a, b, h, E, y), для интеграла $\int_{0,2}^{\infty} (e^x/x)\,dx$ запишем АЛГОЛ-программу в форме:

```
begin real a, b, h, E, epsilon, y, s, T;
ввод (a, b, h, epsilon);
s := 0; E := epsilon/2;
l : int (a, b, h, E, y);
T := abs (s — y); s := y;
if T > E then begin b := 2 × b; go to l end;
печать (y) end
```

§ 10.4. ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ

Интегрирование в квадратурах может быть выполнено для немногих типов обыкновенных дифференциальных уравнений с заданными начальными условиями. Кроме того, часто замкнутые решения получаются громоздкими, и форма решений мало пригодна для инженерного исследования их качественного поведения в разнообразных условиях. Поэтому распространенным способом решения сложных дифференциальных уравнений является их численное интегрирование. Получаемое при этом решение уравнений представляет собой последовательность дискретных значений частного интеграла уравнения.

Методика численного решения дифференциального уравнения первого порядка

$$dy/dx = f(x, y) \tag{10.6}$$

с начальными условиями x_0, $y(x_0) = y_0$ связана с разложением решения в ряд Тэйлора в h-окрестности точки x_0:

$$y_1 = y(x_0 + h) = y(x_0) + hy'(x_0) + \frac{h^2}{2!} y''(x_0) + \dots .$$

При отбрасывании всех членов ряда, содержащих производные второго и высших порядков, получена формула

$$y_1 = y(x_0 + h) = y_0 + hf(x_0, y_0),$$

где $f(x, y)$ — правая часть дифференциального уравнения (10.6).

Пользуясь значением y_1, из разложения $y(x)$ в h-окрестности точки $x_1 = x_0 + h$ получим:

$$y_2 = y(x_0 + 2h) = y_1 + hf(x_1, y_1)$$

и далее

$$y_{i+1} = y_i + hf(x_i, y_i).$$

На рис. 10.21 показана форма численного решения $y(x_i)$, $(i = 1, 2, \ldots)$, получаемая с помощью таких вычислений. Этот метод называется *методом Эйлера.*

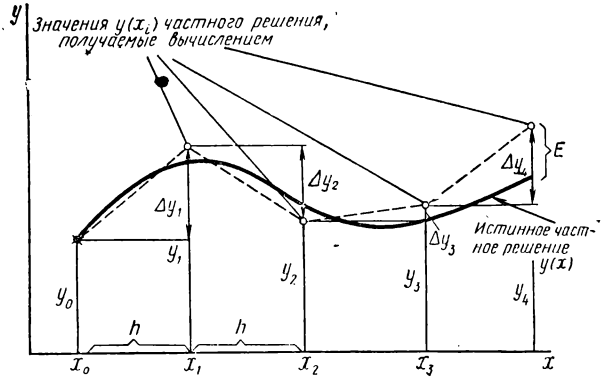


Рис. 10.21. Численное интегрирование дифференциального уравнения

При достаточно малой величине шага h метод Эйлера дает решение с большой точностью, так как погрешность решения близка к h^2 ($h \ll 1$) на каждом шаге интегрирования.

Исходными данными для интегрирования уравнения являются: начальные значения x_0, y_0, шаг интегрирования h и значение абсциссы x_k конца интервала интегрирования.

В алгоритме удобно предусмотреть вычисление промежуточных переменных x, y, принимающих значения x_i и y_i.

Блок-схема алгоритма приведена на рис. 10.22.

Например, программа интегрирования методом Эйлера для уравнения $dy/dx = 3xy - 5x + 2$ следующая:

```
begin real x0, y0, h, xk;
ввод (x 0, y 0, h, xk); x := x 0, y := y 0;
l: y := y + h × (3 × x × y − 5 × x + 2); x := x + h;
печать (x, y); if x < xk then go to l end
```

Недостатком метода Эйлера является замедление вычислений при выборе малой величины шага h, задающей точность решения.

Наиболее распространенным методом численного интегрирования дифференциальных уравнений служит метод Рунге—Кутта, обеспечивающий ускорение за счет большей точности вычисле-

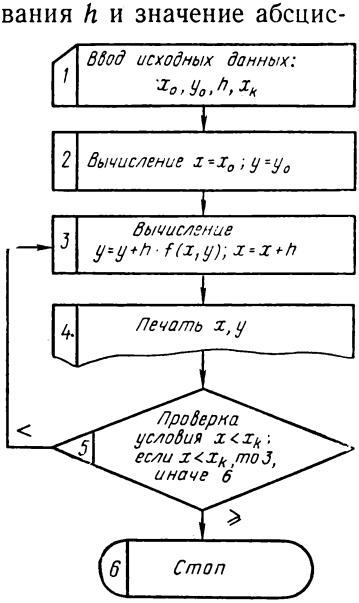


Рис. 10.22. Блок-схема алгоритма интегрирования дифференциальных уравнений методом Эйлера

ний на каждом шаге. Точность метода Рунге—Кутта оценивается величиной $E \approx h^5$. Уточнение достигается за счет специального подбора координат четырех точек, в которых вычисляется первая производная $f(x, y)$. Вместо первой производной $hf(x_i, y_i)$, используемой в формуле Эйлера, вычисляется усредненная первая производная $\tilde{f}_i$.

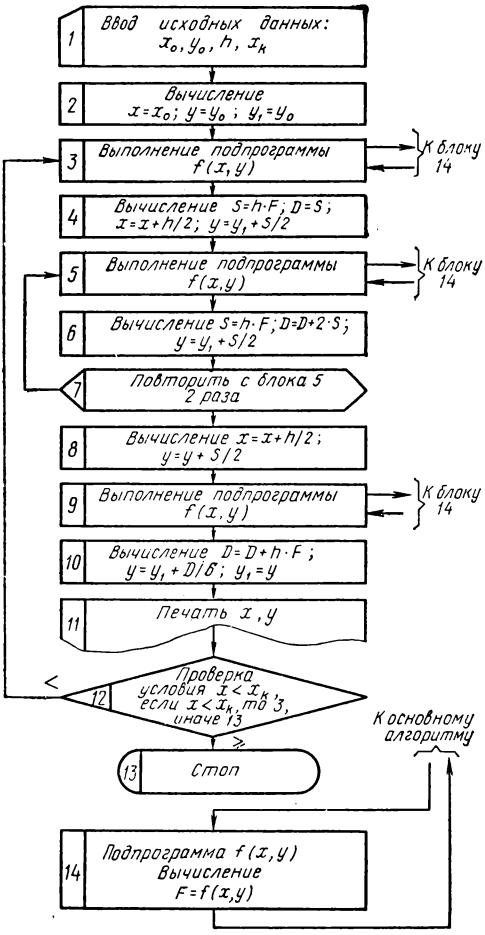


Рис. 10.23. Блок-схема алгоритма интегрирования дифференциального уравнения методом Рунге — Кутта

Формулы интегрирования по методу Рунге—Кутта имеют вид:

$$y_{i+1} = y_i + \tilde{f}_i,$$

где

$$\tilde{f}_i = (k_{1i} + 2k_{2i} + 2k_{3i} + k_{4i})/6;$$

$$k_{1i} = hf(x_i, y_i);$$

$$k_{2i} = hf\left(x_i + \frac{h}{2}, y_i + k_{1i}/2\right);$$

$$k_{3i} = hf\left(x_i + \frac{h}{2}, y_i + k_{2i}/2\right);$$

$$k_{4i} = hf(x_i + h, y_i + k_{3i}).$$

Здесь символом $f(\ldots)$, так же как и в описании метода Эйлера, обозначена правая часть решаемого дифференциального уравнения. За исключением этапа вычисления усредненной производной $\tilde{f}_i$, алгоритм интегрирования подобен алгоритму, составленному для метода Эйлера. На этапе вычисления усредненной производной удобно выделить в подпрограмму вычисление правых частей по формуле $F = f(x, y)$, придавая аргументам последовательно значения: $x = x_i$, $x_i + h/2$, $x_i + h$; $y = y_i$, $y_i + k_{1i}/2$, $y_i + k_{2i}/2$, $y_i + k_{3i}$.

Блок-схема алгоритма интегрирования дифференциального уравнения первого порядка с заданными начальными условиями x_0, y_0 приведена на рис. 10.23.

Например, программа интегрирования дифференциального уравнения методом Рунге—Кутта для уравнения $dy/dx = 3xy - 5x + 2$ следующая:

```
begin real x0, y0, h, xk, x, y,s, D;
integer i; real procedure F (x, y);
real x, y; F:=3×x×y−5×x+2;
ввод (x0, y0, h, xk);
x:=x0; y:=y0;
l: s:=h×F (x, y); D:=s;
for i:=1 step 1 until 2 do begin
s:=h×F (x+h/2, y+s/2); D:=D+2×s end;
x:=x+h; D:=D+h×F (x, y+s); y:=y+D/6;
печать (x, y); if x < xk then go to l end
```

При интегрировании дифференциальных уравнений высших порядков и систем уравнений высших порядков численными методами, уравнения необходимо представить в форме нормальной системы уравнений первого порядка. В нормальных системах каждое уравнение разрешено относительно входящей в него производной первого порядка своей функции.

Задача 10.6. Пусть дано уравнение второго порядка, полученное при моделировании деформации образца в процессе сжатия:

$$\frac{d^2y}{dt^2} = -\frac{\sigma Q p}{G}\left(\frac{dy/dt}{H_0 - y}\right)^B \left(\frac{y}{H_0}\right)^C \frac{1}{H_0 - y}.$$

Начальные условия t_0, $y_0 = y(t_0)$, $(dy/dt)_{t=t_0} = y_0'$.

σ, Q, G, p, H_0, B, C — постоянные, имеющие смысл физических параметров задачи.

После замены переменной $v = dy/dt$ уравнение преобразуется в нормальную систему:

$$\begin{cases} \dfrac{dy}{dt} = v = f_1(t, y, v); \\ \dfrac{dv}{dt} = -\dfrac{\sigma Q p}{G}\left(\dfrac{v}{H_0 - y}\right)^B \left(\dfrac{y}{H_0}\right)^C \dfrac{1}{H_0 - y} = f_2(t, y, v). \end{cases}$$

Интегрирование по формулам Рунге — Кутта производится для каждого из уравнений, т. е. для каждого следующего значения аргумента $t_{i+1} = t_i + h$ вычисляются значения $y_{i+1} = y_i + \tilde{f}_{1i}$; $v_{i+1} = v_i + \tilde{f}_{2i}$.

Блок-схема программного алгоритма интегрирования нормальной системы обыкновенных дифференциальных уравнений методом Рунге — Кутта приведена на рис. 10.24. Начальные значения переменных y_0 и $v_0 = y_0'$ и текущие значения этих переменных y_i и v_i размещаются в одномерном массиве A ($A_1 = v$, $A_2 = y$). Массивы D, s служат для размещения промежуточных результатов счета. В массиве F размещаются значения правых частей уравнений, соответственно приведенным выше обозначениям $F_1 = f_1(t, y, v)$, $F_2 = f_2(t, y, v)$. Кроме того, значения y_i и v_i запоминаются в ячейках $y1$ и $v1$ и используются затем для образования значений аргументов при вычислении правых частей уравнений.

Программа интегрирования нормальной системы дифференциальных уравнений соответствует блок-схеме, приведенной на рис. 10.24. Для вычисления правых частей введена процедура *PRAV*.

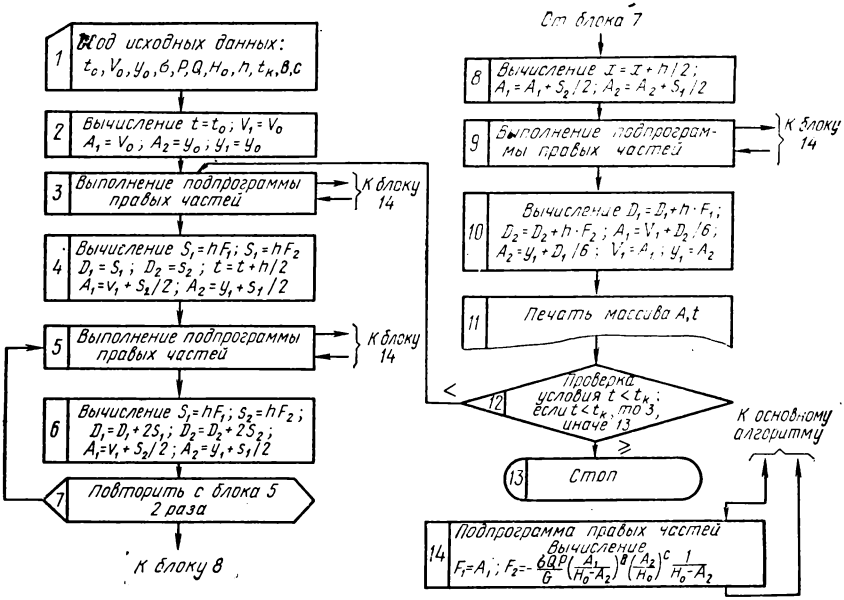


Рис. 10.24. Блок-схема алгоритма интегрирования нормальной системы дифференциальных уравнений методом Рунге — Кутта

```
begin real t0, v0, y0, sigma, p, Q, C, H 0, B, h, G, tk, t, v1, y1; integer i, j;
array A [1 : 2], s [1 : 2], D [1 : 2], F [1 : 2];
procedure PRAV (A, F); array A, F;
begin F [1] : = A [1]; F [2] : = — sigma × Q × p × (A [1]/
(H0 — A [2])) ↑ B × (A [2]/H0) ↑ C/G/ (H0 — A [2]) end;
ввод (t0, v 0, y 0, sigma, p, Q, H0, G, B, C, h, t k);
t : = t0; v1 : = v0; A [1] : = v0; A [2] : = y 0; y1 : = y0;
l : PRAV (A, F); s [1] : = h × F [1]; s [2] : = h × F [2];
D [1] : = s [1]; D [2]: = s [2]; t : = t + h/2; A [1] : = v1 +
s [2]/2; A [2] : = y1 + s [1]/2; for i : = 1 step 1 until 2 do begin
PRAV (A, F); for j : = 1 step 1 until 2 do begin
s [j] : = h × F [j]; D [j] : = D [j] + 2 × s [j] end;
A [1] : = v1 + s [2]/2; A [2] : = y1 + s [1]/2 end; t : = t + h/2;
A [1] : = A [1] + s [2]/2; A [2] : = A [2] + s [1]/2; PRAV (A, F);
D [1] : = D [1] + h × F [1]; D [2] : = D [2] + h × F [2];
A [1] : = v1 + D [2]/6; A [2] : = y1 + D [1]/6;
v1 : = A [1]; y1 : = A [2]; печать (t, A);
if t < tk then go to l end
```

§ 10.5. ДИФФЕРЕНЦИАЛЬНЫЕ УРАВНЕНИЯ С ЗАДАННЫМИ ГРАНИЧНЫМИ УСЛОВИЯМИ И ИХ ЧИСЛЕННОЕ РЕШЕНИЕ С ПОМОЩЬЮ РАЗНОСТНЫХ УРАВНЕНИЙ

К уравнениям такого типа приводят задачи об изгибе балок, внешней баллистики, расчета теплопроводности. Такие задачи называют *краевыми задачами для дифференциальных уравнений различных типов* (*обыкновенных дифференциальных уравнений*, *уравнений в частных производных*).

Краевая задача для обыкновенных дифференциальных уравнений ставится следующим образом: необходимо найти решение $y = y(x)$ уравнения

$$F(x, y, y', \ldots, y^{(s)}) = 0,$$

для которого значения y и ее производных

$$y_r^{(s)} = y^{(s)}(x_r), \quad s = 0, 1, \ldots, \sigma_r$$

в заданной системе граничных точек $x = x_r$ ($r = 1, 2, \ldots, k$; $k \geqslant 2$) удовлетворяют m независимым между собой краевым условиям, в общем случае нелинейным:

$$V_\nu\left(y_1, y_1', \ldots, y_1^{(\sigma_{1\nu})}, \ldots, y_k, y_k', \ldots, y_k^{(\sigma_{r\nu})}\right) = 0,$$
$$(\nu = 1, 2, \ldots, n),$$

где $\sigma_{r\nu}$ — максимальный порядок производной y_r в ν-м краевом условии.

Одним из наиболее распространенных методов решения двухточечной краевой задачи ($k = 2$) является метод конечных разностей.

Пусть задано уравнение

$$y'' + (1 + x^2)\, y = -1.$$

Краевые условия следующие:

$$y(-l/2) = y(l/2) = 0.$$

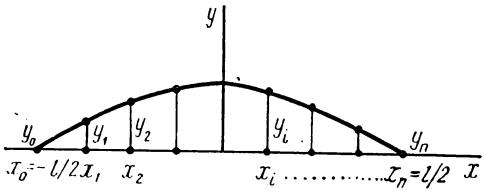


Рис. 10.25. Конечно-разностное приближение для двухточечной краевой задачи

Представление функции $y(x)$ членами ряда Тейлора в $\pm h$-окрестностях некоторой точки x_i дает:

$$y(x_i + h) = y(x_i) + hy'(x_i) + \frac{h^2}{2!} y''(x_i) + \ldots$$

$$y(x_i - h) = y(x_i) - hy'(x_i) + \frac{h^2}{2!} y''(x_i) - \ldots .$$

Складывая и вычитая эти уравнения, можно выразить конечно-разностное приближение для $y'(x_i)$ и $y''(x_i)$ соответственно в формах:

$$y'(x_i) = \frac{1}{2h}[y(x_i + h) - y(x_i - h)] + \ldots$$

$$y''(x_i) = \frac{1}{h^2}[y(x_i + h) - 2y(x_i) + y(x_i - h)] + \ldots .$$

Аналогично получают конечно-разностные приближения для производных высших порядков.

Разобьем интервал $(-l/2, l/2)$ на n участков длиной h (рис. 10.25). Заменяя конечными разностями исходное уравнение для всех точек x_i, а также учитывая заданные граничные условия, получаем систему

$n-1$ линейных алгебраических уравнений для определения $n-1$ неизвестных: $y_1, y_2, \ldots, y_{n-1}$; $y_0 = y\left(-\frac{l}{2}\right)$, $y_n = y\left(\frac{l}{2}\right)$.

$$\left.\begin{aligned}
-2y_1 + y_2 + h^2(1+x_1^2)\,y_1 &= -h^2 - y_0;\\
y_1 - 2y_2 + y_3 + h^2(1+x_2^2)\,y_2 &= -h^2;\\
&\cdots\cdots\cdots\cdots\cdots\cdots\\
y_{i-1} - 2y_i + y_{i+1} + h^2(1+x_i^2)\,y_i &= -h^2;\\
&\cdots\cdots\cdots\cdots\cdots\cdots\\
y_{n-2} - 2y_{n-1} + h^2(1+x_{n-1}^2)\,y_{n-1} &= -h^2 - y_n.
\end{aligned}\right\} \qquad (10.7)$$

Замена дифференциальных соотношений разностными дает ошибку второго порядка, т. е. при уменьшении шага h в два раза ошибка уменьшается в четыре раза.

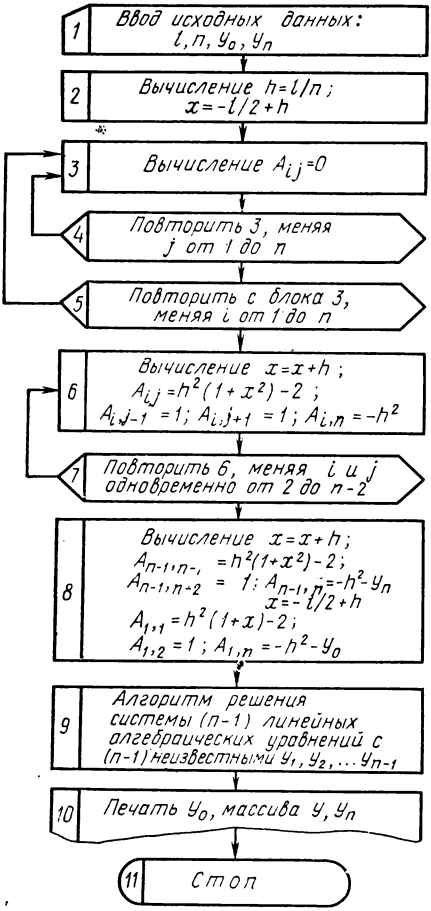


Рис. 10.26. Блок-схема алгоритма решения краевой задачи для обыкновенного дифференциального уравнения

Система (10.7) линейных алгебраических уравнений может быть решена одним из рассмотренных методов.

Таким образом, для решения приведенного уравнения с заданными граничными условиями необходимо составить матрицы коэффициентов и вектор-столбец свободных членов, а затем решить систему уравнений (10.7). Исходные данные представляют собой краевые условия y_0 и y_n, интервал между граничными точками ($l/2$, $l/2$), а также количество шагов n и позволяют вычислить матрицу коэффициентов и столбец свободных членов для решения краевой задачи в соответствии с приведенными конечно-разностными уравнениями.

Блок-схема алгоритма приведена на рис. 10.26.

Массив A предназначен для размещения коэффициентов системы и свободных членов. Блоки *3*, *4*, *5* описывают присваивание нулевых значений элементам матрицы коэффициентов, поскольку большинство этих коэффициентов при больших n равно 0. Блоки *6*, *7* представляют алгоритм вычисления коэффициентов матрицы раз-

ностных уравнений (10.7). Подпрограмма решения $n-1$ линейных алгебраических уравнений с $n-1$ неизвестным представляет собой любую из программ, рассмотренных в § 10.2, но предпочтительнее использование итерационного способа решения системы.

В приводимой здесь АЛГОЛ-программе решения краевой задачи процедура решения системы линейных алгебраических уравнений с $n-1$ неизвестными считается стандартной. Обращение к этой процедуре производится с помощью оператора процедуры *alg* ($n-1$, A, y), где A — идентификатор массива коэффициентов системы; y — идентификатор массива, содержащего значения корней системы после окончания решения.

```
begin integer n; ввод (n);
begin real l, y0, yn, h, x; array A[1 : n—1,
1 n], y[1 : n—1]; integer i, j; ввод (l, n, y0, yn);
h := l/n; x := —l/2+h;
for i:=1 step 1 until n—1 do for j:=1 step 1
until n do A[i, j]:=0; j:=1;
for i:=2 step 1 until n—2 do begin x:=x+h;
A[i, j]:=h↑2×(1+x↑2)—2; A[i, j—1]:=1;
A[i, j+1]:=1; A[i, n]:=—h↑2; j:=j+1 end;
x:=x+h; A[n—1, n—1]:=h↑2×(1+x↑2)—2;
A[n—1, n—2]:=1; A[n—1, n]:=—h↑2—yn; x:=—l/2+h;
A[1, 1]:=h↑2×(1+x↑2)—2; A[1, 2]:=1;
A[1, n]:=—h↑2—y0;
alg (n—1, A, y); печать (y0, y, yn) end end
```

Для оценки решения уравнений с помощью конечно-разностной модели можно провести несколько повторных решений задачи, меняя величину шагов h изменения аргументов до тех пор, пока расхождения решений не станут достаточно малыми. Кроме того, нужно иметь в виду, что получаемая с помощью конечно-разностной аналогии система линейных алгебраических уравнений сама может давать большую погрешность решения в силу малости величины определителя Δ.

§ 10.6. ПОСТАНОВКА И РЕШЕНИЕ ЗАДАЧ ОПТИМИЗАЦИИ

Задачи оптимизации представляют собой наиболее важный раздел прикладной инженерной математики. Их роль очень важна в экономическом планировании и подготовке производства, выборе оптимальных технологических процессов и оптимального расхода материалов.

Понятие «оптимума» в приложении к инженерным проблемам несколько шире понятия «оптимума» в математике непрерывных функций, поскольку не каждое явление или процесс возможно достаточно полно охарактеризовать одной непрерывной функцией даже и многих аргументов, так как природа этих аргументов может не позволить сделать их сравнимыми; учесть веса аргументов, с которыми они входят в функцию, при различных условиях затруднительно. Алгоритмизировать задачу выбора функции, подлежащей оптимизации,

невозможно. Поэтому выбор функции, подлежащей оптимизации (*целевой функции*), целиком возлагается на проектировщика устройства, исследователя процесса и т. д. Если подлежащая оптимизации функция выбрана, то ее аргументы всегда ограничены по величине.

Ограничения на аргументы вытекают из физического смысла задачи и могут быть наложены либо в форме неравенств, например $x_1 \geqslant 3$; $x_2 > 2$, ..., $x_1 + 2x_2 + 4 \ln x_3 \leqslant 2$ и т. д., либо в форме уравнений, например $x_1 + 3x_2 + 4x_3 = 900$.

Таким образом, формулировка задач оптимизации обычно включает в себя формулировку целевой функции $F(x_1, x_2, \ldots, x_j, \ldots)$ и ограничений, наложенных на аргументы x_j. Примером формули-

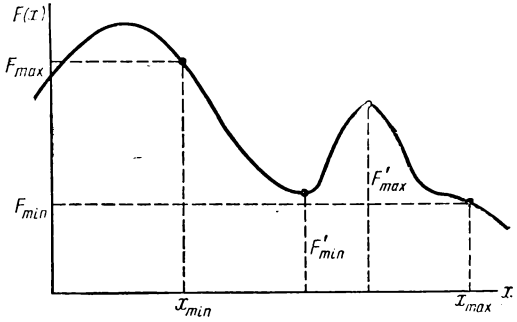


Рис. 10.27. Оптимизация при ограничениях.

ровки задачи оптимизации является формулировка задачи раскроя материала, рассмотренная в § 3.1. Там c — подлежащая минимизации (оптимизации) целевая функция, а система уравнений (1) и неравенств (2) представляет собой ограничения, наложенные на аргументы. Результатом решения задачи оптимизации будут значения аргументов, для которых достигается оптимальное значение функции.

В условиях оптимизации при ограничениях минимальные (или максимальные) значения целевой функции не обязательно совпадают с минимумом (максимумом) целевой функции и могут располагаться на границах интервалов изменения аргументов в точках $x_{\min}$, $x_{\max}$ (рис. 10.27). Здесь минимальное $F_{\min}$ и максимальное $F_{\max}$ значения функции не совпадают со значениями экстремумов функции $F'_{\min}$ и $F'_{\max}$. Подобные задачи часто называют *задачами минимизации (максимизации) функции*. Для функций одного аргумента $F(x)$ из условия $dF(x)/dx = 0$ находят корни уравнения, затем для корней, удовлетворяющих ограничениям $x_{\min} \leqslant x \leqslant x_{\max}$, отыскивают значения $F(x)$, далее находят значения $F(x_{\min})$ и $F(x_{\max})$ для граничных точек и сравнивают их с экстремальными значениями функции.

Рассмотрим алгоритм решения задачи минимизации целевой функции $F(x) = x^3 + \operatorname{ctg} x + e^x$, при ограничениях $0{,}1 \leqslant x \leqslant 1$. Сначала необходимо отыскать корни уравнения $dF/dx = 3x^2 - 1/\sin^2 x + e^x$, а затем проверить граничные точки $x = 0{,}1$ и $x = 1$.

Одним из возможных методов решения этой задачи на ЦВМ является метод последовательного перебора вариантов. Строя график

зависимости $F(x)$, сразу же получают форму функции $F(x)$, что позволяет исключить необходимость решения уравнения dF/dx.

Блок-схема алгоритма (рис. 10.28) включает в себя задание последовательных дискретных значений аргументов с некоторым шагом h, вычисление по «точкам» значений функций для заданных значений аргументов, определение частных минимумов среди значений $F(x)$ (см. задачу 3.10). Переменной PR придается произвольное значение, большее $F(x)$. Программа на языке АЛГОЛ имеет вид:

```
begin real PR, h, Fmin, F, x, x min; ввод (h, PR);
F min:=PR; for x:=0.1 step h until 1 do begin
F:=x↑3+cos(x)/sin(x)+exp(x); if F ⩾ F min
then go to l else begin F min:=F; x min:=x end;
l: end; печать (x min, F min) end
```

Метод последовательного перебора вариантов при отыскании максимума (минимума) функции многих переменных с достаточно высокой точностью требует для решения задачи перебора большого количества вариантов, которые невозможно выполнить даже с помощью ЦВМ.

Метод последовательного перебора целесообразно применять для приближенного определения положения оптимума. Вычисления при этом проводятся для больших приращений аргументов x (шагов h), и только после приближенного определения координат оптимальной точки в ее окрестностях отыскивается точный максимум (минимум).

Если установлено, что оптимум функции $F(x_1, x_2, \ldots, x_n)$ располагается внутри ограниченной области значений аргументов, то, чтобы его отыскать, нужно найти корни системы нелинейных уравнений:

$$\begin{cases} \partial F/\partial x_1 = 0; \\ \partial F/\partial x_2 = 0; \\ \ldots\ldots \\ \partial F/\partial x_n = 0. \end{cases}$$

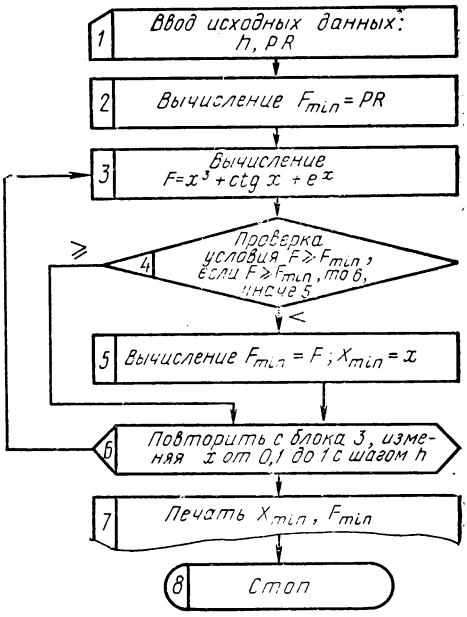


Рис. 10.28. Блок-схема алгоритма отыскания минимума функции $F(x)$ перебором ее значений

Поскольку грубое значение координат оптимума известно, то систему таких нелинейных уравнений можно решить итерационным методом (см. § 10.2), либо методом простого перебора с малым шагом. Однако более практичным для определения точного оптимума сложных

нелинейных функций нескольких переменных считают метод скорейшего спуска (метод градиента), который дает всегда сходящуюся последовательность координат оптимума.

По методу скорейшего спуска уточненные значения координат минимума:

$$\begin{cases} x_{1,\,i+1} = x_{1,\,i} - h_i\,(\partial F/\partial x_1)_i; \\ x_{2,\,i+1} = x_{2,\,i} - h_i\,(\partial F/\partial x_2)_i; \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ x_{n,\,i+1} = x_{n,\,i} - h_i\,(\partial F/\partial x_n)_i, \end{cases}$$

где h_i — величина i-го шага в направлении экстремума.

Величину h_i определяют по формуле

$$h_i = \frac{\sum\limits_{j=1}^{n} (\partial F/\partial x_j)_i^2}{2\sum\limits_{j=1}^{n}\sum\limits_{k=1}^{n}\left(\frac{\partial^2 F}{\partial x_j\,\partial x_k}\right)_i\left(\frac{\partial F}{\partial x_j}\,\frac{\partial F}{\partial x_k}\right)_i + \sum\limits_{i=1}^{n}\left(\frac{\partial^2 F}{\partial x_i^2}\right)_i\left(\frac{\partial F}{\partial x_j}\right)_i^2}.$$

Исходными данными для программной реализации метода наискорейшего спуска являются:

а) приближенные координаты оптимума (начальные значения) $x_{1,0}$; $x_{2,0}$; ...; $x_{j,0}$; ..., $x_{n,0}$; (хранятся в массиве x);

б) точность определения оптимума ε.

Для хранения промежуточных значений $x_{1,1}$, $x_{2,1}$, ... отводится массив $x1$.

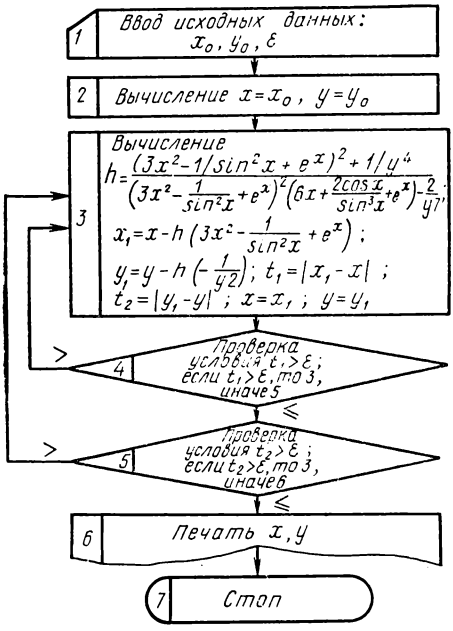


Рис. 10.29. Блок-схема алгоритма уточнения оптимума методом наискорейшего спуска

Блок-схема алгоритма для отыскания минимума функции

$$z = F(x,\ y) = x^3 + \operatorname{ctg} x + \\ + e^x + y + 1/y$$

с точностью ε приведена на рис. 10.29.

Если построить график функции, то можно определить, что оптимум функции $F(x, y)$ будет находиться внутри области $0{,}5 \leqslant x \leqslant 0{,}6$; $0{,}1 \leqslant y \leqslant 2$. Выбирают начальные приближения для отыскания минимума этой функции: $x_0 = 0{,}6$; $y_0 = 1{,}8$.

Алгоритм отыскания минимума методом наискорейшего спуска строится на основе последовательных приближений.

Частные производные вычисляют по формулам:

$$\frac{\partial F}{\partial x} = 3x^2 - \frac{1}{\sin^2 x} + e^x; \quad \frac{\partial^2 F}{\partial x^2} = 6x + \frac{2\cos x}{\sin^3 x} + e^x;$$

$$\frac{\partial F}{\partial y} = -\frac{1}{y^2}; \quad \frac{\partial^2 F}{\partial y^2} = \frac{2}{y^3}; \quad \frac{\partial^2 F}{\partial x\,\partial y} = \frac{\partial^2 F}{\partial y\,\partial x} = 0.$$

Величину h находят по формуле

$$h = \frac{(3x^2 - 1/\sin^2 x + e^x)^2 + (-1/y^2)^2}{(3x^2 - 1/\sin^2 x + e^x)^2 (6x + 2\cos x/\sin^3 x + e^x) + (-1/y^2)^2 (2/y^3)};$$

На языке АЛГОЛ-60 программу можно записать, вводя промежуточные переменные A, B, позволяющие сделать запись компактной:

```
begin real x0, y0, epsilon, x, y, t1, t2, h, A, B;
ввод (x0, y0, epsilon); x := x0; y := y0;
l: A = (3 × x ↑ 2 − 1/sin (x) ↑ 2 + exp (x));
B := A ↑ 2 × (6 × x + 2 × cos (x)/sin (x) ↑ 3 + exp (x));
h := (A ↑ 2 + 1/y ↑ 4)/(B + 2/y ↑ 7);
x1 := x − h × A; y1 := y + h/y ↑ 2; t1 := abs (x1 − x);
t2 := abs (y1 − y); x := x1; y := y1;
if t1 > epsilon ∨ t2 > epsilon then go to l;
печать (x, y) end
```

При уточнении оптимума функции методом наискорейшего спуска погрешность сначала уменьшается с увеличением количества шагов в направлении градиента, а затем уточняемые значения колеблются около точки истинного значения оптимума, как во всех итерационных методах, поэтому выбирать допустимую погрешность ε слишком малой величины не имеет смысла.

Составим алгоритм решения задачи оптимизации раскроя материала по минимуму функции c (см. задачу 3.2), аргументы которой принимают только дискретные значения.

Здесь для вычисления x, y, z, u, v существует лишь два уравнения:

$$\begin{cases} x = 560 - 3y/4 - 7z/12 - u/3; \\ v = 250 - y/4 - z/2 - 3u/4, \end{cases}$$

поэтому надо задавать различные значения y, z, u и вычислять соответствующие им x и v, рассматривая при этом варианты с разными соотношениями между задаваемыми величинами. Например, при первых значениях z и u вычисляются x, v и c для значений y, изменяющихся от 0 до 600 с шагом изменения, равным 100. Такой вычислительный процесс представляет собой простой цикл. Затем, изменяя z от 0 до 500 с шагом изменения, равным 100, повторяют этот простой цикл для

каждого нового значения z. В результате получают два цикла, вложенных один в другой, т. е. вычисляют x и v как функцию y, z, u:

$$\left\| \begin{array}{l} f(y_1, z_1, u_1)\, f(y_2, z_1, u_1)\, f(y_3, z_1, u_1) \ldots f(y_7, z_1, u_1) \\ f(y_1, z_2, u_1)\, f(y_2, z_2, u_1)\, f(y_3, z_2, u_1) \ldots f(y_7, z_2, u_1) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ f(y_1, z_6, u_1)\, f(y_2, z_6, u_1)\, f(y_3, z_6, u_1) \ldots f(y_7, z_6, u_1) \end{array} \right\|.$$

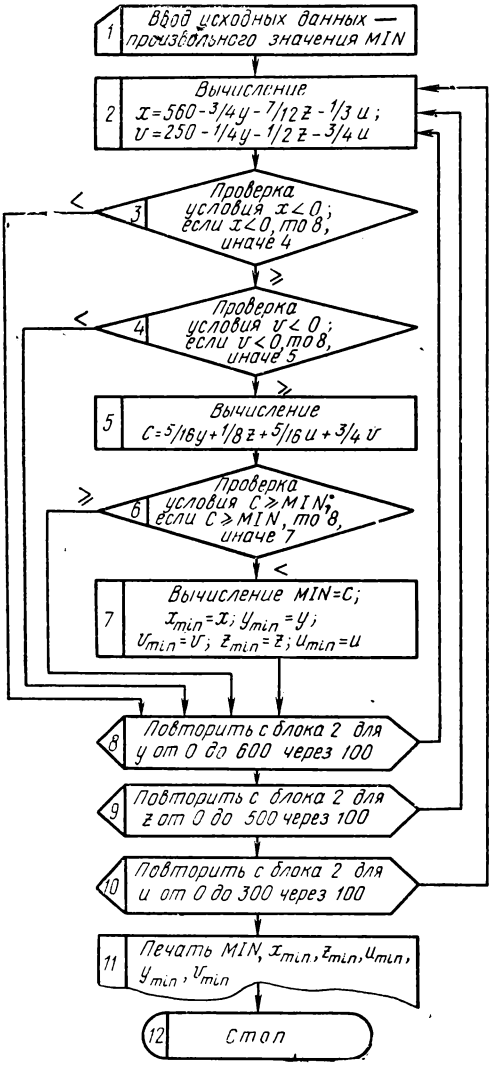


Рис. 10.30. Блок-схема алгоритма выбора оптимального варианта раскроя материала

Каждая строка этой матрицы вычисляется во внутреннем цикле при постоянных значениях переменных z и u, но при различных значениях y. Всякое новое повторение этого внутреннего цикла приводит к вычислению других строк при одном и том же значении u, но при различных значениях z.

Если повторить этот сложный (двойной) цикл, при всяком повторении меняя значение u, то можно получить набор матриц, вычисленных при различных значениях u. Этот третий цикл будет внешним циклом, т. е. будет охватывать двойной цикл, в результате которого вычисляется матрица.

Таким образом, вычислительный процесс представляет собой сложный цикл с тремя вложенными один в другой циклами.

Кроме вычисления x, v, c, необходимо выбрать вариант значений x, y, z, u, v, при которых получается минимальное значение c. Для этого можно сравнивать минимальное значение из всех предыдущих значений MIN с каждым вновь получаемым значением c. Если сравниваемое значение c будет больше MIN, то данный вариант не представляет интереса и, из-

меняя значения аргумента y, надо переходить к вычислению новых значений x, v и c. Если же значение c будет меньше MIN, то оно является наименьшим из всех ранее рассмотренных значений, и тогда надо считать MIN, равным этому значению c.

Перед началом цикла необходимо вычислить MIN, равный какому-то очень большому числу, для того чтобы при первом же сравнении этого MIN со значением c получился MIN, равный этому значению c.

Блок-схема такого вычислительного процесса представлена на рис. 10.30.

Здесь блоки *8* и *9* проверяют условия $x < 0$ и $v < 0$. Если x и v отрицательные, то решение не имеет смысла, поэтому сразу же надо переходить к изменению аргумента y и возвращаться к началу цикла. После окончания цикла будут определены x_{min}, y_{min}, z_{min}, u_{min}, v_{min}, MIN, которые выводятся на печать.

Более точное и быстрое решение этой задачи осуществляется методом линейного программирования, при этом АЛГОЛ-программа имеет вид:

```
begin real x, y, v, u, z, MIN, x min, y min, v min,
u min, z min; ввод (MIN);
for u := 0 step 100 until 350 do
for z := 0 step 100 until 550 do
for y := 0 step 100 until 650 do begin
x := 560 — 3 × y/4 — 7 × z/12 — u/3, v := 250 — y/4 — z/2 — 3 × u/4;
if x < 0 ∨ v < 0 then go to l else c := 5 × y/16 + z/8 + 5 × u/16 + 3 × v/4;
if c ⩾ MIN then go to l; MIN := c; x min := x; y min := y;
z min := z; v min := v; u min := u; l : end; печать (x min,
y min, z min, v min, u min, MIN) end
```

§ 10.7. МЕТОДЫ МОНТЕ — КАРЛО

Методами Монте—Карло называют методы, основанные на использовании приемов статистической выборки для приближенного решения той или иной математической задачи. Отличительной чертой методов Монте—Карло является их экспериментальный характер. Не для всякой задачи методы эти известны, и к конкретным задачам не обязательно существует единственный подход при их помощи. Общая методика решения задачи с помощью методов Монте—Карло основана на следующих положениях.

Пусть необходимо вычислить значения многомерной функции

$$F = F(x_1, x_2, \ldots, x_n).$$

На ЦВМ можно вычислить только некоторые значения этой функции для отдельных значений дискретно меняющихся аргументов x_{1j}, x_{2j}, x_{3j}, ..., x_{nj}. Для функции же двух переменных для сравнительно точного суждения о ее форме необходимо вычислить $s \times t$ значений, где s — число значений аргумента x_1; t — число значений аргумента x_2. Для функций, имеющих большее количество измерений, объем

расчетов существенно возрастает. Вместе с тем во многих практических случаях для сравнительно точного представления о форме функции F нет необходимости перебирать все значения аргументов ввиду того, что их влияние на величину функции можно оценить и не применяя равного шага изменения аргумента.

Использование неравных приращений h_j аргументов x при машинных вычислениях функций создает определенные трудности: необходимо хранение большого количества значений аргументов в памяти машины и решение проблемы выбора тех значений аргументов, для которых существенно вычисление значений функции. Решение этих проблем в соответствии с методами Монте—Карло состоит в случайном выборе значений аргументов x_{1j}, x_{2j}, ..., x_{nj} для каждого j-го набора аргументов.

Случайные значения аргументов суть не что иное как случайные числа. На ЦВМ получить случайные числа можно с помощью довольно простых алгоритмов.

Получение равномерно распределенных случайных чисел основано на имитации хаотического перемешивания их разрядов в следующей последовательности:

а) берутся два произвольных числа A и B, состоящих из $2N$ цифр, и перемножаются;

б) в полученном произведении отбрасываются по N цифр с начала и конца;

в) оставшиеся в произведении $2N$ цифры образуют первое случайное число ξ_1;

г) случайное число ξ_1 совместно с B используется для получения тем же методом второго случайного числа ξ_2 и т. д.

В результате выполнения алгоритма получается бесконечная последовательность чисел, в которой 1-е, 2-е, ... числа могут рассматриваться как случайные (в дальнейшем числа этой последовательности повторяются с периодом 2^{2N}), равномерно распределенные в интервале (0,1). Эту последовательность равномерно распределенных случайных чисел используют в качестве неравноотстоящих координат аргументов x_{1j}, x_{2j} ...., x_{nj}.

Программы-генераторы случайных чисел обычно записывают в коде машины и используют как стандартные.

Равномерно распределенные случайные числа используют для генерирования случайных чисел, подчиняющихся законам распределения, отличным от равномерного, и для вычисления кратных интегралов больших размерностей методом Монте—Карло.

Вычисление кратных интегралов. Пусть требуется вычислить k-кратный интеграл: $I = \iint\limits_{(s)} \ldots \int f(x_1, x_2, \ldots, x_k)\, dx_1 dx_2 \ldots dx_k$.

Число I есть k-мерный объем прямого цилиндроида, построенного на основании s и ограниченного сверху поверхностью $f(X)$, где $X = (x_1, x_2, \ldots, x_k)$.

Интеграл I преобразуют таким образом, чтобы новая область интегрирования целиком содержалась внутри единичного k-мерного куба.

Если предположить, что область s расположена в k-мерном параллелепипеде:

$$a_i \leqslant x_i \leqslant A_i,$$
$$i = 1, 2, \ldots, k$$

и ввести замену переменной:

$$x_i = a_i + (A_i - a_i)\,\xi_i,$$

то k-мерный параллелепипед преобразуется в k-мерный единичный куб

$$0 \leqslant \xi_i \leqslant 1,$$
$$i = 1, 2, \ldots, k$$

(рис. 10.31).

Новая область интегрирования σ будет целиком расположена внутри такого куба, а интеграл будет преобразован к виду:

$$I = \underset{(\sigma)}{\int\int} \ldots \int F(\xi_1, \xi_2, \ldots, \xi_k)\, d\xi_1, d\xi_2, \ldots, d\xi_k,$$

где

$$F(\xi_1, \xi_2, \ldots, \xi_k) = (A_1 - a_1)(A_2 - a_2) \ldots (A_k - a_k) \times$$
$$\times f\left(a_1 + (A_1 - a_1)\,\xi_1,\ a_2 + (A_2 - a_2)\,\xi_2, \ldots, a_k + (A_k - a_k)\,\xi_k\right).$$

Для вычисления интеграла методом Монте — Карло выбирают k равномерно распределенных на отрезке (0,1) последовательностей случайных чисел:

$$\xi_1^{(1)}, \xi_2^{(1)}, \xi_3^{(1)}, \ldots, \xi_m^{(1)}, \ldots;$$
$$\xi_1^{(2)}, \xi_2^{(2)}, \xi_3^{(2)}, \ldots, \xi_m^{(2)}, \ldots;$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$\xi_1^{(k)}, \xi_2^{(k)}, \xi_3^{(k)}, \ldots, \xi_m^{(k)}, \ldots.$$

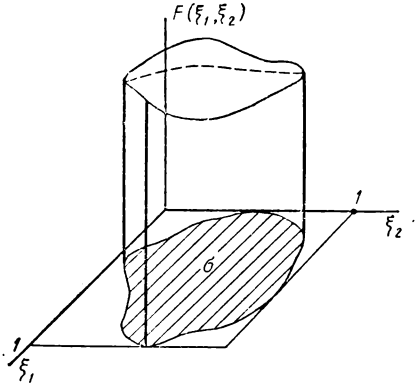


Рис. 10.31. Область интегрирования после преобразования

При этом точки M_i, имеющие координаты $\xi_i^{(1)}, \xi_i^{(2)}, \ldots, \xi_i^{(k)}$ $(i = 1, 2, \ldots)$, можно рассматривать как случайные. Выбрав достаточно большое число N таких точек $M_1, M_2, \ldots, M_N$, проверяют, какие из них принадлежат области интегрирования σ (первая категория) и какие не принадлежат ей (вторая категория).

Пусть $M_i \in \sigma$ при $i = 1, 2, \ldots, n$ и $M_i \notin \sigma$ при $i = n + 1, n + 2, \ldots, N$. Для удобства изменена нумерация точек, поскольку вырабатываемые последовательно случайные числа могут попадать в область интегрирования σ по случайному закону. Тогда средняя

высота цилиндроида:

$$y_{\text{ср}} = \frac{1}{n} \sum_{i=1}^{n} F(M_i),$$

а искомый интеграл

$$I \approx y_{\text{ср}}\, \sigma = \frac{\sigma}{n} \sum_{i=1}^{n} F(M_i), \tag{10.8}$$

где под σ понимается k-мерный объем области интегрирования σ.

Если вычисление σ затруднительно, то можно считать (для больших N)

$$\sigma \approx n/N$$

и тогда

$$I \approx \frac{1}{n} \sum_{i=1}^{n} F(M_i). \tag{10.9}$$

Задача 10.6. Составить алгоритм вычисления методом Монте — Карло двойного интеграла

$$\iint\limits_{(s)} (x_1^2 + \sin^2 x_2)\, dx_1\, dx_2.$$

Область s ограничена окружностью

$$x_1^2 + x_2^2 = 1.$$

Ясно, что эта область расположена внутри квадрата со сторонами 2×2.

Преобразование координат дает (при $a_1 = -1$, $A_1 = 1$, $a_2 = -1$, $A_2 = 1$)

$$x_1 = -1 + 2\xi_1,$$
$$x_2 = -1 + 2\xi_2,$$

где $0 \leqslant \xi_1 \leqslant 1$; $0 \leqslant \xi_2 \leqslant 1$.

Тогда

$$\iint\limits_{(s)} (x_1^2 + \sin^2 x_2)\, dx_1\, dx_2 = \iint\limits_{(\sigma)} [4\xi_1^2 - 4\xi_1 + 1 + \sin^2 (2\xi_2 - 1)]\, d\xi_1\, d\xi_2,$$

а область σ определяется неравенством:

$$(2\xi_1 - 1)^2 + (2\xi_2 - 1)^2 \leqslant 1.$$

Алгоритм вычисления двойного интеграла методом Монте — Карло состоит из следующих этапов:

1. Получение случайных чисел ξ распределенных равномерно в интервале (0, 1) с помощью описанной выше программы.

2. Все числа $\xi_i^{(j)}$ $(j = 1,2)$ рассматривают как координаты точек M_i, расположенных внутри единичного квадрата.

3. Определяют, принадлежит ли очередная i-я точка M_i $(\xi_i^{(1)}, \xi_i^{(2)})$ области σ, т. е. для пары последовательно выработанных случайных чисел $\xi_i^{(1)}$ и $\xi_i^{(2)}$ проверяется выполнение следующего неравенства:

$$(2\xi_i^{(1)} - 1)^2 + (2\xi_i^{(2)} - 1)^2 \leqslant 1.$$

Если это условие выполнено, то точка с координатами $(\xi_i^{(1)}, \xi_i^{(2)})$ принадлежит области σ, а значит надо вычислить $F(\xi_i^{(1)}, \xi_i^{(2)})$ и добавить это значение к предыдущему значению $\sum_{j=1}^{i-1} F_j$.

Если точка с координатами $\xi_i^{(1)}$ и $\xi_i^{(2)}$ не принадлежит области σ, то необходимо повторить выбор следующей пары случайных чисел $\xi_{i+1}^{(1)}$ и $\xi_{i+1}^{(2)}$.

4. Повторить все вычисления N раз, после чего вычислить значение интеграла по формуле 10.9.

Блок-схема алгоритма представлена на рис. 10.32, где $D = 2$, s, y и n — промежуточные переменные.

В АЛГОЛ-программе вычисления двойного интеграла процедуру выработки случайных чисел *KSI* (*KSI* — указатель функции) считают стандартной процедурой без параметров. Полученную пару ($D = 2$) случайных чисел $\xi^{(1)}$ и $\xi^{(2)}$ размещаем в массиве *KS* [1 : 2] на время работы с одной i-й точкой $(\xi_i^{(1)}, \xi_i^{(2)})$.

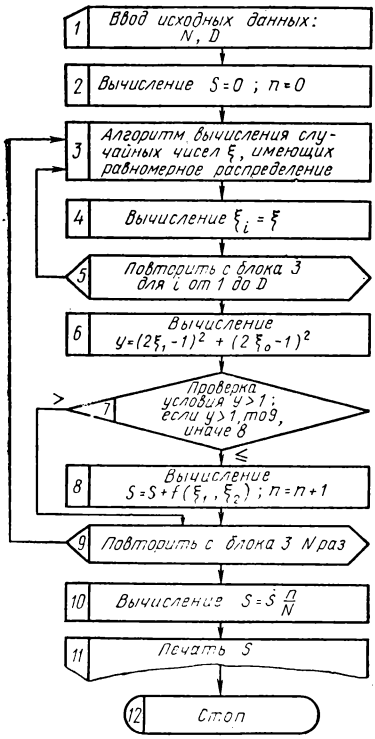


Рис. 10.32. Блок-схема алгоритма вычисления двойного интеграла методом Монте — Карло

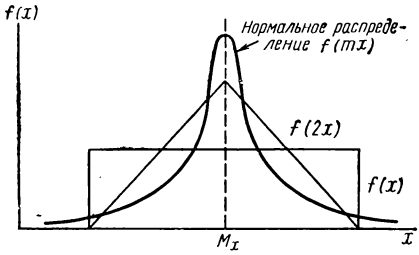


Рис. 10.33. Получение нормально-распределенных случайных чисел с помощью равномерно распределенных чисел

```
begin real s, v; array KS [1 : 2];
integer i, j, n, N; ввод (N); n := 0;
for j := 1 step 1 until N do begin
for i := step 1 until 2 do KS[i] := KSI;
y := (2 × KS [1] − 1) ↑ 2 + (2 × KS [2] − 1) ↑ 2;
if y > 1 then go to l else begin
s := s + 1 + 4 × KS [1] ↑ 2 − 4 × KS [1] + sin (2 × KS [2] − 1) ↑ 2;
n := n + 1 end; l: end; s := s × n/N; печать (s) end
```

Случайные числа с другими распределениями, отличающимися от равномерного распределения, можно получить, используя различные алгоритмы. Так, сумма случайных величин с равномерным распределением стремится к нормальному распределению (рис. 10.33). Для получения последовательности нормально распределенных чисел x_i с пара-

метрами распределения M_x и σ_x, располагая последовательностью равномерно распределенных в интервале (0,1) чисел, пользуются соотношением

$$x_i = M_x + \sigma_x \sqrt{\frac{12}{m}} \left(\sum_{j=1}^{m} \xi_j - \frac{m}{2} \right),$$

где m — число равномерно распределенных слагаемых; оно может быть, в зависимости от требуемой точности, выбрано в пределах от 5 до 15. Выбирая $m = 12$, получаем

$$x_i = M_x + \sigma_x \left(\sum_{j=1}^{12} \xi_j - 6 \right).$$

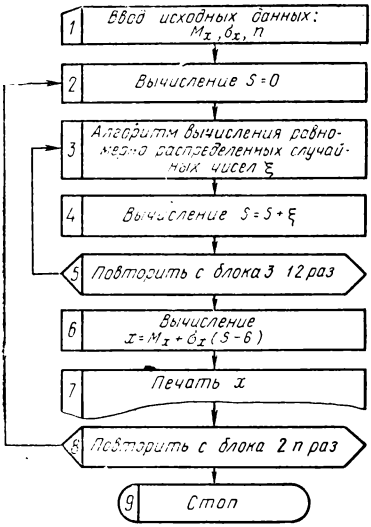


Рис. 10.34. Блок-схема алгоритма вычисления n нормально распределенных случайных чисел

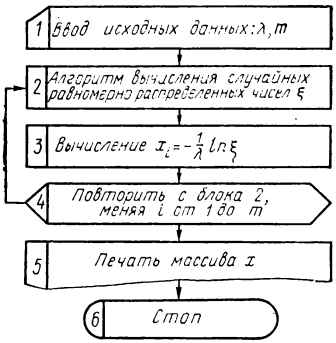


Рис. 10.35. Блок-схема алгоритма выработки m случайных чисел x, имеющих экспоненциальное распределение

Блок-схема алгоритма получения n нормально распределенных чисел представлена на рис. 10.34.

В АЛГОЛ-программе вычисления N случайных чисел, подчиняющихся нормальному распределению, процедура вычисления случайных чисел, равномерно распределенных в интервале (0,1), будет стандартной без параметров и обозначается идентификатором *KSI* (*KSI* — указатель функции):

```
begin real x, Mx, sigmax, S; integer N, i, j;
ввод (Mx, sigmax, N);
for i:=1 step 1 until N do begin s:=0;
for j:=1 step 1 until 12 do s:=s+KSI;
x:=Mx+sigmax×(s−6); печать (x) end end
```

Для получения экспоненциально-распределенных чисел воспользуемся формулой

$$x_i = -\frac{1}{\lambda} \ln \xi_i,$$

где ξ_i — случайные числа, равномерно распределенные в интервале (0,1); λ — параметр экспоненциального распределения; x_i — случайные числа, подчиненные экспоненциальному распределению.

Блок-схема алгоритма получения m случайных чисел, имеющих экспоненциальное распределение, приведена на рис. 10.35.

В программе получения m экспоненциально-распределенных случайных чисел стандартная процедура ξ_i получения равномерно распределенных чисел используется с помощью указателя функции *KSI*

```
begin integer m; ввод (m); begin real lambda,
integer i; array x [1 : m]; ввод (lambda); for i := 1 step 1;
until m do x [i] := — ln (KSI)/lambda;
печать (x) end end
```

ГЛАВА 11

МЕТОДЫ РЕШЕНИЯ ЭКОНОМИЧЕСКИХ И ПРОИЗВОДСТВЕННО-ТЕХНИЧЕСКИХ ЗАДАЧ

Ускорение темпов технического прогресса в существенной мере определяется эффективностью использования ЦВМ при научных исследованиях, технических расчетах, а также решении задач учета, планирования народного хозяйства, управления производством и экономикой. Математической базой решения этих задач являются такие разделы современной математики, как исследование операций, теория оптимального управления, математическое программирование (не путать с программированием для ЦВМ), включающее в себя линейное и нелинейное (динамическое, выпуклое, геометрическое и др.) программирование, а также методы математического моделирования.

§ 11.1. МЕТОДЫ ИССЛЕДОВАНИЯ ОПЕРАЦИЙ

Под термином «*исследование операций*» понимают применение определенных количественных методов для обоснования решений во всех областях целенаправленной человеческой деятельности. Зародившись в области военных задач, исследование операций в настоящее время все шире используется в промышленности, сельском хозяйстве, торговле, транспорте, здравоохранении и т. д. При постановке и решении задач исследование операций предлагает общие методологические приемы.

Операция определяется как система действий, объединенных единым замыслом и направленных к достижению определенной *цели*. Степень достижения цели должна описываться некоторой скалярной функцией W (целевой функцией, критерием), принимающей действительные числовые значения. При наличии такой функции цель операции заключается в максимизации (минимизации) целевой функции. Для этого имеются определенные *активные средства* (*управляемые параметры*) $x_1, x_2, \ldots, x_n$, посредством которых можно влиять на *выходные параметры* $y_1, y_2, \ldots, y_m$, являющиеся аргументами целевой функции W. На результат операции оказывают влияние также неуправляемые параметры $(\alpha_1, \ldots, \alpha_k)$, определяющие условия, в которых проводится операция.

При исследовании операции (нахождении значений управляемых параметров, максимизирующих целевую функцию) следует учитывать ограничения, наложенные на управляемые параметры (ограниченность имеющихся в распоряжении активных средств), выходные параметры (удовлетворение заданных технических условий — иногда эти ограничения устраняют соответствующим изменением целевой функции), условия проведения операции (при этом в зависимости от того какими являются неуправляемые параметры, определяющие условия проведения операции будут — постоянными, случайными неопределен-

ными величинами — различают разные постановки задачи исследования операций).

Считается, что математическая модель построена, если имеются следующие зависимости:

$$y_i = y_i(x_1, x_2, \ldots, x_n, \alpha_1, \ldots, \alpha_k), \quad (11.1)$$

$$i = 1,2, \ldots, m,$$

$$W = W(y_1, y_2, \ldots, y_m), \quad (11.2)$$

и зависимости, выражающие ограничения,

$$\varphi_j(x_1, x_2, \ldots, x_n, y_1, \ldots, y_m, \alpha_1, \ldots, \alpha_k) \left\{ \begin{matrix} > \\ < \\ = \end{matrix} \right\} 0; \quad (11.3)$$

$$j = 1, 2, \ldots, l.$$

Значения управляемых параметров, при которых выполняются ограничения (11.3), называют *допустимыми решениями*. Задача исследования операции состоит в том, чтобы из множества допустимых решений выбрать оптимальные, т. е. определить значения управляемых параметров $x_1, \ldots, x_n$, удовлетворяющих заданным ограничениям (11.3) и обращающих в максимум (минимум) целевую функцию W.

В случае, когда неуправляемые параметры являются постоянными и известными величинами, представленная задача будет *обычной оптимизационной задачей* (нахождение экстремума функции при наличии ограничений). Трудности в ее решении могут обусловливаться: сложной структурой зависимостей (11.1) ÷ (11.3) — нелинейность, отсутствие непрерывности (некоторые параметры могут быть существенно дискретными) и т. д.; тем, что управляемые или выходные параметры представляют собой не числа, а функции (скалярные, векторные или матричные), при этом критерий W называют не функцией, а *функционалом*; большой размерностью задачи (большое число управляемых и выходных параметров).

Общие методы нахождения экстремума функции при наличии ограничений разрабатываются областью математики, называемой *теорией математического программирования*. Для разных классов задач, связанных с определенными видами целевых функций и структурой ограничений, разработаны разные методы, называемые методами линейного, динамического, выпуклого, геометрического, целевого программирования, различные комбинаторные и эвристические методы (например, метод случайного поиска, ветвей и границ и др).

Рассмотренный случай, характеризующийся тем, что условия операции однозначно определены, и выбор управляемых параметров определяет значение целевой функции W, в практике встречается не часто и носит название *детерминированной задачи исследования операций*.

Вероятностная или стохастическая постановка задачи исследования операций отличается от детерминированной тем, что целевая функция W (или выходные параметры $y_1, \ldots, y_m$) зависит от некоторых случайных параметров $z_1, \ldots, z_r$, которыми могут быть все или не-

которые из параметров, определяющих условия проведения операции. В этом случае для каждого набора значений $x_1, \ldots, x_n$ целевая функция W является случайной величиной, конкретное значение которой определяется совокупностью случайных значений $z_1, \ldots, z_r$. Если известны функции $F_1(z), \ldots, F_r(z)$ распределения случайных параметров или плотности распределения $f_1(z), \ldots, f_r(z)$, то обычно переходят к оптимизации математического ожидания целевой функции (оптимизации в среднем):

$$MW = \iint \ldots \int W(\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}, z_1, z_2, \ldots, z_r)\, dF_1(z_1)\, dF_2(z_2) \ldots dF_r(z_r) =$$
$$= \iint \ldots \int W(\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}, z_1, \ldots, z_r) f_1(z_1) f_2(z_2) \ldots f_r(z_r)\, dz_1\, dz_2 \ldots dz_r.$$

При конкретной реализации операции с оптимальными с точки зрения критерия MW управляемыми параметрами результат операции W может сильно отклоняться от вычисленного наилучшего среднего результата. Поэтому, используя метод оптимизации в среднем, при исследовании операции желательно определять дисперсию величины W для оптимального набора управляемых параметров, с тем чтобы оценить возможные пределы отклонения случайной величины W. Оптимизация математического ожидания целевой функции сводит вероятностную задачу к детерминированной.

Возможен также случай, когда распределения параметров $z_1, \ldots, z_r$ неизвестны или вообще таких распределений не существует (эти параметры не обладают статистической устойчивостью). При этом параметры $z_1, \ldots, z_r$ называют не случайными, а *неопределенными*. О таких параметрах может быть известна какая-то информация, например пределы, в которых может изменяться значение параметра, его среднее значение и дисперсия, класс законов распределения, которому должна принадлежать неизвестная функция распределения параметра, и т. п. В общем случае можно сказать, что известно некоторое множество Z, которому принадлежит вектор $\mathbf{z} = \{z_1, \ldots, z_r\}$ (Z описывается совокупностью ограничений, наложенных на неопределенные параметры z). За характеристику эффективности операции при этом берут величину $\widetilde{W} = \min\limits_{\mathbf{z} \in Z} W(\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}, \mathbf{z})$, которую стремятся обратить в максимум (принцип гарантированного результата, или расчет на наиболее неблагоприятный случай).

Если целевая функция подлежит минимизации, а не максимизации, то в качестве характеристики эффективности операции берется величина $\widetilde{W} = \max\limits_{\mathbf{z} \in Z} W(\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}, \mathbf{z})$ и в результате определяются управляемые параметры, обеспечивающие $\min \widetilde{W}$ при условии выполнения ограничений (11.3).

Если предполагается, что в условиях фактического проведения операции (после исследования) имеется информация о текущих значениях неопределенных параметров z, то в качестве оптимального решения желательно иметь зависимости $x_i = x_i(\mathbf{z})$, т. е. значения x_i, дающие максимальное значение целевой функции W для фиксированного набора значений $\mathbf{z} = \{z_1, \ldots, z_r\}$. Получение таких зависимостей является

гораздо более трудоемкой задачей, чем нахождение фиксированного набора $\mathbf{x} = \{x_1, \ldots, x_n\}$, максимизирующего W.

Большое значение в исследовании операций имеет характер зависимости оптимального решения от неопределенных или случайных параметров или условий.

Критерий, определяющий эффективность операции W, зависит от набора выходных параметров $y_1, \ldots, y_m$, с каждым из которых можно связать некоторый частный критерий эффективности (частный показатель качества W_i). В качестве W_i можно брать сам параметр y_i, если необходимо его увеличение, или $-y_i$, если необходимо его уменьшение. Если имеется заданное техническими условиями ограничение $y_i \geqslant y_i^0$, где y_i^0 — предельно допустимое значение y_i, то в качестве W_i можно брать $W_i = y_i/y_i^0 - 1$ — «запас» в выполнении ограничения $y_i \geqslant y_i^0$.

Выдвинутые требования одновременной максимизации всех частных показателей обычно несовместимы, так как увеличение одного показателя уменьшает другой, и наоборот (в сущности, надо оптимально «распределить ресурсы» — активные средства — между конкурирующими показателями). Так, увеличение быстродействия ЦВМ обычно достигается за счет увеличения аппаратурных затрат, габаритов и веса устройств, а снижение веса и габаритов без потери быстродействия — за счет применения более дорогой системы элементов и т. д. Уменьшение простоев в системе обслуживания, связанное со снижением стоимости системы (например, при организации работы поликлиники), увеличивает время обслуживания и длины очередей. Следовательно, оценка качества (эффективности) операции просто по набору частных критериев $W_1, \ldots, W_m$ невозможна, если нет некоторой скалярной функции W, аргументами которой являются частные критерии эффективности W_i или выходные параметры y_i.

Определение вида целевой функции W, значения которой описывают меру достижения поставленной цели операции, представляет собой очень важную задачу, правильное решение которой в значительной мере определяет успех исследования операции. Для решения этой задачи нельзя использовать чисто формальные методы, поэтому очень большое значение имеет здесь четкое неформальное представление о цели исследуемой операции: критерий есть лишь математическое выражение осознанной, ясно представляемой цели. Вместе с тем теория исследования операций предлагает ряд способов формирования единого критерия W из набора частных критериев W_i с оценкой свойств этих способов. Приведем два характерных примера таких способов.

Способ 1. Критерий W является взвешенной суммой частных критериев W_i:

$$W = \sum_{i=1}^{m} \lambda_i W_i. \tag{11.4}$$

Неравнозначность частных критериев W_i можно отразить выбором весовых коэффициентов λ_i, что приводит к разным целям в операции. В исследовании операций разработана методика так называемых экс-

пертных оценок этих коэффициентов. Общим свойством критерия (11.4) является то, что в оптимальных решениях возможно достижение высокого значения W в ущерб какому-то частному показателю W_i. Поэтому критерий типа взвешенной суммы часто используется в том случае, когда в число ограничений включены ограничения на каждый из выходных параметров y_i.

Способ 2. Этот способ заключается в том, что в качестве W берут минимальный из частных показателей W_i:

$$W = \min_{1 \leqslant i \leqslant m} W_i.$$

Так поступают, в частности, когда W_i представляет собой «запас» в выполнении некоторых ограничений на параметр y_i:

$$W_i = y_i/y_i^0 - 1.$$

При этом W является минимальным из запасов, т. е. запасом в выполнении всей совокупности ограничений на выходные параметры. Естественной целью операции будет максимизация минимального запаса. Использование этого критерия особенно оправдано в условиях неопределенности некоторых параметров, поскольку оптимальное в смысле данного критерия решение лучше всего гарантирует выполнение заданных ограничений на выходные параметры при возможных колебаниях значений неопределенных параметров. Свойством этого критерия будет тенденция к равномерной степени достижения целей по каждому частному показателю. Тут невозможно улучшение результата операции в целом в ущерб какому-то одному показателю или, наоборот, за счет какого-то рекордного показателя. Для того чтобы придать этому способу необходимую гибкость, используют его модификацию в виде

$$W = \min_{1 \leqslant i \leqslant m} \lambda_i W_i. \qquad (11.5)$$

Изменяя набор λ_i, можно получить математическую формулировку самых разнообразных целей.

При решении задач исследования операций с несколькими показателями W_i часто выделяют главный показатель W_{i^*}, отвечающий главной цели операции, и производят его максимизацию при условиях, что остальные показатели не превосходят допустимых значений (например, проектирование ЦВМ по критерию максимальной надежности, когда параметры быстродействия, стоимости, веса и габаритов удовлетворяют определенным ограничениям). Этот случай можно рассматривать как частный случай критерия (11.5).

Пример. Ставится задача проектирования ЦВМ, показатель надежности которой (вероятность безотказной работы в течение определенного времени или средняя наработка на отказ) должен иметь максимальный запас по отношению к минимально допустимому уровню y_1^0. Стоимость y_2 не должна превосходить значения y_2^0, быстродействие y_3 должно быть не ниже значения y_3^0, вес и габариты y_4 и y_5 не должны превосходить соответственно значений y_4^0 и y_5^0.

В этом случае величины $W_1 = y_1/y_1^0 - 1$; $W_2 = y_2^0/y_2 - 1$; $W_3 = y_3/y_3^0 - 1$; $W_4 = y_4^0/y_4 - 1$; $W_5 = y_5^0/y_5 - 1$ определяют соответственно запасы по надежности, стоимости, быстродействию, весу, габаритам.

Отрицательное значение W_i говорит о том, что по данному показателю требуемые ограничения не выполнены. В качестве критерия, описывающего поставленную цель проектирования, можно взять функцию $W = \min\limits_{1 \leqslant i \leqslant 5} \lambda_i W_i$, при $\lambda_1 = 1$, $\lambda_2 = \lambda_3 = \lambda_4 = \lambda_5 = M$, где M — большое число, например 10^6.

Действительно, если ограничения на y_i для $i = 2, \ldots, 5$ выполнены, то $W_i > 0$ для $i = 2, \ldots, 5$ и $\lambda_1 W_1 < \lambda_i W_i$, т. е. $\min\limits_{1 \leqslant i \leqslant 5} \lambda_i W_i$ реализуется при значении $i = 1$.

Если же по какому-то показателю $y_{i'}$ ограничение не выполнено, то соответствующее значение $W_{i'} < 0$, $\lambda_{i'} W_{i'}$ имеет очень большое отрицательное значение, и таким образом именно это i' реализует $\min\limits_{1 \leqslant i \leqslant 5} \lambda_i W_i$. Поэтому любой алгоритм оптимизации, использующий этот критерий, автоматически учитывает сначала все частные критерии W_i, $i = 2, 3, 4, 5$, изменяя значения управляемых параметров так, чтобы удовлетворить всем ограничениям на y_i, а затем все оставшиеся ресурсы вкладывает в увеличение частного критерия W_1.

Отрицательное значение критерия W, полученное в результате оптимизации, будет указывать, что одновременное выполнение заданных ограничений на выходные параметры при заданных активных средствах недостижимо, и необходимо либо отказаться от выполнения части ограничений (например, увеличить предельно допустимое значение стоимости), либо расширить диапазон имеющихся активных средств.

Измененная задача проектирования, например, задача одновременного стремления к максимизации запаса по надежности и стоимости при выполнении ограничений на остальные параметры описывается тем же критерием $W = \min\limits_{1 \leqslant i \leqslant 5} \lambda_i W_i$ при следующих значениях λ_i : $\lambda_1 = \lambda_2 = 1$; $\lambda_3 = \lambda_4 = \lambda_5 = 10^6$.

Метод введения ограничений на выходные параметры в целевую функцию (в критерий) позволяет решать задачу оптимизации, в которой ограничения на выходные параметры можно не учитывать (они учитываются автоматически), что часто облегчает построение алгоритма оптимизации.

Идея преобразования задачи максимизации с ограничениями в задачу максимизации без ограничений путем изменения целевой функции является основой целой группы методов, называемых *методами штрафных функций.*

Рассмотрим один из возможных методов решения задачи оптимизации по критерию max W, где $W = \min\limits_{1 \leqslant i \leqslant m} \lambda_i W_i$ в условиях $A_j \leqslant x_j \leqslant B_j$, $j = 1, 2, \ldots, n$ (задача будет детерминированной, с заданными функциями $y_i = y_i(\mathbf{x})$ и $W_i = W_i(y_i)$). Рассматриваемый метод является одним из градиентных методов поиска экстремума (*алгоритм оптимального градиента*).

Градиентом функции $W(x_1, \ldots, x_n)$ будет вектор

$$\mathbf{G} = \left\{ \frac{\partial W}{\partial x_1}, \frac{\partial W}{\partial x_2}, \ldots, \frac{\partial W}{\partial x_n} \right\}.$$

Если j-я компонента вектора градиента положительна, то движение соответствующей переменной x_j вдоль оси в положительном направлении увеличивает значение функции W. Если же j-я компонента вектора градиента отрицательна, то увеличение значения функции W достигается при движении соответствующей переменной x_j в отрицательном направлении. Таким образом, вектор градиента указывает направле-

ние движения от минимума к максимуму функции W, причем самое быстрое изменение функции в окрестности рассматриваемой точки $\mathbf{x} = (x_1, \ldots, x_n)$ происходит именно вдоль направления вектора градиента.

Величина модуля M градиента определяется по формуле

$$M = \sqrt{\sum_{j=1}^{n} \left(\frac{\partial W}{\partial x_j}\right)^2}.$$

Вектор направления градиента, имеющий единичную длину,

$$\mathbf{D} = \mathbf{G}/M.$$

Идея метода заключается в последовательном изменении вектора управляемых параметров $\mathbf{x}$ в соответствии со значением вектора градиента функции W и с учетом ограничений на управляемые параметры x_j. При этом особенностью метода оптимального градиента является то, что один и тот же градиентный вектор направления используется до тех пор, пока значение целевой функции в некоторой точке $\mathbf{x}$ не начнет уменьшаться. При первом уменьшении значения целевой функции производится возврат к предыдущей точке и вычисление нового значения градиента.

Преимущество этого метода состоит в его относительной простоте, так как на каждом шаге не производится вычисление компонент вектора градиента, а только оценивается значение целевой функции. Величина приращения каждой компоненты вектора управляемых параметров может изменяться с целью ускорения сходимости метода и уменьшения колебаний вокруг экстремальной точки.

Для простоты изложения рассмотрим метод, использующий постоянное значение приращения a каждой компоненты. В этом случае

$$\Delta\mathbf{x} = a\mathbf{D},$$

где $\Delta\mathbf{x}$ — вектор поправок; $\mathbf{D}$ — вектор направления градиента.

Вычисление компонент вектора градиента производят, используя линейную аппроксимацию частных производных функций W в окрестности данной точки $\mathbf{x}$:

$$\partial W(\mathbf{x})/\partial x_j \approx \left[W(x_1, x_2, \ldots, x_j + h, \ldots, x_n) - W(x_1, \ldots, x_j, \ldots, x_n)\right]/h.$$

Исходными данными являются значения m, n, a, ε и векторы $\mathbf{A} = (A_1, \ldots, A_n)$, $\mathbf{B} = (B_1, \ldots, B_n)$, $\boldsymbol{\Lambda} = (\lambda_1, \ldots, \lambda_m)$.

Координаты начальной точки $\mathbf{x}$, которую можно взять, например, в центре допустимой области, вычисляют так:

$$x_j = (A_j + B_j)/2, \quad j = 1, 2, \ldots, n.$$

В этой точке последовательно производится расчет значений функций $y_i = y_i(\mathbf{x})$, $W_i = W_i(y_i)$ и $W = \min\limits_{1 \leqslant i \leqslant m} \lambda_i W_i$. Затем вычисляют компоненты вектора градиента и его модуль M.

Выполнение условия $M < \varepsilon$ говорит о том, что данная точка является точкой экстремума. Если это условие не выполнено, то компоненты вектора поправок вычисляют путем деления компонент вектора

градиента на его модуль и умножения на приращение a. При этом необходима проверка условия $A_j \leqslant x_j + \Delta x_j \leqslant B_j$. Если это условие не выполнено, то $\Delta x_j = 0$. После расчета всех компонент вектора поправок необходима следующая проверка: если каждая из этих компонент меньше ε, то дальнейшее увеличение значения функции W при заданных ограничениях на $\mathbf{x}$ невозможно. Если условие выполнено, то производится добавление вектора поправок к вектору управляемых параметров и вычисление нового значения критерия W. При новом значении W, меньшем предыдущего значения, производится возврат к предыдущей точке и новый расчет вектора градиента, при большем значении W происходит вычисление компонент вектора поправок в соответствии со старым значением вектора градиента (вектор поправок остается прежним, за исключением тех его компонент, прибавление которых к компонентам вектора $\mathbf{x}$ выводит последний за пределы допустимой области — эти поправки принимают равными нулю).

§ 11.2. ЛИНЕЙНОЕ ПРОГРАММИРОВАНИЕ

Важный класс оптимизационных задач составляют линейные задачи оптимизации, в которых целевая функция W линейно зависит от управляемых параметров $W = \sum_{i=1}^{n} \lambda_i x_i$ и функции ограничений также линейны. Методами решения таких задач занимается раздел теории математического программирования — линейное программирование, развитие которого вызвано к жизни потребностью решения задач, связанных с экономикой, планированием и организацией производства. Из практических задач, решаемых методами линейного программирования, наибольшее распространение получили задачи производственного планирования, на составление смесей, транспортные, составления графиков, проблемы запасов, размещения оборудования и т. д.

Простые задачи линейного программирования допускают графическую интерпретацию, позволяющую непосредственно из графика получить решение и представить идею решения более сложных задач линейного программирования.

Из численных методов решения наиболее универсальным является симплексный метод. Для частных задач линейного программирования более эффективными будут другие методы (например, транспортную задачу решают распределительным методом или методом потенциалов).

Общая формулировка задачи линейного программирования следующая.

Выбрать среди n-мерных векторов $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$, удовлетворяющих соотношениям:

$$\left.\begin{array}{c} a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n \leqslant b_1; \\ a_{21} x_1 + a_{22} x_2 + \ldots + a_{2n} x_n \leqslant b_2; \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{mn} x_n \leqslant b_m; \\ x_i \geqslant 0 \text{ для } i = 1, 2, \ldots, n, \end{array}\right\} \qquad (11.6)$$

такой, для которого достигается минимум линейной функции

$$W = \sum_{i=1}^{n} \lambda_i x_i. \tag{11.7}$$

Задачи, в которых в некоторых соотношениях (11.6) стоят знаки равенства или знак $\geqslant$, или требуется отыскание не минимума, а максимума линейной функции, легко допускают запись в приведенном виде. Действительно, соотношение $a_{j1}x_1 + a_{j2}x_2 + \ldots + a_{jn}x_n = b_j$ эквивалентно совокупности соотношений:

$$a_{j1}x_1 + a_{j2}x_2 + \ldots + a_{jn}x_n \leqslant b_j;$$
$$a_{j1}x_1 + a_{j2}x_2 + \ldots + a_{jn}x_n \geqslant b_j,$$

соотношение $a_{j1}x_1 + a_{j2}x_2 + \ldots + a_{jn}x_n \geqslant b_j$ эквивалентно соотношению $-a_{j1}x_1 - a_{j2}x_2 - \ldots - a_{jn}x_n \leqslant -b_j$, а $\max\limits_x W(x) = -\min\limits_x \{-W(x)\}$. Поэтому приведенная форма записи является общей, хотя не всегда каждую задачу линейного программирования сводят к этой форме.

Обычные методы отыскания экстремума функции с помощью приравнивания нулю производных целевой функции и решения системы получающихся уравнений для задачи линейного программирования не годятся из-за линейности целевой функции (ее производные нигде не равны нулю), а минимум (максимум) функции достигается в точках границы области, описываемой системой соотношений (11.6).

Основную идею методов линейного программирования составляет направленный перебор некоторых вариантов решения с его последовательным улучшением.

Пример. (Пример задачи на составление смеси.) Для подкормки растений необходимо внесение в почву минеральных веществ B_1 и B_2, при этом известно, что для состава почвы, которая имеется в колхозе, на единицу площади требуется внесение не менее b_1 весовых единиц B_1 и не менее b_2 весовых единиц B_2. Минеральные вещества B_1 и B_2 содержатся в имеющихся в продаже продуктах (простых удобрениях) A_1 и A_2, характеристика которых задается табл. 11.1.

Таблица 11.1

Продукт	Цена за единицу продукта	Количество B_i в единице продукта	Количество B_i в единице продукта
A_1	λ_1	a_{11}	a_{21}
A_2	λ_2	a_{12}	a_{22}

Колхозу надо решить вопрос об оптимальном составе комбинированного удобрения, т. е. определить количество продукта A_1 и количество продукта A_2, позволяющих получить необходимое количество веществ B_1 и B_2 по минимальной стоимости.

При приобретении x_1 весовых единиц A_1 и x_2 единиц A_2 затраты W составляют

$$W = \lambda_1 x_1 + \lambda_2 x_2.$$

Задача состоит в том, чтобы определить x_1 и x_2, для которых W имеет минимальное значение при выполнении ограничений:

$$\left.\begin{aligned} &a_{11}x_1 + a_{12}x_2 \geqslant b_1;\\ &a_{21}x_1 + a_{22}x_2 \geqslant b_2;\\ &x_1 \geqslant 0;\\ &x_2 \geqslant 0. \end{aligned}\right\} \qquad (11.8)$$

Пусть в конкретном случае $\lambda_1 = 1$, $\lambda_2 = 3$, $a_{11} = a_{22} = 1$, $a_{12} = 5$, $a_{21} = 3$, $b_1 = 15$, $b_2 = 6$.

Изобразим область, описываемую совокупностью ограничений (11.8) на плоскости $x_1 0 x_2$ (рис. 11.1). Переменные x_1 и x_2 неотрицательны, поэтому множество точек (x_1, x_2), являющееся возможным решением задачи, лежит в первом квадранте. Если в неравенстве $x_1 + 5x_2 \geqslant 15$ заменить знак неравенства на знак равенства, то получим уравнение прямой линии: $x_1 + 5x_2 = 15$. Эта прямая делит плоскость $x_1 0 x_2$ на две полуплоскости так, что для всех точек, лежащих в одной из полуплоскостей, $x_1 + 5x_2 > 15$, а для точек другой полуплоскости $x_1 + 5x_2 < 15$. Путем непосредственной подстановки точки (0,0) в левую часть уравнения прямой можно убедиться в том, что эта точка лежит в полуплоскости, не удовлетворяющей требуемому соотношению $x_1 + 5x_2 \geqslant 15$, т. е. допустимой с точки зрения выполнения этого соотношения является полуплоскость, не содержащая точку (0, 0). Точно так же определяется область пространства, удовлетворяющая соотношению $3x_1 + x_2 \geqslant 6$. Полученная допустимая область (область допустимых решений) отмечена на рис. 11.1 штриховкой.

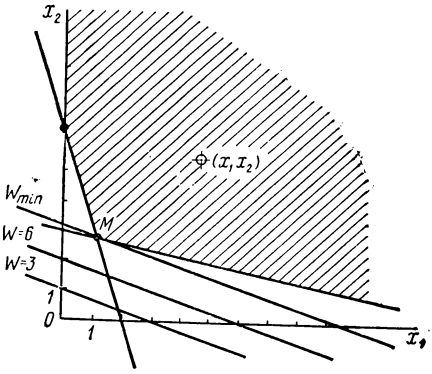


Рис. 11.1. Область допустимых решений задачи линейного программирования

Любая точка (x_1, x_2), лежащая в заштрихованной области, определяет количество компонент A_1 и A_2, обеспечивающее необходимое количество минеральных веществ B_1 и B_2. Из множества этих точек надо выбрать такую, в которой функция $W = x_1 + 3x_2$ имеет минимальное значение.

Для некоторого фиксированного значения W^* линейная функция $W^* = x_1 + 3x_2$ представляет собой прямую линию. Задаваясь различными значениями W^* (например, 0, 1, 2), получим семейство параллельных прямых. Увеличение значения линейной функции соответствует перемещению прямой параллельно самой себе вверх, поэтому, как видно из рис. 11.1, минимальное значение функции на допустимом множестве точек соответствует прямой, проходящей через точку M пересечения прямых $x_1 + 5x_2 = 15$ и $3x_1 + x_2 = 6$.

Решая систему уравнений

$$\begin{cases} x_1 + 5x_2 = 15;\\ 3x_1 + x_2 = 6 \end{cases}$$

получаем оптимальное решение $x_1 = 1{,}07$; $x_2 = 2{,}8$, соответствующее минимальному значению целевой функции, равному 9,47.

Таким образом, на единицу площади надо вносить 1,07 единиц A_1 и 2,8 единиц A_2. Это комбинированное удобрение обходится колхозу в 9,47 единиц стоимости на каждую единицу площади. Для сравнения отметим, что если бы колхоз покупал только удобрение A_1, то его пришлось бы вносить по 15 единиц на единицу площади, что обошлось бы в 15 единиц стоимости, а при внесении

одного простого удобрения A_2 на единицу площади норма внесения составляет шесть единиц, что обходится в 18 единиц стоимости.

Анализируя графическое решение задачи, можно убедиться, что при других значениях коэффициентов λ_1 и λ_2 в целевой функции оптимальное решение могло достигаться в других вершинах заштрихованного многоугольника решений или на прямых, являющихся его границами (если прямая, соответствующая целевой функции, параллельна какой-либо из границ).

Решение задачи линейного программирования симплексным методом состоит в аналитическом определении одной из вершин многогранника решений (хотя для случая $n > 3$ этот многогранник уже не имеет наглядной геометрической интерпретации) и последовательном переходе к его соседним вершинам, каждая из которых ближе к оптимальному решению, чем предыдущая (в ней имеет меньшее значение подлежащая минимизации целевая функция).

Пример. (Пример задачи производственного планирования.) Завод производит вычислительные машины типов А-50 и А-70. Процесс изготовления этих машин включает в себя три технологические операции: сборку, монтаж и наладку (остальные операции к заводу не относятся). Спрос на вычислительные машины практически неограничен. Предположим, что прибыль, получаемая от машины типа А-50, составляет 15 тыс. руб., а от А-70 — 12,5 тыс. руб.

Расчет производственных мощностей завода показал, что в течение одного квартала может быть достигнута следующая производительность (табл. 11.2).

Таблица 11.2

Операции	Производительность (шт.) машин	
	А-50	А-70
Сборка	25	30
Монтаж	50	25
Наладка	15	18

Допустим, наладка А-50 и А-70 производится разными отделами предприятия. Если объем рабочего времени за квартал составляет 100%, то доля затрат времени на проведение каждой операции может быть выражена в виде табл. 11.3.

Таблица 11.3

Операции	Доля затрат времени	
	А-50	А-70
Сборка	4	3,333
Монтаж	2	4
Наладка	6,666	5,555

Пусть x_1 и x_2 соответственно количество вычислительных машин типа А-50 и А-70, которые должны быть выпущены за квартал, тогда задачу линейного программирования можно поставить следующим образом.

Дана линейная форма

$$Q = 15x_1 + 12{,}5x_2;$$

Ограничения на переменные задаются в виде:

$$\begin{cases} 4x_1 + 3x_2 \leqslant 100; \\ 2x_1 + 4x_2 \leqslant 100; \\ 6{,}6x_1 \leqslant 100; \\ 5{,}5x_2 \leqslant 100; \\ x_1 \geqslant 0 \quad x_2 \geqslant 0. \end{cases}$$

Требуется максимизировать линейную форму.

На рис. 11.2 проведены прямые, соответствующие уравнениям

$$4x_1+3x_2=100; \quad (1-1)$$
$$2x_1+4x_2=100; \quad (2-2)$$
$$6{,}6x_1=100; \quad (3-3)$$
$$5{,}5x_2=100. \quad (4-4)$$

Прямые и оси координат выделяют на плоскости многоугольник $OBCDE$, внутри которого точка представляет собой допустимое решение задачи.

Рассмотрим семейство прямых $Q = 15\,x_1 + 12\,x_2$.

Прямая $A - A$ принадлежит данному семейству. Значение Q, как известно из аналитической геометрии, пропорционально расстоянию OR данной прямой от начала координат. Перемещая прямую $A - A$ параллельно самой себе до крайней точки D, которая является последней общей точкой прямой $A' - A'$ и многоугольника, получим требуемое оптимальное решение.

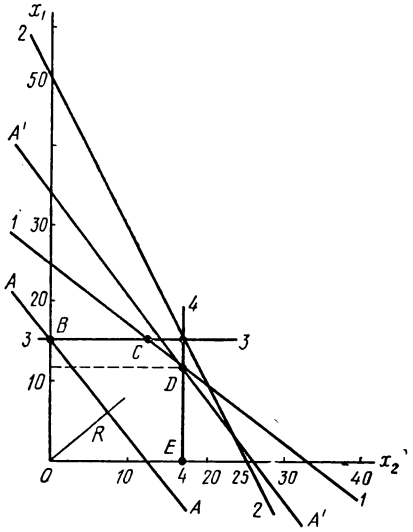


Рис. 11.2. Графическое решение задачи линейного программирования

Из графика видно, что прямая $2 - 2$, соответствующая производительности операции «монтаж», находится вне прямоугольника условий $OBCDE$ и практически не вносит ограничений в задачу. С математической точки зрения это условие является лишним, а с точки зрения планирования — отражает резерв производства по этой операции, который может быть использован для других целей.

Поскольку максимум линейной функции Q достигается в точке D и соответствует значениям переменных $x_1 = 10$ и $x_2 = 18$, то $Q = 15 \cdot 10 + 12{,}5 \times 18 = 375$ тыс. руб.

Следовательно, если завод в течение квартала выпустит 10 машин типа А-50 и А-18 машин типа А-70, то его наибольшая прибыль составит 375 тыс. руб. Из графика также видно, что можно, например, сократить производительность по операции «наладка» машины А-50, что приведет к снижению себестоимости. Вопрос о снижении или увеличении производительности по той или другой операции должен решаться в комплексе с целым рядом других производственных вопросов, которые в данной задаче не рассматривались.

Каноническая форма задачи линейного программирования. Пусть дана система m линейных неравенств с n переменными:

$$\left.\begin{aligned} a_{11}x_1+a_{12}x_2+\ldots+a_{1n}x_n &\leqslant b_1;\\ a_{21}x_1+a_{22}x_2+\ldots+a_{2n}x_n &\leqslant b_2;\\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\ a_{m1}x_1+a_{m2}x_2+\ldots+a_{mn}x_n &\leqslant b_m. \end{aligned}\right\} \quad (11.9)$$

Более коротко эту систему можно записать так:

$$\sum_{j=1}^{n} a_{ij}x_j \leqslant b_i, \quad i=1,2,\ldots,m.$$

Таким образом может быть записана любая система неравенств, поскольку знак неравенства можно изменить на противоположный путем умножения обеих его частей на —1. Систему неравенств (11.9) можно преобразовать в систему уравнений посредством прибавления к левой части каждого неравенства неотрицательных переменных, которые называют *дополнительными переменными*. В результате получают следующую систему уравнений:

$$\left.\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n + x_{n+1} &= b_1; \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n + x_{n+2} &= b_2; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n + x_{n+m} &= b_m, \end{aligned}\right\} \qquad (11.10)$$

где $x_{n+i} \geqslant 0$, $i = 1, 2, \ldots, m$.

Можно показать, что всякому решению системы неравенств (11.9) соответствует определенное решение системы линейных уравнений (11.10), у которого те же самые значения основных переменных $x_1, x_2, \ldots, x_n$. Поскольку систему линейных неравенств путем введения дополнительных неотрицательных переменных можно свести к системе линейных уравнений, то общую задачу линейного программирования можно сформулировать следующим образом.

При условиях

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i = 1, 2, \ldots, m, \qquad (11.11)$$

$$x_j \geqslant 0, \quad j = 1, 2, \ldots, n \qquad (11.12)$$

найти минимум линейной функции

$$W = \sum_{j=1}^{n} \lambda_j x_j \qquad (11.13)$$

Форму записи общей задачи линейного программирования в виде системы (11.11)—(11.13) называют *канонической формой*. При записи задачи линейного программирования в канонической форме число линейно независимых уравнений (т. е. таких уравнений, которые не могут быть получены из остальных уравнений путем сложения с умножением на произвольные числовые множители), как правило, меньше числа переменных. Если число независимых уравнений больше числа переменных, то такая система несовместна; если число независимых уравнений равно числу переменных, то система имеет единственное решение (это решение будет либо недопустимым, если хотя бы одна из его компонент отрицательна, либо искомым оптимальным).

Поэтому практический интерес при оптимизации представляют задачи, в которых число линейно независимых уравнений меньше числа переменных. Если число независимых уравнений равно m, а число переменных равно n, то при совместности системы у нее существует бесконечное множество решений. В этих решениях $n - m$ переменных могут принимать произвольные значения (так называемые *свободные*

переменные), а остальные m переменных выражаются через них (их называют *базисными переменными*).

Задача линейного программирования, записанная в канонической форме, допускает геометрическое изображение (и решение) на плоскости, если число переменных n на два превышает число независимых уравнений m. В этом случае надо выразить все переменные через две из них, тогда задача описывается в виде:

$$\left.\begin{aligned} &x_k = a'_{1k}\, x_1 + a'_{2k}\, x_2 + b'_k, \quad k = 3, 4, \ldots, n; \\ &x_i \geqslant 0, \quad i = 1, 2, \ldots, n \\ &W = \lambda'_1\, x_1 + \lambda'_2\, x_2. \end{aligned}\right\}$$

Прямые $x_k = 0$, $k = 1, 2, \ldots, n$ (две из них будут осями координат) определяют на плоскости x_1Ox_2 границы выпуклого многоугольника, а совместный учет «допустимых полуплоскостей» $x_i \geqslant 0$ определяет допустимую область решений (при этом может встретиться случай, когда такой области не будет и условия неотрицательности переменных противоречат друг другу).

Коэффициенты линейной функции определяют семейство параллельных прямых на плоскости и направление, в котором увеличивается значение функции. При существовании допустимой области решений для задачи минимизации, прямую на плоскости надо перемещать параллельно самой себе в сторону уменьшения ее значений до тех пор, пока она еще будет содержать точки многогранника.

Если число переменных n более чем на три превышает число независимых уравнений задачи линейного программирования, записанной в канонической форме, то необходимо использование аналитических методов, поскольку геометрическая интерпретация становится невозможной.

Наиболее универсальным из вычислительных методов линейного программирования является симплексный метод.

Симплексный метод решения задачи линейного программирования.

Решением называют любой набор переменных $x_1, x_2, \ldots, x_n$, удовлетворяющий уравнениям (11.11).

Допустимым решением называют решение с неотрицательными переменными.

Базисом называют набор таких переменных, при которых матрица, составленная из коэффициентов этих переменных в уравнениях, будет невырожденной, т. е. ее определитель отличен от нуля.

Базисным решением называют такое решение, которое получится, если положить все небазисные (свободные) переменные равными нулю и решить уравнения относительно базисных переменных.

Принцип нахождения оптимального решения в симплексном методе состоит в следующем. Задачу линейного программирования записывают в канонической форме. Определяют допустимое базисное решение и проверяют его оптимальность. Если решение не оптимальное, то из базиса вычеркивают определенную переменную и вместо нее вводят другую. В результате многократного повторения описанного процесса либо будет получено оптимальное решение, либо будет выявлена про-

тиворечивость ограничений, либо станет видно, что при допустимых базисных решениях линейная функция является неограниченной.

Процедура пересчета коэффициентов в уравнениях при переходе к новому базису может быть формализована и сведена к заполнению стандартных *симплексных таблиц*. Эта формализация особенно важна при использовании для вычислений ЦВМ, что становится насущной необходимостью в задачах планирования, где число переменных составляет несколько десятков.

Пример. Минимизировать функцию

$$W = -x_1 + x_2$$

в условиях ограничений

$$\begin{cases} -2x_1 + x_2 \leqslant 2; \\ x_1 - 2x_2 \leqslant 2; \quad x_1 \geqslant 0; \quad x_2 \geqslant 0. \\ x_1 + x_2 \leqslant 5 \end{cases}$$

Вводя дополнительные переменные x_3, x_4 и x_5, приводим задачу к каноническому виду:

$$\begin{cases} -2x_1 + x_2 + x_3 = 2; \\ x_1 - 2x_2 + x_4 = 2; \quad x_i \geqslant 0, \quad i = 1, 2, \ldots, 5. \\ x_1 + x_2 + x_5 = 5 \end{cases}$$

Выбирая в качестве базиса дополнительные переменные x_3, x_4 и x_5 (коэффициенты при этих переменных образуют единичную матрицу), получаем допустимое базисное решение (0, 0, 2, 2, 5). Первая симплексная таблица составляется на основе исходных данных задачи и выбранного первого базиса. Для данного случая она имеет следующий вид (табл. 11.4).

Таблица 11.4

Базисные переменные	λ_i базисных переменных	λ_j линейной функции					Свободные члены
		−1	1	0	0	0	
		элементы таблицы S_{ij}					
3	0	−2	1	1	0	0	2
4	0	1	−2	0	1	0	2
5	0	1	1	0	0	1	5
Индексная строка z_j		1	−1	0	0	0	0

Элементы S_{ij} первой симплексной таблицы представляют собой коэффициенты левых частей уравнений канонической формы, справа записан столбец свободных членов уравнений, а слева — номера базисных переменных и соответствующие им коэффициенты линейной функции. Ниже симплексной таблицы записывается *индексная строка*, анализ элементов которой позволяет установить, достигнуто ли оптимальное решение. Ее элементы получаются по формулам

$$z_j = \sum_{i=1}^{m} \lambda_{бi} S_{ij} - \lambda_j, \quad j = 1, 2, \ldots, n.$$

Последний элемент строки $z_{n+1} = \sum_{i=1}^{m} \lambda_i b_i$ представляет собой значение линейной функции при данном базисном решении. Признаком того, что полученное решение не будет оптимальным, служит наличие в индексной строке положитель-

ных элементов в случае, когда отыскивается минимум (или отрицательных — в случае отыскания максимума).

Если решение не оптимальное, то составляется вторая симплексная таблица, соответствующая улучшенному решению. Для этого в индексной строке отыскивается наибольшее положительное число (в случае нахождения максимума — наибольшее по модулю отрицательное число). Столбец, в котором оно расположено, называют *ключевым вектором-столбцом*. Далее находится в таблице *ключевая вектор-строка*, определяемая как строка, содержащая наименьшее положительное частное от деления компонент столбца свободных членов на элементы ключевого столбца. Затем переменную, соответствующую ключевой строке, выводят из базиса, заменяя ее переменной, стоящей в ключевом столбце. Для этого r-й элемент столбца $\lambda_б$ ($\lambda_{б_r}$) должен быть заменен на λ_k, а номер базисного переменного r — на k (r — номер ключевой строки, k — номер ключевого столбца).

При этом происходит следующее изменение табл. 11.4. Преобразуется ключевая строка путем деления ее элементов на *разрешающий элемент* — элемент на пересечении ключевой строки и ключевого столбца. Элементы ключевого столбца заменяются нулями, кроме разрешающего элемента, который заменяется единицей. Остальные элементы новой симплексной таблицы вычисляются по формулам

$$S^*_{ij} = S_{ij} - S_{ik} S_{rj}/S_{rk},$$
$$i = 1, 2, \ldots, m \quad i \neq r;$$
$$j = 1, 2, \ldots, n \quad j \neq k,$$

где r — номер ключевой строки, k — номер ключевого столбца. По таким же правилам преобразуются элементы столбца свободных членов и индексной строки, поэтому их считают входящими в симплексную таблицу:

$$b_i = S_{i,\, n+1}; \quad z_j = S_{m+1,\, j},$$
$$i = 1, 2, \ldots, m;$$
$$j = 1, 2, \ldots, n+1.$$

Преобразованная табл. 11.4 имеет следующий вид (табл. 11.5)

Таблица 11.5

$б_i$	$\lambda_{б_i}$	λ_j					b_i
		—1	1	0	0	0	
		S_{ij}					
3	0	0	—3	1	2	0	6
1	—1	1	—2	0	1	0	2
5	0	0	3	0	—1	1	3
z_j		0	1	0	—1	0	—2

Анализ индексной строки этой таблицы показывает, что значение целевой функции уменьшилось до —2, но в индексной строке имеется положительный элемент, указывающий на то, что решение еще не оптимально, и определяющий собой новый ключевой столбец.

Повторение описанного процесса нахождения ключевой строки и преобразования базиса приводит к третьей симплексной таблице (табл. 11.6), имеющей следующий вид:

Таблица 11.6

$б_i$	$\lambda_{б_i}$	λ_j					b_i
		—1	1	0	0	0	
		S_{ij}					
3	0	0	0	1	1	1	9
1	—1	1	0	0	1/3	2/3	4
2	1	0	1	0	—1/3	1/3	1
z_j		0	0	0	—2/3	—1/3	—3

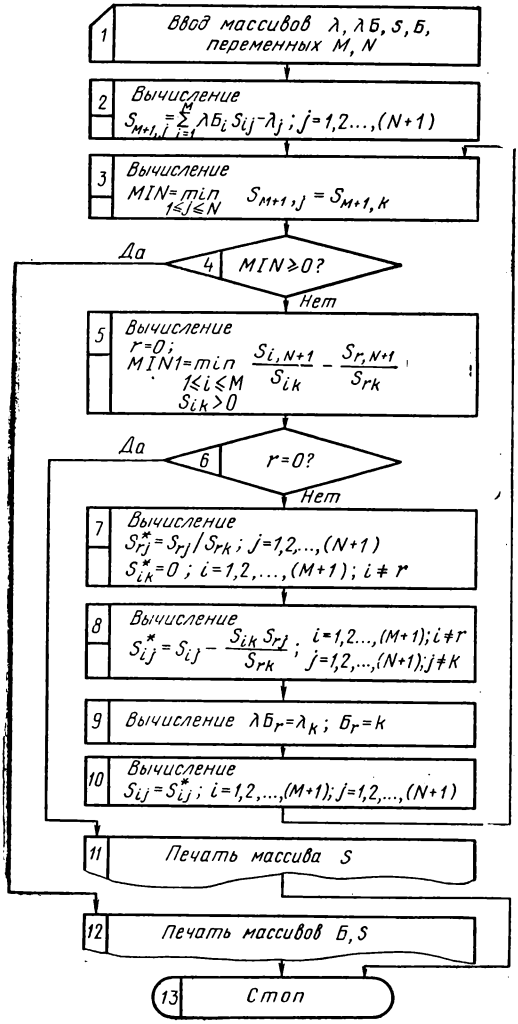


Рис. 11.3. Блок-схема алгоритма симплексного метода

Отсутствие положительных элементов в индексной строке указывает на то, что получено оптимальное решение. Минимальное значение целевой функции равно — 3.

Укрупненная блок-схема описанного алгоритма для $m \leqslant 20$ и $n \leqslant 50$ приведена на рис. 11.3. Блок анализа *6* служит для определения наличия в ключевом столбце положительных элементов. Если таких элементов в нем не содержится, то задача не имеет оптимального решения.

Таким образом, описанный алгоритм может использоваться для решения задачи линейного программирования, приведенной к канонической форме при выбранном допустимом базисном решении. Если по условию задачи непосредственно имеется система уравнений, а не неравенств (первоначальный базис при этом найти трудно, так как таблица коэффициентов не содержит единичной матрицы), то для нахождения первого базиса используют *М-метод* (метод искусственного базиса), заключающийся в следующем.

Пусть требуется найти минимум линейной функции

$$W = \sum_{j=1}^{n} \lambda_j x_j$$

при условиях

$$\sum_{j=1}^{n} a_{ij} x_j = b_i;\ x_j \geqslant 0,$$
$$i = 1, 2, \ldots, m;$$
$$j = 1, 2, \ldots, n.$$

Введя в уравнения *искусственные переменные* x_{n+i}, получим:

$$\sum_{j=1}^{n} a_{ij} x_j + x_{n+i} = b_i;$$
$$x_j \geqslant 0;$$
$$i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, (n+m).$$

Требуется найти минимум функции

$$f = \sum_{j=1}^{n} \lambda_j x_j + M \sum_{i=1}^{n} x_{n+i},$$

где M — достаточно большое положительное число. Тогда переменные $x_{n+1}, x_{n+2}, \ldots, x_{n+m}$ образуют базис (коэффициенты при этих переменных образуют единичную матрицу), называемый *искусственным*. Если исходная задача имеет хотя бы одно решение, то это решение является также решением расширенной задачи. Применение симплексного метода к расширенной задаче обеспечивает такое решение, в котором каждое из искусственных переменных x_{n+i} равно нулю. Если исходная задача не имеет решений, то решение расширенной задачи будет содержать, по крайней мере, одно $x_{n+i} > 0$.

§ 11.3. ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Динамическое программирование представляет собой особый математический метод отыскания оптимальных решений, позволяющий производить оптимизацию поэтапно. Этот метод заменяет решение исходной задачи оптимизации большой размерности решением последовательности задач оптимизации малой размерности и применим к многошаговым операциям (типичная задача — планирование производства на ряд лет).

Основную идею метода динамического программирования составляет *принцип оптимальности*, сформулированный Р. Беллманом. Принцип оптимальности заключается в том, что, каково бы ни было состояние оптимизируемой (управляемой) системы в результате какого-то числа шагов управления, управление на ближайшем шаге должно быть выбрано так, чтобы оно обеспечивало максимальный выигрыш на всех оставшихся шагах, включая данный. Применение этого принципа приводит к составлению *основного функционального уравнения* в каждой решаемой задаче.

Пример. Начальное количество средств C^* распределяется в течение m лет между двумя отраслями производства A и B. Каждая отрасль производства обеспечивает годовой доход, зависящий от объема капиталовложений в начале года. Вложение x в отрасль A дает годовой доход $f_A(x)$, вложение x в отрасль B — $f_B(x)$. При этом происходит частичное уменьшение (амортизация) вложенных

средств на величину $\varphi_A(x)$ в отрасли A и на величину $\varphi_B(x)$ в отрасли B. По истечении года оставшиеся средства снова распределяются между отраслями (в рассматриваемом случае доход в производство не вкладывается).

Требуется найти распределение ресурсов для каждого года, при котором суммарный доход за m лет будет максимальным.

Целевая функция имеет вид

$$W = \sum_{i=1}^{m} W_i, \tag{11.14}$$

где

$$W_i = f_A(x_i) + f_B(c_i - x_i) = w_i(c_i, x_i),$$

где x_i — средства, вкладываемые в отрасль A в i-м году; c_i — общие капиталовложения в начале i-го года.

При этом

$$c_{i+1} = c_i - \varphi_A(x_i) - \varphi_B(c_i - x_i) = c'(c_i, x_i), \tag{11.15}$$

$$c_1 = C^*.$$

Необходимо отыскать $x_1, x_2, \ldots, x_m$, где $x_i \leqslant c_i \quad i = 1, 2, \ldots, m$, максимизирующие целевую функцию W.

Для составления основного функционального уравнения расмотрим планирование в начале i-го года, предполагая, что имеется сумма капиталовложений c. В соответствии с принципом оптимальности эта сумма должна быть распределена так, чтобы получить максимальный доход в течение оставшихся $m - i$ лет. $\widetilde{W}_i(c)$ называют условным оптимальным выигрышем, если это максимальный выигрыш, который можно получить за оставшиеся $m - i$ лет при наличии к i-му году запаса средства c. Тогда

$$\widetilde{W}_i(c) = \max_{x_i \leqslant c} \{W_i(c, x_i) + \widetilde{W}_{i+1}(c')\}, \tag{11.16}$$

где $c' = c - \varphi_A(x_i) - \varphi_B(c - x_i)$.

Задача нахождения $\widetilde{W}_i(c)$ есть одномерная задача оптимизации (отыскание максимума функции одного переменного), но ее надо решать для разных значений параметра c. Нахождение условных оптимальных выигрышей $\widetilde{W}_i(c)$ будем производить «от конца к началу». Определяется

$$\widetilde{W}_m(c) = \max_{x_m \leqslant c} W_m(c, x_m).$$

Зная функцию $\widetilde{W}_m(c)$ и определяя для каждого значения c значение c' по формуле

$$c' = c - \varphi_A(x_{m-1}) - \varphi_B(c - x_{m-1}),$$

можем решить следующую одномерную задачу оптимизации: найти функцию $\widetilde{W}_{m-1}(c)$, используя основное функциональное уравнение

$$\widetilde{W}_{m-1}(c) = \max_{x_{m-1} \leqslant c} \{W_{m-1}(c, x_{m-1}) + \widetilde{W}_m(c')\}.$$

Функции $\widetilde{W}_{m-2}(c)$, $\widetilde{W}_{m-3}(c)$, ..., $\widetilde{W}_1(c)$ определяются аналогично.

Окончательно $\max_x W = \widetilde{W}_1(c^*)$, а оптимальные распределения средств для каждого года можно получить в такой последовательности: $x_1 = x_1(C^*)$; $c_2 = c'(C^*, x_1)$; $x_2 = x_2(c_2)$; $c_3 = c'(c_2, x_2)$; ...; $x_m = x_m(c_m)$.

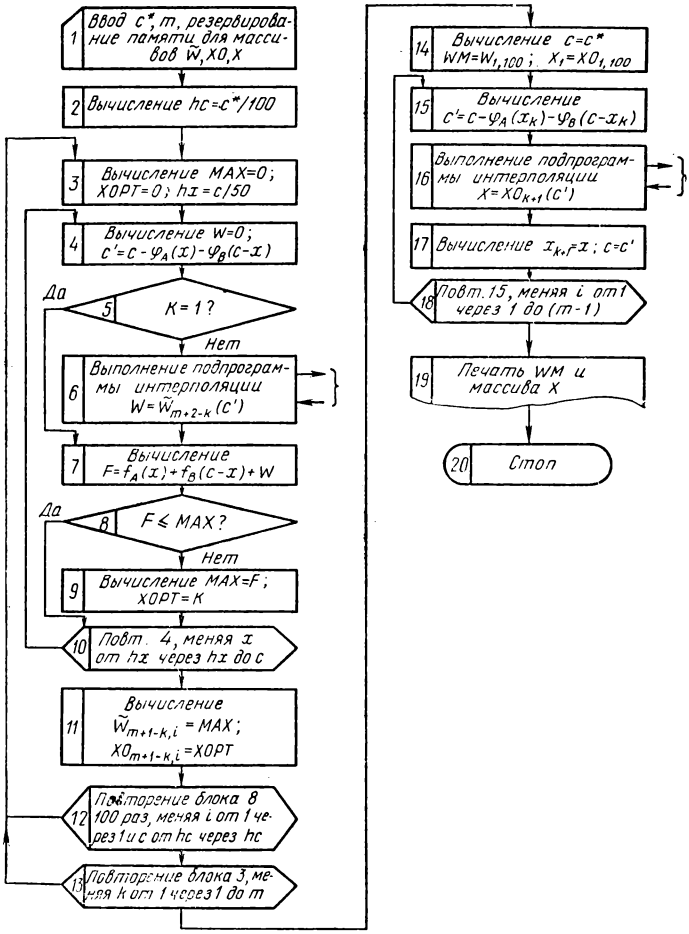


Рис. 11.4. Блок-схема алгоритма динамического программирования

Блок-схема алгоритма решения задачи, в котором каждая одномерная задача оптимизации решается методом простого перебора с количеством точек $r = 50$, приведена на рис. 11.4.

§ 11.4. ЦИФРОВОЕ МОДЕЛИРОВАНИЕ

Моделирование определяется как процесс представления динамической системы моделью для получения информации о поведении системы путем проведения экспериментов над ее моделью. Для моделирования могут использоваться как АВМ, так и ЦВМ. Применение АВМ для построения моделей динамических систем, описываемых системами дифференциальных уравнений, известно давно. В последние годы

в связи с быстрым развитием ЦВМ, значительным ростом их быстродействия и объема запоминающих устройств возрос интерес к использованию в качестве моделирующих средств универсальных ЦВМ. При этом моделируемая система представляется в виде машинной программы, в которой имитируются процессы, протекающие в реальной системе. Такая имитация процесса называется *цифровым моделированием.* В силу универсальности ЦВМ возникает возможность создавать модели самых разнообразных систем, при этом их функционирование не обязательно должно описываться дифференциальными или алгебраическими уравнениями. Моделирование можно применять для изучения производственных и экономических систем, сетей транспорта и связи, сложных технических комплексов, биологических систем и процессов.

Моделирование дает возможность исследования и имитации особенностей функционирования системы в любых условиях. При этом параметры системы и окружающей среды можно варьировать с целью определения оптимального варианта структуры и получения зависимостей выходных характеристик от изменения условий. Модель позволяет легко реализовать имитацию работы системы при наличии случайных параметров или условий.

Применение метода цифрового моделирования полезно в том случае, когда исследуемая система не поддается изучению аналитическими методами, а прямое экспериментирование с системой затруднительно или нецелесообразно.

Для составления цифровых моделей используют как универсальные языки программирования — АЛГОЛ, ФОРТРАН, АКИ, так и специализированные языки, разработанные специально для представления алгоритмов моделирования: СОЛ, СИМУЛА, СИМСКРИПТ; СЛЭНГ, СТАМ и др. В этих языках предусматриваются средства автоматического управления последовательностью изменений («событий») в модели, динамического распределения данных в памяти, необходимого для построения сложных и больших по объему моделей, стандартные программы статистической обработки результатов моделирования (накопления и вывода гистограмм, средних значений случайных величин, их дисперсий и т. п.).

Пример. Основные моменты, связанные с построением цифровой модели, проиллюстрируем на примере моделирования работы кассы для компостирования и продажи железнодорожных билетов. Целью моделирования такой системы может являться изучение таких характеристик, как длина очереди и время ожидания в очереди в зависимости от интенсивности потока пассажиров и производительности кассира с целью получения рекомендаций о количестве касс или о целесообразности автоматизации, и пр.

Поскольку моменты прихода пассажиров в кассу являются случайными величинами, то и длина очереди и время ожидания также будут случайными; поэтому в результате моделирования надо определять их статистические характеристики: среднее значение, дисперсию, гистограмму. Функционирование кассы описывается следующим образом.

В кассу приходят два типа пассажиров: одни для компостирования билета (время обслуживания таких пассажиров составляет τ_1 с); другие — для покупки билета (время обслуживания таких пассажиров равно τ_2 с). Для простоты изложения будем считать, что τ_1 и τ_2 — постоянные величины (нетрудно видоизмерить модель так, чтобы τ_1 и τ_2 были случайными величинами с заданным законом распределения).

Касса может обслуживать одновременно только одного пассажира. Если в момент прибытия нового пассажира касса занята, то он становится в одну из двух очередей — одна состоит из пассажиров, компостирующих билеты, другая — из пассажиров, покупающих билеты. Пусть обслуживание этих очередей кассой производится в таком порядке: в момент освобождения кассы на обслуживание поступает пассажир, стоящий первым в очереди на компостирование. Тольko если эта очередь пуста, обслуживается первый пассажир из очереди покупающих билеты (выражаясь языком теории очередей, можно сказать, что на вход обслуживающей системы с ожиданием поступают два независимых потока требований (заявок), обслуживаемых по правилу относительных приоритетов).

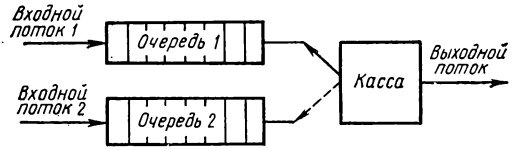


Рис. 11.5. Формализованная схема работы кассы

Формализованная схема системы показана на рис. 11.5. На рис. 11.6 изображена временная диаграмма, иллюстрирующая работу этой системы.

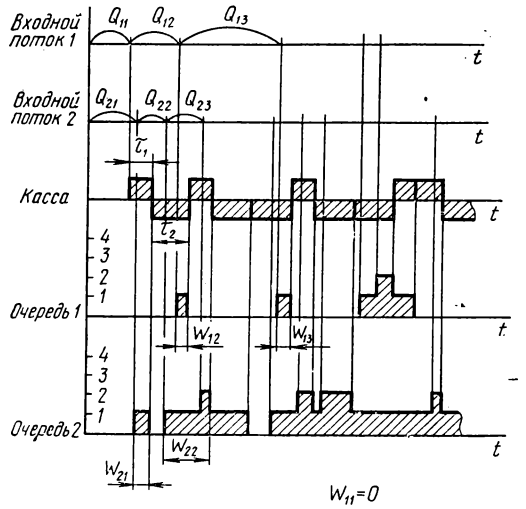


Рис. 11.6. Временная диаграмма работы кассы

Оба потока пассажиров, приходящих в кассу, можно описать функцией распределения $A(t)$ промежутков времени между моментами прибытия их в очередь (эта функция будет своя для каждого потока):

$$A(t) = \Pr\{\theta < t\}.$$

При моделировании эта функция уже должна быть известна (ее можно получить, например, путем регистрации и последующего статистического анализа моментов прибытия пассажиров в очередь в реальных условиях).

Практика исследования систем обслуживания показывает, что во многих случаях удовлетворительной оказывается аппроксимация функции распреде-

ления интервалов между моментами поступления заявок в систему обслуживания экспоненциальной функцией:

$$A(t)=\begin{cases}1-e^{-\lambda t}, & \text{для } t\geqslant 0;\\ 0 & \text{для } t<0,\end{cases}$$

где λ — величина, обратная среднему интервалу времени между заявками (интенсивность потока).

В этом случае заявки на входе образуют так называемый *пуассоновский поток*, одним из свойств которого является то, что он «наиболее неудобен» для системы обслуживания, в том смысле, что время на ожидание и длины очередей в случае пуассоновских потоков оказываются большими, чем в случае потоков с другими функциями распределения, но с той же интенсивностью. Это свойство позволяет использовать пуассоновский поток для анализа систем, в которых истинное распределение для входного потока получить затруднительно (своеобразный «расчет на худший случай»).

Если временная диаграмма, отражающая работу системы за достаточно длинный промежуток времени T, построена так, что случайные величины θ_{1i} и θ_{2i} соответствуют реальным законам распределения, то статистические характеристики работы системы можно получить анализом этой временной диаграммы. Предположим, что интерес представляет среднее время ожидания в очереди для пассажиров первого и второго потоков. Для каждого пассажира i время ожидания w_i в очереди равно разности между моментом времени, когда он начал обслуживаться в кассе, и моментом времени, когда он пришел в систему. Среднее время на ожидание составляет

$$w_{1\,\mathrm{ср}}=\frac{1}{n_1}\sum_{i=1}^{n_1}w_{1i};\quad w_{2\,\mathrm{ср}}=\frac{1}{n_2}\sum_{i=1}^{n_2}w_{2i},$$

где n_1 и n_2 — число пассажиров соответственно первого и второго потоков, обслуженных системой за время T.

Суммируя значения количества пассажиров в очереди через небольшие промежутки времени и разделив полученную сумму на число суммирований, получим среднее значение длины очереди:

$$L_{1\,\mathrm{ср}}=\frac{1}{N}\sum_{i=1}^{N}m_{1i};\quad L_{2\,\mathrm{ср}}=\frac{1}{N}\sum_{i=1}^{N}m_{2i},$$

где m_{1i} и m_{2i} — количество пассажиров в первой и второй очереди в момент наблюдения i; N — число моментов наблюдения (моментов снятия статистики) за время T.

Дисперсия величин w и L

$$Dw_1=\frac{1}{n_1-1}\sum_{i=1}^{n_1}(w_{1i}-w_{1\,\mathrm{ср}})^2;$$

$$Dw_2=\frac{1}{n_2-1}\sum_{i=1}^{n_2}(w_{2i}-w_{2\,\mathrm{ср}})^2;$$

$$DL_1=\frac{1}{N-1}\sum_{i=1}^{N}(m_{1i}-L_{1\,\mathrm{ср}})^2;$$

$$DL_2=\frac{1}{N-1}\sum_{i=1}^{N}(m_{2i}-L_{2\,\mathrm{ср}})^2.$$

Эти формулы удобнее преобразовать в такой вид:

$$Dw_1 = \frac{1}{n_1 - 1} \sum_{i=1}^{n_1} w_{1i}^2 - \frac{n_1}{n_1 - 1} w_{1\text{ср}}^2;$$

$$Dw_2 = \frac{1}{n_2 - 1} \sum_{i=1}^{n_2} w_{2i}^2 - \frac{n_2}{n_2 - 1} w_{2\text{ср}}^2;$$

$$DL_1 = \frac{1}{N - 1} \sum_{i=1}^{N} m_{1i}^2 - \frac{N}{N - 1} L_{1\text{ср}}^2;$$

$$DL_2 = \frac{1}{N - 1} \sum_{i=1}^{N} m_{2i}^2 - \frac{N}{N - 1} L_{2\text{ср}}^2.$$

Можно построить и гистограммы, характеризующие распределение величин w и L.

Таким образом, для получения статистических характеристик работы системы достаточно иметь временну́ю диаграмму, в которой все случайные величины должны подчиняться заданным законам распределения. При этом не обязательно временная диаграмма должна быть целиком (для всего временного интервала от 0 до T). Статистику можно накапливать постепенно, в процессе работы системы или модели.

Наблюдение с секундомером за работой реальной системы является длительным и трудоемким процессом, не позволяющим исследовать систему в условиях изменения ее параметров. В этом случае на помощь может придти модель — имитатор системы. Для ее построения прежде всего надо уметь имитировать моменты поступления в очередь пассажиров каждого потока.

Для первого потока

$$t_{1i+1} = t_{1i} + \theta_{1i},$$

где величины θ_{1i} распределены по закону $A_1(t)$.

Нетрудно доказать, что если имеется случайная величина R, распределенная равномерно в интервале (0, 1), то для получения величины θ, имеющей функцию распределения $A(t)$, надо решить уравнение $A(\theta) = R$ относительно θ.

В частности, для

$$A(t) = 1 - e^{-\lambda t}, \quad t \geqslant 0$$

надо решить уравнение

$$1 - e^{-\lambda\theta} = R.$$

Откуда

$$\theta = -\frac{1}{\lambda} \ln (1 - R). \tag{11.17}$$

Таким образом, получение θ сводится к нахождению R, подчиняющегося равномерному распределению, и вычислению θ по формуле (11.17).

Во всех системах программирования имеются специальные стандартные программы, получающие так называемые псевдослучайные числа, последовательность которых подчиняется равномерному распределению в интервале (0, 1). Определение одного такого числа требует одного обращения к этой стандартной программе.

Таким образом, алгоритм получения момента времени прихода следующего пассажира в очередь можно изобразить в виде блок-схемы, показанной на рис. 11.7 для первого потока, в предположении, что этот поток пуассоновский.

Далее необходимо составить алгоритм, описывающий логику работы очереди (очевидно, алгоритм будет идентичным для обеих очередей). Предположим, что очередь имеет максимальную длину LM (число мест для ожидания). Одномерный массив **P**, состоящий из элементов (ячеек) $\mathbf{P}_1$, $\mathbf{P}_2$, ..., $\mathbf{P}_{LM}$, имитирует «места» в этой очереди. Каждая из этих ячеек может быть либо свободна, либо «занята пассажиром». В качестве эквивалента пассажира удобно брать момент его прихода в очередь — эта информация понадобится для определения времени ожидания им в очереди.

Ячейка X используется в качестве рабочей ячейки при записи очередного пассажира в очередь или при выборе из очереди.

Ячейки PER и POS содержат информацию, позволяющую определить соответственно первого и последнего пассажира в очереди. Этой информацией является номер (индекс) соответствующей ячейки массива P.

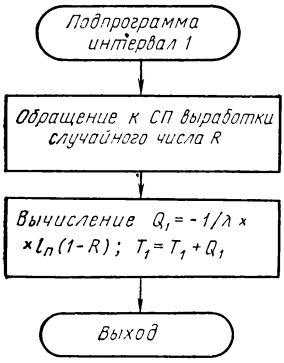


Рис. 11.7. Блок-схема алгоритма имитации входного потока 1

В момент выбора пассажира из очереди содержимое ячейки PER увеличивается на единицу по правилу выбора из очереди в порядке поступления. Содержимое ячейки POS увеличивается на единицу при записи в очередь нового пассажира. Максимальное значение переменных PER и POS равно LM, поэтому их изменение происходит в соответствии с формулами:

$$\text{PER} = \begin{cases} \text{PER} + 1, & \text{если PER} < \text{LM}; \\ \text{PER} + 1 - \text{LM}, & \text{если нет} \end{cases}$$

$$\text{POS} = \begin{cases} \text{POS} + 1, & \text{если POS} < \text{LM}; \\ \text{POS} + 1 - \text{LM}, & \text{если нет.} \end{cases}$$

Переменная NP — число пассажиров в очереди, при выборе из очереди NP уменьшается на единицу, а при записи в очередь — увеличивается на единицу.

Переменная PUST равна единице, если в очереди нет пассажиров, и равна нулю в противном случае.

Переменная POLN равна единице, если в очереди нет свободных мест (в этом случае новый пассажир не может быть записан в очередь).

Переменная WYB должна принимать значение 1, если обращение к алгоритму, имитирующему работу очереди, производится с целью выборки из последней. Если же обращение к этому алгоритму производится для записи, то необходимо установить WYB = 0.

Блок-схема алгоритма, имитирующего работу очереди 1, приведена на рис. 11.8 (все переменные этого алгоритма в отличие от переменных алгоритма очереди 2, имеют в своих названиях цифру 1).

Работу кассы можно описать признаком «свободно — занято», изменяющимся во времени. В качестве этого признака можно взять двоичную переменную KSW (KSW = 1, если касса свободна; KSW = 0, если касса занята).

Пусть предыдущий момент освобождения кассы будет T_3. Тогда в соответствии с логикой работы системы производится обращение к очереди 1, и если она не пуста, следующий момент освобождения кассы будет

$$T_3 = T_3 + \tau_1.$$

Если же очередь 1 пуста, то происходит обращение к очереди 2. При этом возможны два варианта: 1) если эта очередь не пуста, то следующий момент освобождения кассы будет:

$$T_3 = T_3 + \tau_2;$$

2) если очередь 2 пуста, то устанавливается значение KSW = 0.

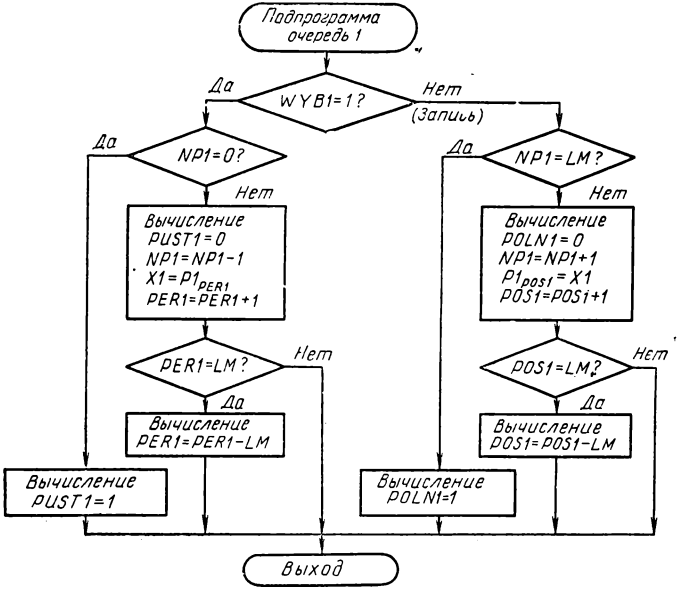


Рис. 11.8. Блок-схема алгоритма имитации работы очереди 1

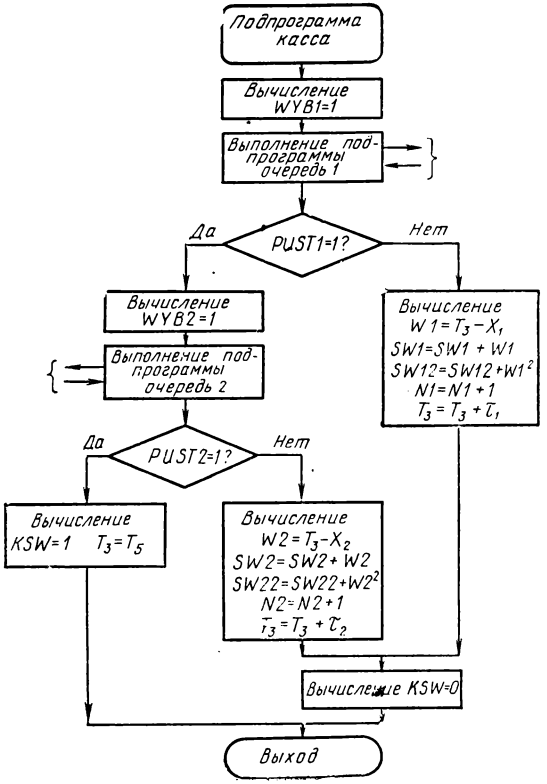


Рис. 11.9. Блок-схема алгоритма имитации работы кассы

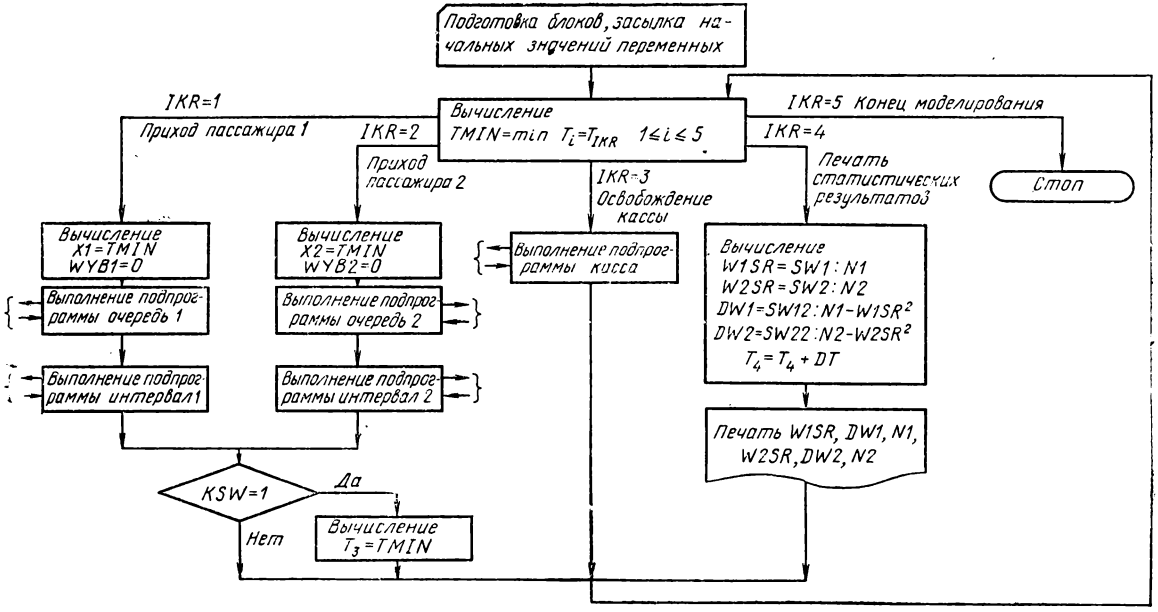


Рис. 11.10. Общий алгоритм моделирования работы кассы

В моменты выбора из очереди можно подсчитывать время ожидания пассажира, выбираемого из очереди на обслуживание кассой, по формуле

$$W = T_3 - X.$$

Для последующего определения среднего времени ожидания и дисперсии этой величины надо производить накапливание (суммирование) времени ожидания и квадрата этой величины:

$$SW = SW + W;$$
$$SW2 = SW2 + W^2;$$
$$N = N + 1.$$

Алгоритм, имитирующий логику работы кассы и одновременно производящий накопление статистических данных, приведен на рис. 11.9.

В определенные моменты времени (с периодом DT) надо производить вычисление и печать следующих статистических характеристик:

$$WSR = SW/N;$$
$$DW = SW2/(N - 1) - N/(N - 1)\, WSR^2.$$

Остается составить алгоритм, обеспечивающий правильную последовательность чередования событий в модели системы. Такими событиями являются приход в систему пассажиров первого или второго потока, освобождение кассы, печать статистических характеристик. Работа этого алгоритма основана на введении понятия *системного времени*, используемого для представления упорядоченных во времени событий. Системное время меняется дискретно, проходя последовательно через все моменты совершения событий. Роль алгоритма, управляющего последовательностью событий, заключается теперь в определении момента наступления ближайшего события и передаче управления тем алгоритмам, которые имитируют это событие. Момент наступления ближайшего события можно определить, если найти минимальный элемент в так называемом *списке будущих событий*, представляющем собой ближайшие моменты прихода пассажиров первого и второго потоков, ближайший момент освобождения кассы, ближайший момент печати статистических характеристик. Этот минимальный элемент определит новое значение системного времени и то, какое событие должно быть в этот момент осуществлено. После осуществления этого события производится обновление списка будущих событий путем вычисления нового момента времени, соответствующего типу совершившегося события.

Например, если минимальный элемент списка является моментом прихода пассажира первого потока, то необходимо обратиться к подпрограмме, реализующей алгоритм работы очереди 2, с целью записи пассажира в эту очередь, а затем выработать значение момента прихода следующего пассажира первого потока (новое будущее событие) в соответствии с алгоритмом, приведенным на рис. 11.7.

Блок-схема алгоритма всей модели представлена на рис. 11.10. В этой схеме в список будущих событий T_i включен также момент окончания моделирования.

Рассмотренные элементы моделирования показывают, что модели систем удобно строить по блочному принципу, причем каждый блок подчиняется собственной логике работы, имитирующей определенный процесс или устройство моделируемой системы. Некоторые из этих блоков могут быть реализованы как стандартные блоки путем включения их в библиотеки стандартных программ. Этот же принцип используется при разработке специальных языков моделирования, в которых типовые алгоритмы реализуются в виде операторов или стандартных процедур.

ГЛАВА 12

АНАЛОГОВЫЕ ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ

§ 12.1. ЭЛЕКТРИЧЕСКОЕ МОДЕЛИРОВАНИЕ

Понятие о моделировании. При исследовании различных процессов в науке и технике широко используется моделирование. *Моделированием* называют построение копии (модели) какого-либо процесса или объекта. Задача моделирования заключается в воспроизведении явления подобного оригиналу и его исследовании. Различают моделирование физическое и математическое.

При *физическом моделировании* исследования проводят на модели, подобной объекту, которая отличается от оригинала только размерами. Процессы, протекающие в модели и оригинале, имеют один и тот же физический характер.

Необходимость создания физических моделей диктуется многими важными практическими задачами. Очень часто процессы, протекающие во вновь создаваемых приборах и машинах, имеют сложный характер. При этом физическое моделирование позволяет не только исследовать такие процессы с наименьшими материальными затратами, но и уточнить их математическое описание в оригинале, находить значения параметров машин и приборов.

Физическое моделирование используют в различных отраслях машиностроения и приборостроения, например, при строительстве новых плотин, энергетических сетей и т. д. Без физического моделирования невозможно определить аэродинамические свойства вновь создаваемых образцов самолетов.

Математическое моделирование основано на идентичности дифференциальных уравнений, описывающих поведение оригинала и модели. Оно производится на модели, физическая природа которой может отличаться от физической природы оригинала.

Если в качестве модели используют электрические цепи, то такое моделирование называют *электрическим*. Можно выделить два направления в реализации электрических моделей: разработка эквивалентных схем (схем замещения или аналогий) и создание АВМ.

В первом случае исследование процессов, происходящих в какой-либо физической системе, производится на схеме, построенной на основе электрических цепей. Процессы, протекающие в исходной физической системе, и модели описываются идентичными дифференциальными уравнениями. При этом параметры исходной физической системы имеют свои аналоги в эквивалентной схеме. Наиболее широкое распространение при математическом моделировании нашли электрические схемы замещения.

Пример. Предположим, что требуется исследовать подвеску автомобиля, схема которой показана на рис. 12.1. Подвеска состоит из рычага, пружины с жесткостью Q, амортизатора с коэффициентом демпфирования P. При движении

автомобиля на колесо с массой m за счет неровностей дороги действует сила $F(t)$.

Подвеска представляет собой механическую колебательную систему с одной степенью свободы. Если пренебречь силами трения и люфтами в шарнирах, то перемещение колеса относительно среднего положения описывается дифференциальным уравнением второго порядка:

$$m\frac{d^2S}{dt^2}+P\frac{dS}{dt}+QS=F(t). \qquad (12\ 1)$$

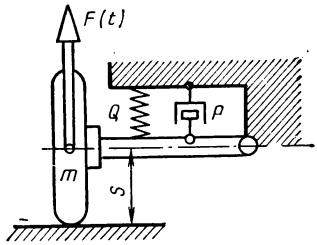


Рис. 12.1. Схема подвески автомобиля

При проектировании такого звена необходимо, чтобы оно обладало заданными свойствами, которые зависят от значений параметров m, P, Q.

Таким образом, задача сводится к выбору соотношений между параметрами подвески и определению их значений и может быть с успехом решена с помощью электрических цепей, состоящих из пассивных элементов R, L, C и работающих на постоянном или переменном токе (рис. 12.2).

В первой схеме (рис. 12.2, *а*) используется источник напряжения с малым внутренним сопротивлением. Величина и форма кривой э. д. с. источника напряжений остаются неизменными при изменении величины отдаваемого источником тока.

Во второй схеме (рис. 12.2, *б*) под источником тока i подразумевается источник с большим внутренним сопротивлением. Величина и форма кривой тока должны оставаться неизменными при изменении напряжений на его зажимах. Для этих схем в соответствии с теорией электромеханических аналогий можно написать уравнения, идентичные уравнению, описывающему поведение механической системы.

Если взять в качестве обобщенной координаты величину электрического заряда q, то процессы в первой схеме будут описываться уравнением

$$L\frac{d^2q}{dt_э^2}+R\frac{dq}{dt_э}+\frac{1}{C}q=u(t_э), \qquad (12.2)$$

где $t_э$ — время протекания процесса в электрической цепи.

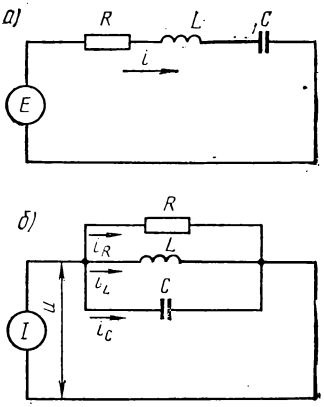


Рис. 12.2. Схемы электрических цепей для исследования механических колебательных систем с одной степенью свободы

Если же за обобщенную координату принять величину магнитного потока Φ, то во второй схеме электрические процессы будут описываться уравнением

$$C\frac{d^2\Phi}{dt_э^2}+\frac{1}{R}\frac{d\Phi}{dt_э}+\frac{1}{L}\Phi=i(t_э). \qquad (12.3)$$

Из сопоставления уравнений (12.1), (12.2) и (12.3), описывающих поведение механической системы и процессов, протекающих в электрических схемах, видно, что эти уравнения имеют одинаковый вид. В первом случае аналогом силы F является напряжение u, а во втором — ток i.

В силу идентичности уравнений процессы в механической системе можно изучить с помощью электрических цепей. При этом параметры электрических цепей R, C, L должны быть связаны определенными соотношениями с параметрами механической системы m, P и Q. Решение задачи сводится к подбору значений параметров R, C, L и измерению напряжений или токов в соответствующих точках схемы.

Электрические модели позволяют существенно сократить затраты на проектирование; упрощают исследование и облегчают измерение переменных величин. К достоинствам этих моделей следует отнести и их простоту; к недостаткам — их невысокую точность и сравнительную сложность составления моделей для дифференциальных уравнений высоких порядков.

Вторым направлением построения математических моделей являются АВМ. Они строятся из отдельных решающих блоков, которые выполняют элементарные математические операции над переменными напряжениями, например, сложение, умножение на постоянную величину, интегрирование и т. д. От электрических цепей моделей прямой аналогии вычислительные машины отличаются тем, что в них отсутствует аналогия между параметрами изучаемой физической системы и параметрами машины. Для решения конкретного уравнения решающие блоки АВМ соединяются так, чтобы получилась схема, процессы в которой описывались таким же образом, как и в исходном уравнении.

Пример. Рассмотрим подход к решению уравнений на АВМ на примере построения схемы для решения дифференциального уравнения второго порядка с постоянными коэффициентами

$$a_1 \frac{d^2x}{dt^2} + a_2 \frac{dx}{dt} + a_3 x = F(t). \qquad (12.4)$$

Существует два метода построения схемы: метод повышения и метод понижения порядка производной. При этих методах необходимо предварительно преобразовать уравнение к виду, облегчающему построение схемы решения уравнения. Метод повышения порядка производной требует разрешения уравнения относительно значения младшей производной, но этот метод имеет определенные недостатки и поэтому на практике наибольшее распространение получил метод понижения порядка производной, требующий разрешения уравнения относительно значения старшей производной. Для этого метода преобразованное уравнение имеет следующий вид:

$$\frac{d^2x}{dt^2} = -\frac{a_2}{a_1}\frac{dx}{dt} - \frac{a_3}{a_1}x + \frac{F(t)}{a_1}. \qquad (12.5)$$

Вторая производная равняется сумме трех членов, стоящих в правой части уравнения. Предположим, что существуют непрерывные напряжения, пропорциональные каждому из трех слагаемых. Если эти напряжения подать на вход блока, выполняющего операцию сложения, то на его выходе получится величина, пропорциональная второй производной. Для понижения порядка производной это напряжение подается на вход интегрирующего блока, после которого его величина пропорциональна значению первой производной. Подключение еще одного интегрирующего блока позволяет получить на его выходе величину напряжения, пропорциональную переменной x (рис. 12.3). Далее с помощью блоков умножения на постоянный коэффициент, изменяющих знак, функции одной переменной требуется обеспечить наличие напряжений на входе суммирующего блока (рис. 12.4). Для этого необходимо первую производную умножить на коэффициент a_2/a_1, а x — на коэффициент a_3/a_1. Величины x и x' в правой части уравнения стоят со знаком минус. Поэтому после умножения на соответствующие коэффициенты требуется поставить блоки, осуществляющие операцию изменения знака. Третье слагаемое образуется с помощью блока воспроизведения заданной нелинейной зависимости $F(t)$ и блока умножения на постоянный коэффициент.

В результате получается схема для решения дифференциального уравнения второго порядка, изображенная на рис. 12.4. Аналогичным образом составляются схемы и для решения дифференциальных уравнений n-го порядка.

Для метода понижения порядка производной в общем случае схемы составляются в следующей последовательности:

1) уравнение разрешается относительно старшего значения производной;

2) предполагается наличие переменных напряжений, пропорциональных членам в правой части уравнения;

3) переменные напряжения подаются на вход суммирующего блока, на выходе которого получается величина, пропорциональная старшей производной;

4) с помощью интегрирующих блоков производится понижение порядка производной (количество интегрирующих блоков численно должно равняться значению порядка производной);

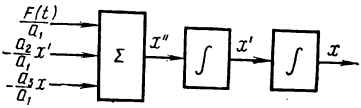


Рис. 12.3. Схема понижения порядка производной

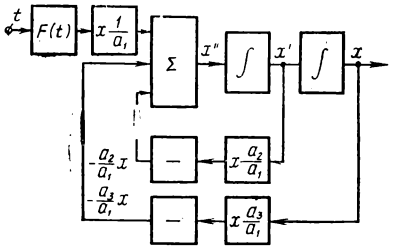


Рис. 12.4. Схема для решения дифференциального уравнения второго порядка

5) вводятся дополнительные блоки, обеспечивающие существование величин на входе суммирующего блока;

6) задаются начальные условия на интегрирующие блоки.

Решение задачи на АВМ сводится к измерению напряжений в различных точках схемы.

Структурная схема АВМ. Под АВМ понимают совокупность функциональных блоков, в которых происходят процессы, определяемые заданными математическими и логическими зависимостями.

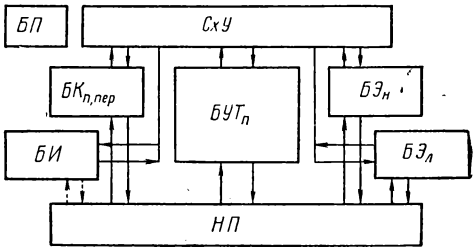


Рис. 12.5. Структурная схема АВМ

Функциональные блоки машины должны выполнять весь комплекс математических операций, требующихся для построения структуры исследуемых уравнений. Выше был рассмотрен пример моделирования дифференциального уравнения второго порядка. Для решения этого уравнения в состав машины входили блоки воспроизведения операции сложения, умножения переменной величины на постоянный коэффициент, получения функции переменного аргумента и операции интегрирования. Состав математических операций, которые должны выполняться при решении задач, и определяет структуру АВМ (рис. 12.5), где *БП* — блок питания, *СхУ* — схема управления; *НП* — наборное поле; $БК_{п.\ пер}$ — блок постоянных и переменных коэффициентов, $БУТ_{п}$ — блок усилителей постоянного тока, $БЭ_{н}$ — блок нелинейных элементов, $БЭ_{л}$ — блок линейных элементов, *БИ* — блок индикации.

Основным элементом АВМ является усилитель постоянного тока с большим коэффициентом усиления, на базе которого формируются практически все функциональные блоки машины. С помощью пассивных элементов, конденсаторов и сопротивлений, выделенных на структурной схеме АВМ в блок линейных элементов, строятся функциональные блоки для воспроизведения операций суммирования, интегрирования, умножения на постоянный коэффициент и т. д.

Блок постоянных и переменных коэффициентов в совокупности с усилителями постоянного тока дает возможность моделировать члены уравнения, содержащие постоянные коэффициенты и коэффициенты, зависящие от времени и произвольного аргумента.

Блок нелинейных элементов совместно с усилителями постоянного тока позволяет строить функциональные блоки для перемножения двух переменных величин, блоки для воспроизведения заданной нелинейной зависимости и типовых нелинейных зависимостей, которые часто встречаются при моделировании систем автоматического регулирования.

Набор структурной схемы для решения уравнения, т.е. связь между отдельными функциональными блоками, осуществляется на наборном поле с помощью гибкой шнуровой коммутации. Это позволяет легко вносить изменения при решении задачи в структурную схему.

Синхронизация работы всех блоков АВМ производится с помощью схемы управления, которая также обеспечивает контроль работы машины при подготовке к решению задачи и в процессе ее решения.

Для наблюдения за результатами решения используют блок индикации, который представляет собой набор регистрирующих и измерительных приборов (например, осциллограф, стрелочные и цифровые вольтметры).

§ 12.2. ОПЕРАЦИОННЫЕ БЛОКИ АВМ

Суммирование напряжений. Рассмотрим схему, состоящую из n параллельно включенных резисторов R_1, R_2, ..., R_n и резистора нагрузки $R_н$ (рис. 12.6). На входы резисторов R_1, R_2, ..., R_n подаются напряжения постоянного тока $e_1(t)$, $e_2(t)$, ..., $e_n(t)$. Предположим, что источники этих напряжений обладают нулевым внутренним сопротивлением, и обозначим через $u(t)$ напряжение на резисторе нагрузки. Тогда токи, протекающие в этой схеме, в соответствии с законом Ома можно выразить следующим образом:

$$i_1(t) = \frac{e_1(t) - u(t)}{R_1} = g_1\left[e_1(t) - u(t)\right];$$

$$i_2(t) = \frac{e_2(t) - u(t)}{R_2} = g_2\left[e_2(t) - u(t)\right];$$

. . . . . .

$$i_n(t) = \frac{e_n(t) - u(t)}{R_n} = g_n\left[e_n(t) - u(t)\right];$$

$$i_н(t) = \frac{u(t)}{R_н} = g_н u(t),$$

где g — проводимость.

В соответствии с первым законом Кирхгофа сумма токов, втекающих в узел, равна сумме токов, вытекающих из узла. Поэтому для точки A

$$i_1(t) + i_2(t) + \ldots + i_n(t) = i_н(t).$$

Подставляя в это уравнение значения токов и разрешая его относительно $u(t)$, получим

$$u(t) = \frac{g_1}{\sum_{i=1}^{n} g_i + g_н} e_1(t) + \ldots + \frac{g_n}{\sum_{i=1}^{n} g_i + g_н} e_n(t).$$

Введем обозначения

$$K_1 = \frac{g_1}{\sum_{i=1}^{n} g_i + g_н}, \quad K_2 = \frac{g_2}{\sum_{i=1}^{n} g_i + g_н}, \ldots, K_n = \frac{g_n}{\sum_{i=1}^{n} g_i + g_н},$$

тогда

$$u(t) = K_1 e_1(t) + K_2 e_2(t) + \ldots + K_n e_n(t). \qquad (12.7)$$

Из (12.7) видно, что напряжение $u(t)$ на резисторе нагрузки является суммой входных напряжений, умноженных на соответствующие *передаточные коэффициенты* K_j, которые отличаются один от другого только числителями.

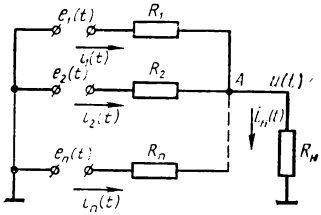


Рис. 12.6. Суммирующая цепочка

В процессе суммирования входных напряжений часто приходится изменять величины передаточных коэффициентов. Это необходимо делать, например, для того, чтобы выходное напряжение $u(t)$ не превышало предельно допустимого значения напряжения для данной схемы. Изменять величины передаточных коэффициентов можно путем изменения величины входного резистора R_j. Однако изменение величины хотя бы одного входного резистора приведет к изменению всех передаточных коэффициентов, так как в знаменателе стоит сумма всех входных проводимостей. Аналогично будут изменяться коэффициенты передачи при изменении сопротивления нагрузки и количества слагаемых, т. е. количества входных резисторов. Для устранения этого недостатка суммирующей цепочки надо таким образом выбирать величины резисторов, чтобы выполнялось неравенство

$$\sum_{i=1}^{n} g_j \ll g_n.$$

Это возможно при условии, что $R_н$ имеет величину, близкую к нулю. Величиной суммы можно пренебречь и тогда коэффициенты передачи будут равны

$$K_j \approx g_j / g_н = R_н / R_j. \qquad (12.8)$$

Выбор малой величины сопротивления нагрузки приводит к тому, что уменьшаются коэффициенты передачи, а следовательно, и суммарное напряжение на выходе цепочки. Если при решении задачи необходимо последовательно соединить несколько цепочек, то напряжение малой величины с выхода первой цепочки будет подаваться на один из входов следующей цепочки и умножаться на коэффициент передачи, значительно меньший единицы. Напряжение на выходе второй цепочки будет близким к нулю и его достаточно трудно измерить. Вследствие этого работать с последовательно соединенными цепочками невозможно.

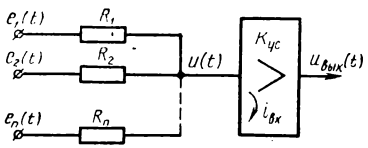


Рис. 12.7. Суммирующая цепочка с УПТ

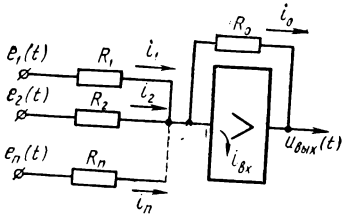


Рис. 12.8. Схема суммирующего ОУПТ

Одним из путей улучшения характеристик суммирующей цепочки является подключение к ее выходу усилителя постоянного тока УПТ, как это показано на рис. 12.7. В этой схеме роль резистора нагрузки играет входное сопротивление усилителя.

Если обозначить через $K_{ус}$ коэффициент усиления УПТ, то напряжение на его выходе

$$u_{вых}(t) = u(t) K_{ус}$$

и значение коэффициента по j-му входу

$$K_j = K_{ус} R_н / R_j.$$

Использовать такую схему для суммирования напряжений можно только при условии, что $K_{ус}$ стабильно во времени. УПТ строятся на основе электронных ламп и коэффициент усиления $K_{ус}$ определяется их параметрами. Значения параметров электронных ламп зависят от ряда случайных факторов, например, от окружающей температуры, стабильности источников питания и т. д. Поэтому создать УПТ с постоянным $K_{ус}$ практически невозможно.

Для стабилизации коэффициента усиления усилителя используют отрицательную обратную связь. Под *обратной связью* понимают передачу части напряжения с выхода усилителя на его вход. Если эти напряжения противоположны по фазе, т. е. если на входе усилителя положительное напряжение, а на выходе — отрицательное, и наоборот, то такую обратную связь будут называть *отрицательной*. Для того чтобы обеспечить отрицательную обратную связь, УПТ должен иметь нечетное число каскадов и изменять знак входного напряжения.

Введем в схему (см. рис. 12.7) резистор обратной связи. При этом получается схема, изображенная на рис. 12.8. Здесь через резистор

обратной связи к суммирующей точке, т. е. на вход усилителя, подается часть напряжения $u_{вых}$.

Если на резисторы подается напряжение, то во входной цепи усилителя протекает ток $i_{вх}$. В реальных схемах усилителей этот ток практически равен нулю. Поэтому для входа усилителя уравнение Кирхгофа можно записать так:

$$i_1(t) + i_2(t) + \ldots + i_n(t) = i_0(t).$$

Ток $i_0(t)$ через резистор обратной связи

$$i_0(t) = \left(u(t) - u_{вых}(t)\right)/R_0.$$

После подстановки в уравнение Кирхгофа значений токов $i_1(t)$, $i_2(t)$, ..., $i_n(t)$ и $i_0(t)$ получим

$$\frac{e_1(t) - u(t)}{R_1} + \frac{e_2(t) - u(t)}{R_2} + \ldots + \frac{e_n(t) - u(t)}{R_n} = \frac{u(t) - u_{вых}}{R_0}. \quad (12.9)$$

Так как для получения отрицательной обратной связи УПТ имеет нечетное число каскадов, то

$$u_{вых}(t) = -u(t)\,K_{ус}.$$

отсюда

$$u(t) = -u_{вых}(t)/K_{ус}.$$

Подставим в (12.9) значение $u(t)$ и разрешим его относительно $u_{вых}(t)$

$$u_{вых}(t) = -\frac{\sum_{j=1}^{n} e_j(t)/R_j}{1/R_0 + 1/K_{ус}\left[1/R_0 + \sum_{j=1}^{n}(1/R_j)\right]}. \quad (12.10)$$

Потребуем, чтобы величина коэффициента усиления усилителя была очень большой. Тогда величина $1/K_{ус}$ будет очень малой и вторым членом в знаменателе можно пренебречь. Поэтому

$$u_{вых}(t) = -R_0 \sum_{j=1}^{n} e_j(t)/R_j \quad (12.11)$$

и коэффициент передачи по j-му входу

$$K_j = R_0/R_j.$$

Анализируя последние выражения, можно сделать следующие выводы.

Во-первых, $u_{вых}(t)$ не зависит от величины $K_{ус}$, поэтому нет необходимости предъявлять жесткие требования к его стабильности, достаточно, чтобы он был большим. АВМ используют усилители с коэффициентом усиления в диапазоне от 10^4 до 10^6.

Во-вторых, коэффициент передачи по j-му входу не зависит от сопротивления нагрузки, как это имело место в суммирующей цепочке, кроме того, он не зависит и от числа слагаемых.

Таким образом, введение УПТ с отрицательной обратной связью позволяет устранить недостатки, присущие простой суммирующей цепочке и получить суммирующий операционный УПТ (ОУПТ) для моделирования операции сложения.

Если в схеме суммирующего блока использовать только один вход, то получится блок (рис. 12.9, *а*), который осуществляет умножение входного напряжения на постоянный коэффициент:

$$u_{\text{вых}}(t) = -\text{e}(t)\, R_0/R_1. \tag{12.12}$$

Часто этот блок называют *масштабным*. Величину коэффициента в масштабном блоке можно изменять, меняя отношение $R_0 | R_1$, т. е.

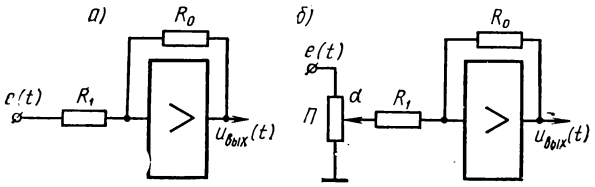


Рис. 12.9. Схемы масштабных блоков

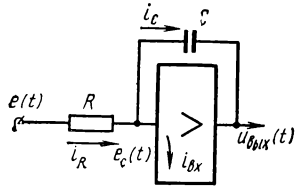


Рис. 12.10. Интегрирующий ОУПТ

величину сопротивлений. Кроме того, входное напряжение можно подавать через потенциометр, как показано на рис. 12.9, *б*. При этом

$$K = \alpha R_0/R_1,$$

где $0 \leqslant \alpha \leqslant 1$.

Если резисторы R_1 и R_0 в схеме, показанной на рис. 12.9 *а*, равны, то блок осуществляет операцию

$$u_{\text{вых}} = -\text{e}(t). \tag{12.13}$$

Такой блок, называемый *инвертирующим*, умножает входное напряжение на —1, т. е. изменяет знак входного напряжения.

Интегрирующий операционный усилитель постоянного тока. Схема интегрирующего операционного усилителя показана на рис. 12.10. На входе усилителя стоит резистор R и в цепь обратной связи включен конденсатор C.

Напряжение на выходе интегрирующего операционного усилителя постоянного тока описывается уравнением

$$u_{\text{вых}}(t) = -\frac{1}{RC}\int_0^t \text{e}(t)\, dt + u_{\text{вых}_0}, \tag{12.14}$$

где $u_{\text{вых}_0}$ — постоянная интегрирования.

Схема осуществляет операцию интегрирования входного напряжения по времени с коэффициентом передачи $1/RC$.

Рассмотрим схему (рис. 12.11), с помощью которой можно задавать на интегрирующий усилитель напряжение начальных условий $u_{\text{вых}_0}$, т. е. постоянную интегрирования. Эта схема состоит собственно из ин-

тегрирующего усилителя с конденсатором C в цепи обратной связи и резистором R на входе. Потенциометр $П$ с заземленной средней точкой, резисторы R_1, R_2, R_3 и контакты реле P_1, P_2 относятся к схеме задания начальных условий.

При установке начальных условий замыкаются верхние контакты реле P_1 и P_2, в результате чего конденсатор C включается на выход усилителя, а вход усилителя подключается к потенциометру начальных условий $П$. Усилитель при этом работает в режиме масштабного блока. Напряжение $u_{вх_0}$, устанавливаемое потенциометром $П$, заряжает кондесатор C, создавая требуемое значение выходного напряжения

$$u_{вых_0} = -u_{вх_0} R_2/R_1.$$

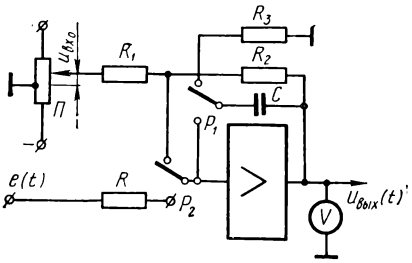


Рис. 12.11. Схема задания начальных условий

Контроль выходного напряжения производится по вольтметру V. После установки напряжения начальных условий контакты реле P_1 и P_2 переключаются в нижнее положение, переводя усилитель в режим интегратора.

При решении задач на АВМ достаточно часто возникает необходимость интегрирования алгебраической суммы напряжений. Эта операция может быть выполнена с помощью операционного усилителя, схема которого показана на рис. 12.12.

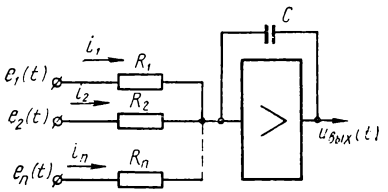


Рис. 12.12. Интегросуммирующий ОУПТ

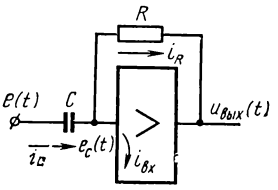


Рис. 12.13. Дифференцирующий ОУПТ

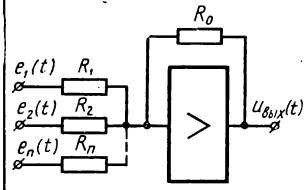
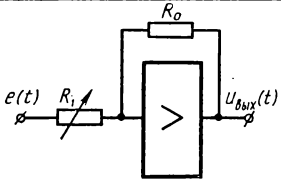
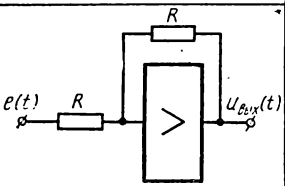
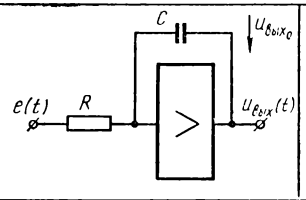
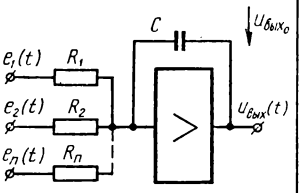
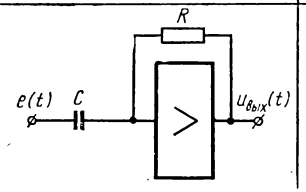
Напряжение на выходе усилителя описывается уравнением

$$u_{вых}(t) = -\frac{1}{C}\int_0^t \left(\sum_{j=1}^{n} \frac{e_j(t)}{R_j}\right) dt + u_{вых_0}. \qquad (12.15)$$

Каждое из входных напряжений умножается на соответствующий передаточный коэффициент, равный $1/(R_1C)$, $1/(R_2C)$, ..., $1/(R_nC)$. Таким образом, с помощью этого блока можно одновременно выполнять операции сложения, интегрирования суммы и умножения на постоянный коэффициент.

Если на входе УПТ стоит конденсатор C, а в цепи обратной связи — резистор R (рис. 12.13), то такая схема выполняет операцию дифференцирования входного напряжения.

Таблица 12.1

Название блока	Схема блока	Машинное уравнение	Операция	Коэффициент передачи
Суммирующий		$U_{вых}(t)=-R_0\sum_{j=1}^{n}\frac{e_j(t)}{R_j}$	$y=\sum_{j=1}^{n}a_i x_i$	$K_j=\frac{R_0}{R_j}$
Масштабный		$U_{вых}(t)=-\frac{R_0}{R_1}e(t)$	$y=ax$	$K=\frac{R_0}{R_1}$
Инвертор		$U_{вых}(t)=-e(t)$	$y=-x$	$K=1$
Интегрирующий		$U_{вых}(t)=-\frac{1}{RC}\int_0^t e(t)dt+U_{вых_0}$	$y=a\int_0^t x\,dt$	$K=\frac{1}{RC}$
Интегросуммирующий		$U_{вых}(t)=-\frac{1}{C}\int_0^t\left(\frac{e_1(t)}{R_1}+\frac{e_2(t)}{R_2}+\ldots+\frac{e_n(t)}{R_n}\right)dt+U_{вых_0}$	$y=\int_0^t\sum_{j=1}^{n}a_j x_j\,dt$	$K_j=\frac{1}{R_jC}$
Дифференцирующий		$U_{вых}(t)=-RC\frac{de(t)}{dt}$	$y=a\frac{dx}{dt}$	$K=RC$

Напряжение на выходе дифференцирующего блока описывается уравнением

$$u_{\text{вых}} = -RC\frac{de(t)}{dt}. \qquad (12.16)$$

Дифференцирующий усилитель обладает повышенной чувствительностью к помехам, что приводит к существенным искажениям выходного сигнала. Поэтому дифференцирующие усилители практически не применяют в АВМ.

Наиболее часто встречающиеся блоки сведены в табл. 12.1.

§ 12.3. ФУНКЦИОНАЛЬНЫЕ НЕЛИНЕЙНЫЕ ПРЕОБРАЗОВАТЕЛИ

Функциональными нелинейными преобразователями называют устройства для воспроизведения заданной функции одного или нескольких произвольных аргументов. Принято делить все функциональные преобразователи на универсальные и специализированные.

К *универсальным преобразователям* относят устройства для воспроизведения широкого класса функций.

К *специализированным преобразователям* относят устройства, предназначенные для воспроизведения только определенных зависимостей, например таких характеристик как люфт, зона нечувствительности, ограничение.

Среди универсальных функциональных устройств можно выделить устройства для воспроизведения функций, зависящих от времени и устройства для моделирования функций, зависящих от произвольного аргумента.

Функциональные преобразователи с временным аргументом. Предположим, что с помощью АВМ требуется решить следующее дифференциальное уравнение:

$$\frac{d^2x}{dt^2} + a_1(t)\frac{dx}{dt} + a_2(t)\,x = F(t),$$

где $a_1(t)$ и $a_2(t)$ — переменные коэффициенты; $F(t)$ — внешнее возмущающее воздействие.

Для решения этого уравнения необходимо иметь устройство воспроизведения заданной нелинейной функции $F(t)$.

Наибольшее распространение в АВМ для набора функций, зависящих от времени, получили секционированные потенциометры, работающие совместно с шаговым искателем (рис. 12.14). Схема такого функционального преобразователя состоит из секционированного потенциометра, шагового искателя и генератора импульсов.

Секционированный потенциометр представляет собой последовательно соединенные резисторы r одинаковой величины. Обычно потенциометры содержат по 50 или 100 резисторов. От каждого резистора потенциометра сделан отвод к клемме. На верхний конец потенциометра подается постоянное напряжение E. Допустим, что $E = 100$ В и по-

тенциометр содержит 50 резисторов r. Тогда напряжение на клемме 0 будет равно 100 В, на клемме 1 — 98 В, на клемме 2 — 96 В и т. д. На клемме 49 напряжение равно 0 В. Шаговый искатель обеспечивает перемещение ползушки $П$ по токосъемной шине $Ш$ и подключение ее к ламелям. В начальный момент времени ползушка находится в верхнем положении и соединяет ламель 0 с токосъемной шиной $Ш$. При включении генератора импульсов $ГИ$ импульсы поступают на обмотку возбуждения $ОВ$ шагового искателя.

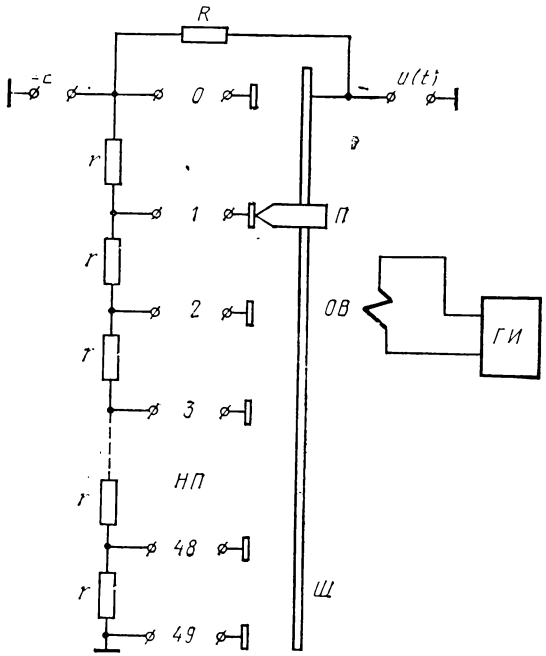


Рис. 12.14. Схема секционированного потенциометра

Под действием первого импульса ползушка переместится на одно положение и подключит токосъемную шину $Ш$ к ламели 1 шагового искателя. Второй импульс переместит ползушку еще на одно положение и подключит шину к ламели 2 и т. д. (все ламели шагового искателя соединены с клеммами).

Совокупность клемм секционированного потенциометра и клемм, подключенных к ламелям шагового искателя, представляет собой наборное поле. Здесь производится набор заданной функциональной зависимости $F(t)$ с помощью шнуровой коммутации путем соединения клемм потенциометра с ламелями шагового искателя.

В каждый момент времени величина выходного напряжения $u(t)$, снимаемого с токосъемной шины $Ш$, определяется положением ползушки $П$ и изменяется скачком при переходе с одной ламели на другую. Следовательно, заданная функция $F(t)$ будет иметь ступенчатую аппроксимацию.

Заданная функция $F(t)$ набирается на функциональном преобразователе следующим образом. Предварительно заданная кривая должна трансформироваться в машинные координаты «напряжение — время». Обычно за максимальное значение функции принимается максимально допустимое напряжение, равное 100 В. Временной интервал разбивается на 50 отрезков и по графику для каждого момента времени определяется значение напряжения. Ламели шагового искателя последовательно соединяются с теми клеммами потенциометра, на которых имеется соответствующее данному моменту времени напряжение.

На рис. 12.15 показан в виде ступенек график напряжения, получаемого на выходе функционального преобразователя.

Подключение преобразователя к другим устройствам вызывает погрешности от действия сопротивления нагрузки, которое шунтирует секционированный потенциометр. Для уменьшения этой ошибки специально введен резистор R. Следует отметить, что в генераторах импульсов, управляющих работой шагового искателя, обычно предусматривается возможность изменения тактовой частоты в достаточно широких пределах. Кроме того, они имеют такие режимы работы, когда частоту импульсов в течение одного цикла работы шагового искателя можно изменять. Это обеспечивается специальными программными устройствами.

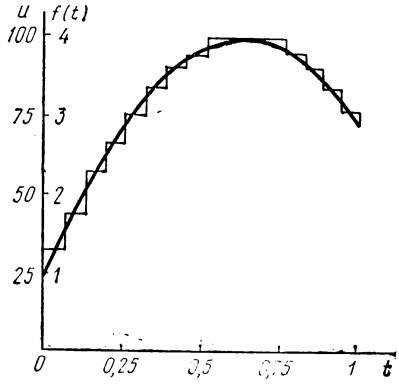


Рис. 12.15. Ступенчатая аппроксимация нелинейной зависимости

С помощью рассмотренного функционального преобразователя можно не только набирать заданную функцию $F(t)$, но и воспроизводить зависимость вида $a_1(t)\frac{d^n x}{dt^n}$. Так, например, в исходном уравнении имеются переменные коэффициенты $a_1(t)$ и $a_2(t)$. Набор этих коэффициентов возможно произвести с помощью функционального преобразователя следующим образом: нелинейный коэффициент $a_1(t)$ трансформируется в машинные координаты «напряжение — время» и набирается на наборном поле. На вход секционированного потенциометра в этом случае подается не постоянное напряжение E, а напряжение, моделирующее первую производную dx/dt. На выходе преобразователя будет напряжение $a_1(t)\frac{du}{dt}$.

Аналогично получается и коэффициент $a_2(t)$. Коэффициент набирается на функциональном преобразователе и на его вход подается напряжение, пропорциональное x.

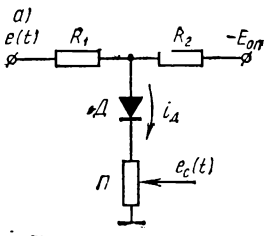
Нелинейный функциональный преобразователь произвольного аргумента. В общем случае схема нелинейного блока для воспроизведения функции произвольного аргумента имеет вид, показанный на рис. 12.16.

На входе блока ставится нелинейный элемент, сопротивление которого зависит от амплитуды и знака входного напряжения. Обычно в качестве нелинейных элементов используют схемы на диодах. На рис. 12.17, *а* показана типовая схема нелинейного элемента, который состоит из резисторов R_1, R_2, диода $Д$ и потенциометра $П$.

На резистор R_2 подается линейно изменяющееся напряжение $e(t)$, на резистор R_2 — постоянное отрицательное напряжение $-E_{оп}$. Рассмотрим работу этой схемы.

В начальный момент времени, когда e $(t) = 0$, на аноде диода $Д$ будет отрицательный потенциал. Следовательно, ток через диод $Д$ протекать не будет и $e_C(t) = 0$. Если увеличивать e (t), то при e $(t) = E_{оп}R_1/R_2$ положительное напряжение скомпенсирует отрицательное напряжение на аноде диода. При этом напряжение на аноде диода равно нулю и диод находится на грани открывания. Дальнейшее увеличение входного напряжения приводит к тому, что на аноде диода будет положительный потенциал. Через диод протекает ток $i_д$, который создает падение напряжения на потенциометре $П$.

Характеристика такого элемента представлена на рис. 12.17, *б*. Угол наклона характеристики α зависит в общем случае от величин резисторов R_1, R_2 и величины сопротивления потенциометра между движком и землей. Величину угла α можно регулировать, изменяя положение движка потенциометра, а момент открывания диода $Д$ — изменяя величины резисторов R_1 и R_2.



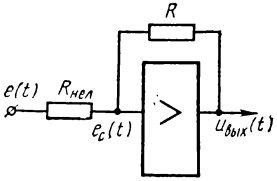


Рис. 12.16. Схема ОУПТ с нелинейным элементом на выходе

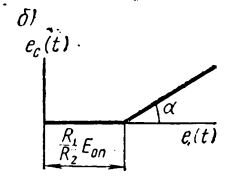


Рис. 12.17. Нелинейный элемент

Показанный на рис. 12. 17, *а* элемент позволяет получить характеристику в первом квадранте. Изменяя полярность включения диода, а также полярности входного и опорного напряжения,можно получать линейные отрезки во всех четырех квадрантах. Схемы таких элементов и их характеристики даны в табл. 12.2.

Напряжение $e_C(t)$ с выхода нелинейного элемента подается на вход УПТ (см. рис. 12.16), в цепь обратной связи которого включен резистор R. Поэтому необходимо учитывать, что напряжение на выходе усилителя будет получаться с обратным знаком.

Параллельное подключение нелинейных элементов к входу УПТ позволяет получить схему, сопротивление которой описывается функцией, имеющей вид ломаной линии. Однако при решении задач на АВМ, как правило, встречается обратная задача, т. е. требуется подобрать величины сопротивлений нелинейных элементов под заданную нелинейную функцию.

Как известно, любую однозначную гладкую линейную функцию можно аппроксимировать с помощью конечного числа линейных отрез-

ков. При этом заданная функция записывается в виде многочлена

$$f(x)=f_0+Kx+$$

$$+\sum_{j=1}^{n}\alpha_j(x-x_{j\,\text{нач}}),\quad (12.17)$$

где f_0 — значение функции при $x=0$; K — угол наклона первого линейного участка; α_j — угол наклона j-го линейного участка; $x_{j\,\text{нач}}$ — значение аргумента x в начале j-го линейного участка

$$\alpha_j=\begin{cases}0 & x\leqslant x_{j\,\text{нач}};\\ \text{const} & x>x_{j\,\text{нач}}.\end{cases}$$

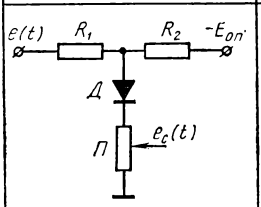
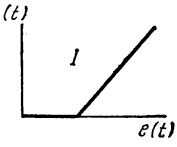
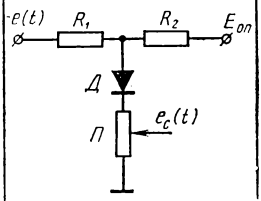
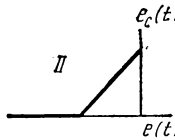
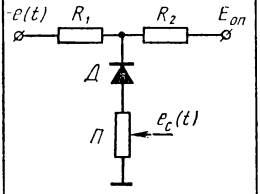
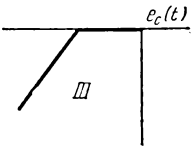
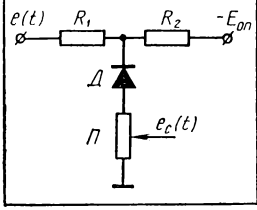
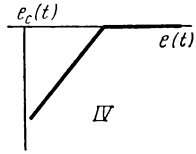
Уравнение (12.17) обычно реализуется в АВМ с помощью нелинейного диодного функционального преобразователя, схема которого показана на рис. 12.18, и при этом трансформируется в следующее машинное уравнение:

$$u_{\text{вых}}=u_0+K\mathrm{e}(t)+$$

$$+\sum_{j=1}^{n}\alpha_j[\mathrm{e}(t)-\mathrm{e}_{\text{вх}\,j\,\text{нач}}].$$

(12.18)

Таблица 12.2

Схема нелинейного сопротивления	Характеристика
	
	
	
	

Постоянная составляющая устанавливается с помощью потенциометра $П_1$, на который можно подавать либо положительное, либо отрицательное постоянное напряжение с помощью переключателя K_1. Потенциометр $П_2$ служит для установки угла наклона первого линейного участка. На этот потенциометр подается с помощью переключателя K_2 отрицательное или положительное напряжение e (t).

Нелинейные элементы $R_{\text{н}_1} \div R_{\text{н}_n}$ представляют собой схемы, которые рассмотрены выше и показаны в табл. 12.2. В зависимости от того, в каком квадранте требуется получить характеристику, на вход нелинейных элементов подается с помощью переключателей либо положительное, либо отрицательное напряжение e (t). Все линейные отрезки суммируются на первом усилителе. На выходе этого усилителя получается $u_{\text{вых}} = -f$ (e). Второй усилитель является инвертирующим и на его выходе будет $u_{\text{вых}} = f$ (e).

На рис. 12.19 показана аппроксимация функции тремя линейными отрезками. Очевидно, что точность аппроксимации зависит от количества линейных участков. Обычно в АВМ в нелинейных функциональных преобразователях используется от восьми до двадцати нелинейных элементов.

Далее при составлении структурных схем для решения задач на АВМ нелинейный блок НБ воспроизведения функции одной переменной схематически будем изображать так, как показано на рис. 12.20.

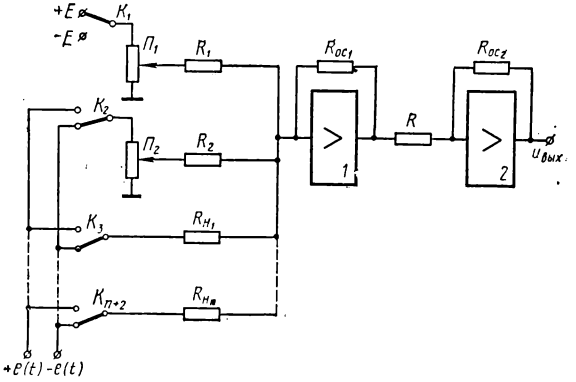


Рис. 12.18. Нелинейный функциональный преобразователь

На вход блока должны подаваться положительное и отрицательное напряжения e (*t*), которые необходимы для набора линейных отрезков при аппроксимации заданной функции во всех четырех квадрантах.

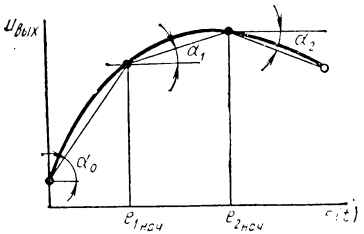


Рис. 12.19. Аппроксимация функции линейными отрезками

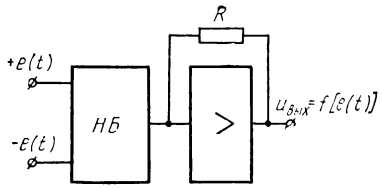


Рис. 12.20. Схема блока для воспроизведения функции одной переменной

Блоки для перемножения двух переменных напряжений. Устройства для перемножения двух переменных напряжений могут быть представлены в общем случае схемой, изображенной на рис. 12.21. Здесь e_1 и e_2 входные напряжения, подлежащие перемножению. Выходное напряжение $u_{вых}$ пропорционально произведению двух входных напряжений:

$$u_{вых} = ce_1e_2, \qquad (12.19)$$

где c — постоянная.

Существует несколько методов выполнения операции умножения. Наиболее широкое распространение получил метод выполнения операции умножения в соответствии с уравнением

$$x_1 x_2 = \left[\left(\frac{x_1 + x_2}{2} \right)^2 - \left(\frac{x_1 - x_2}{2} \right)^2 \right].$$

При этом умножение производится косвенным путем за счет автоматического выполнения вспомогательных операций. Устройство, реализующее это уравнение, называемое *блоком перемножения* на квадраторах, показано на рис. 12.22. Оно состоит из схем выделения модулей полусуммы и полуразности, возведения в квадрат и вычитания.

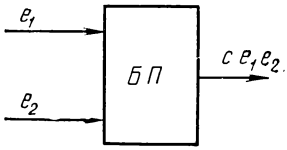


Рис. 12.21. Блок переключения

Операция возведения в квадрат в этих устройствах реализуется с помощью диодно-функциональных преобразователей, использующих линейно-кусочную аппроксимацию параболы. На них требуется получить квадрат знакопеременной величины, так как знаки выражений в круглых скобках изменяются в зависимости от значений x_1 и x_2. Квадраторы позволяют возводить в квадрат входное напряжение только при одном знаке. Поэтому требуется подавать на входы квадраторов модули алгебраической полусуммы и полуразности.

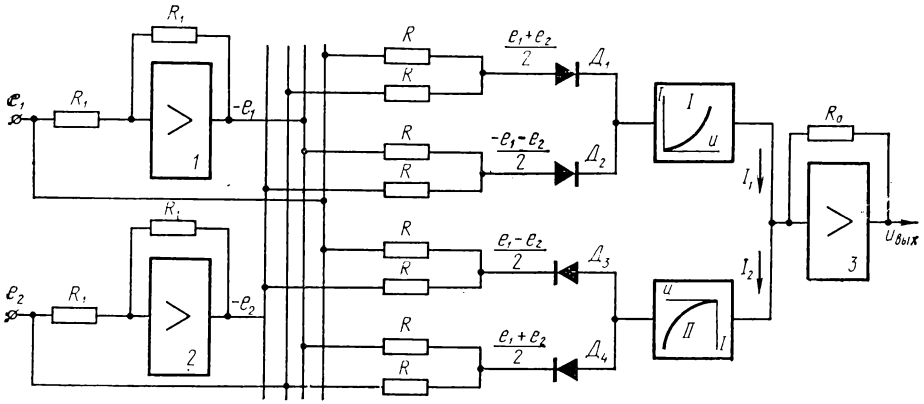


Рис. 12.22. Схема блока перемножения на квадраторах

Операция суммирования в блоке перемножения (рис. 12.22) осуществляется суммирующими цепочками, а выделение моделей образованных полусумм — диодами $Д_1$ и $Д_2$. Схемы суммирования образуют одновременно положительную и отрицательную полусумму и полуразность. Положительная и отрицательная полусуммы поступают на аноды диодов $Д_1$ и $Д_2$. В зависимости от величины и знака потенциала, приложенного к анодам этих диодов, будет включаться один из них. При включении одного из диодов другой запирается. Поэтому на квад-

ратор *I* всегда будет поступать положительный модуль полусуммы $(e_1 + e_2)/2$. Аналогичным образом происходит выделение модуля полуразности и на квадратор *II* поступает только отрицательная полуразность $-(e_1 - e_2)/2$.

Операционный усилитель *3* вырабатывает напряжение $u_{вых}$, пропорциональное разности токов I_1 и I_2, зависящих от квадратов $(e_1 + e_2)/2$ и $(e_1 - e_2)/2$, и пропорциональное произведению e_1 на e_2.

Для работы блока перемножения во всех четырех квадрантах на его вход должны подаваться сомножители e_1 и e_2 со своим и обратным знаком. С этой целью входные напряжения инвертируются усилителями *1* и *2*.

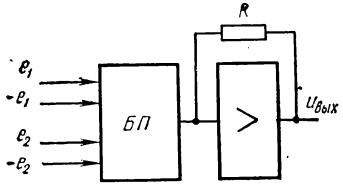


Рис. 12.23. Блок перемножения

Погрешность блоков перемножения, построенных в соответствии с рассмотренной схемой, характеризуется максимальной относительной ошибкой порядка 0,5—1%.

В дальнейшем, при составлении структуры схем для решения уравнений на АВМ, блок перемножения будем изображать так, как показано на рис. 12.23.

Следует отметить, что напряжения, моделирующие сомножители могут изменяться в диапазоне ±100 В. Напряжение $u_{вых}$ также не должно выходить за эти пределы. В связи с этим величину постоянного коэффициента с принимают равной 0,01 и блок выполняет операцию

$$u_{вых} = 0{,}01 e_1 e_2. \qquad (12.20)$$

§ 12.4. ПОДГОТОВКА ЗАДАЧ К РЕШЕНИЮ НА АВМ

Дифференциальные уравнения для исследования на АВМ могут задаваться:

а) в виде одного дифференциального уравнения, записанного относительно исследуемой координаты:

$$a_0 \frac{d^n x}{dt^n} + a_1 \frac{d^{n-1} x}{dt^{n-1}} + \ldots + a_{n-1} \frac{dx}{dt} + a_n x =$$

$$= b_0 \frac{d^m y}{dt^m} + b_2 \frac{d^{m-1} y}{dt^{m-1}} + \ldots + b_{m-1} \frac{dy}{dt} + b_m y,$$

где a и b — коэффициенты уравнения; x — исследуемая координата; y — возмущение, действующее на систему;

б) в виде системы дифференциальных уравнений первого порядка:

$$\frac{dx_j}{dt} + \sum_{k=1}^{n} a_{jk} x_k + F_j(t) = 0, \quad j = 1, 2, \ldots,$$

где $F_j(t)$ — заданные нелинейные функции;

в) в виде уравнений динамических звеньев:

$$T_1 \frac{d^2 x_2}{dt^2} + x_2 = k_1 x_1;$$

$$T_2 = \frac{dx_3}{dt} + x_3 = k_2 x_2;$$

$$T_3 T_4 \frac{d^2 x_4}{dt^2} + T_5 \frac{dx_4}{dt} + x_4 = k_3 x_3;$$

$$\frac{dx_5}{dt} = k_4 x_4;$$

$$x_1 = y - x_5.$$

Подготовка заданного уравнения или системы для исследования на АВМ состоит из следующих этапов: составления структурной схемы; определения масштабов переменных; расчета коэффициентов передачи функциональных блоков; определения напряжений начальных условий.

Для решения задачи на АВМ из функциональных блоков составляется структурная схема, в которой над переменными напряжениями должны осуществляться преобразования, аналогичные преобразованиям над переменными в исходном уравнении (порядок составления структурной схемы методом понижения порядка производной был рассмотрен выше). При этом структурные схемы составляются из функциональных блоков (см. табл. 12.1). Каждый из этих блоков изменяет знак входного напряжения, что необходимо учитывать при составлении структурной схемы.

Выбор масштабов переменных. При решении на АВМ уравнения переменные этого уравнения изображаются соответствующими напряжениями, называемыми *машинными переменными*. Для того чтобы машинные переменные изменялись по тем же законам, что и переменные в исходном уравнении, они должны быть связаны определенными соотношениями — масштабами. *Масштабом* называют отношение машинной переменной u_x и переменной исходного уравнения x, т. е.

$$M_x = \frac{u_x}{x}. \qquad (12.21)$$

Для получения наименьшей относительной ошибки u_x принимают равным максимально возможному напряжению на выходе УПТ. В большинстве отечественных АВМ напряжение на выходе УПТ изменяется в линейном диапазоне ± 100 В. В противном случае усилитель будет работать на нелинейном участке, что внесет значительные погрешности в решение задачи. Поэтому масштаб переменной должен определяться из условия

$$M_x = \frac{u_{x\,\max}}{x_{\max}} = \frac{100}{x_{\max}}, \qquad (12.22)$$

где $u_{x\,\max}$ — максимальное значение машинной переменной; $x_{\max}$ — максимальное значение переменной исходного уравнения.

Часто при постановке задачи исследователь знает предельные значения переменных величин (дальность полета объекта, максимальная скорость и др.). При отсутствии каких-либо данных о решаемой задаче руководствуются максимальными значениями переменных ранее решаемых аналогичных задач, если таковые не имеются, то максимальные значения берут произвольно. На их основании рассчитываются масштабы и осуществляется пробное решение задачи на машине. При этом возможны следующие случаи.

1. Значение x_{max} взято больше, чем есть на самом деле. Если в действительности x меняется в меньших пределах, то напряжение, пропорциональное x, будет небольшим. При этом для уменьшения относительной ошибки необходимо уменьшить x_{max}, т. е. увеличить масштаб.

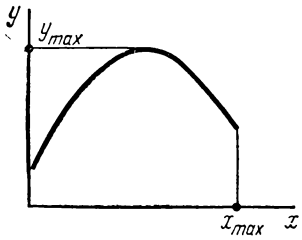


Рис. 12.24. График нелинейной функции

2. Значение x_{max} взято меньше, чем есть на самом деле. Тогда напряжение, пропорциональное x, будет превышать 100 В и УПТ будет выходить из линейного режима. Обычно в усилителях предусматривается сигнализация, которая позволяет судить о том, что напряжение превысило допустимо возможное. Естественно, что при этом необходимо увеличить x_{max}, т. е. уменьшить масштаб и повторить решение задачи.

По формуле (12.22) определяется масштаб любой производной, в которой вместо x_{max} берется максимальное значение соответствующей производной.

В блоках нелинейности, используемых для моделирования зависимостей $y = f(x)$, переменная x моделируется напряжением e, а функция y изображается соответственно выходным напряжением $u_{вых}$.

Обычно график функции $y = f(x)$ известен (рис. 12.24). Напряжения e и $u_{вых}$ в функциональных преобразователях могут также, как в УПТ, изменяться в диапазоне ± 100 В. Поэтому масштабы определяются уравнениями:

$$M_x = \frac{100}{x_{max}}; \qquad M_y = \frac{100}{y_{max}},$$

где x_{max} и y_{max} — определяются видом нелинейной зависимости $y = f(x)$.

Блок перемножения двух переменных величин моделирует уравнение

$$y = ax_1 x_2, \tag{12.23}$$

которое трансформируется в следующее машинное уравнение:

$$u_{вых} = ce_1 e_2, \tag{12.24}$$

где c — постоянный для данной машины коэффициент.

Введем масштабы, связывающие переменные исходного уравнения (12.23) и машинные переменные (12.24)

$$M_y = u_y/y; \quad M_{x_1} = e_1/x_1; \quad M_{x_2} = e_2/x_2.$$

Определив из этих соотношений u_y, e_1, e_2 и подставив их значения в машинное уравнение, получим

$$y = \frac{M_{x_1} M_{x_2} c}{M_y} x_1 x_2. \qquad (12.25)$$

Выражение (12.25) тождественно моделируемому уравнению (12.23) при

$$a = \frac{M_{x_1} M_{x_2}}{M_y} c.$$

Откуда

$$M_y = \frac{c}{a} M_{x_1} M_{x_2},$$

т. е. масштаб произведения в машине вычисляется по известным масштабам сомножителей. Для большинства АВМ $c = 0{,}01$.

Время протекания процесса в модели отличается от времени протекания реального процесса и зависит от масштаба времени, определяемого по формуле

$$M_t = \tau / t, \qquad (12.26)$$

где τ — время протекания процесса в модели; t — время протекания реального процесса.

Если время протекания процесса в модели соответствует времени протекания реального процесса, т. е. $M_t = 1$, то такой масштаб времени называют *натуральным*. Этот масштаб времени используют в тех случаях, когда АВМ сопрягается с реальной аппаратурой.

В тех случаях, когда реальные процессы протекают очень быстро и их трудно наблюдать, можно использовать замедленный масштаб, т. е. $M_t > 1$ за счет увеличения τ.

И, наконец, если исследуемый процесс в действительности протекает очень медленно, то для его моделирования потребуется очень много времени. В таком случае можно ввести ускоренный масштаб времени $M_t < 1$.

Изменять масштаб времени можно за счет изменения постоянных времени интегрирующих блоков.

Расчет коэффициентов передачи. Определение напряжений начальных условий. Коэффициенты исходных математических зависимостей моделируются на АВМ коэффициентами передачи функциональных блоков. При этом последние связаны определенными математическими зависимостями с выбранными масштабами и коэффициентами исходных уравнений. Характер этих зависимостей определяется конкретным блоком и видом операции, которую он выполняет.

Найдем выражение для расчета коэффициентов передачи суммирующего блока. Для этого рассмотрим моделирование суммы n слагаемых:

$$y(t) = \sum_{j=1}^{n} a_j x_j(t).$$

Данное уравнение в суммирующем блоке трансформируется в машинное уравнение

$$u_{\text{вых}}(t) = -\sum_{j=1}^{n} K_j e_j(t).$$

Введем масштабы, которые связывают переменные исходного уравнения с машинными переменными:

$$M_y = u_{\text{вых}}(t)/y(t), \quad M_{x_j} = e_{x_j}(t)/x_j(t).$$

Определив из этих соотношений $u_{\text{вых}}(t)$, $e_{x_j}(t)$ и поставив их значения в машинное уравнение, получим

$$y(t) = -\frac{1}{M_y}\sum_{j=1}^{n} K_j M_{x_j} x_j(t).$$

Это уравнение тождественно исходному уравнению, если соответствующие коэффициенты этих уравнений равны между собой, т. е.

$$a_j = K_j M_{x_j}/M_y.$$

Знак минус в машинном уравнении на величину коэффициента передачи не влияет и поэтому он здесь не учитывается.

Следовательно,

$$K_j = a_j \frac{M_y}{M_{x_j}} = a_j \frac{M_{\text{вых}}}{M_{\text{вх}j}}. \qquad (12.27)$$

Из (12.27) видно, что коэффициент передачи по j-му входу равен отношению масштабов выходной величины к масштабу входной величины, умноженному на коэффициент, стоящий перед переменной исходного уравнения.

Таким образом, выбрав масштабы и зная коэффициенты уравнения, можно определить численное значение коэффициента передачи.

Коэффициент передачи по j-му входу суммирующего блока

$$K_j = \alpha \frac{R_0}{R_j}.$$

Путем изменения отношения сопротивлений или α $(0 \leqslant \alpha \leqslant 1)$ устанавливается значение коэффициента передачи, полученное расчетным путем.

Коэффициент передачи интегрирующего блока определяется следующим образом.

Интегрирующий блок реализует математическую операцию

$$y(t) = a\int_0^t x(t)\,dt.$$

которая моделируется уравнением

$$u_{\text{вых}}(\tau) = -\frac{1}{RC}\int_0^{\tau} e(\tau)\, d\tau.$$

Время τ протекания процесса в АВМ, как было показано выше, может отличаться от времени t протекания реального процесса. Введем масштабы, связывающие переменные исходного уравнения с машинными переменными:

$$M_y = u_{\text{вых}}(\tau)/y(t); \quad M_x = e(\tau)/x(t); \quad M_t = \tau/t.$$

Определив из этих соотношения $u_{\text{вых}}(\tau)$, $e(\tau)$, τ и подставляя их значения в уравнение интегрирующего блока, получим

$$y(t) = -\frac{1}{RC}\,\frac{M_x M_t}{M_y}\int_0^{t} x(t)\, dt.$$

Это уравнение тождественно моделирующему уравнению при равенстве соответствующих коэффициентов. Без учета знака минус в машинном уравнении

$$a = \frac{1}{RC}\,\frac{M_x M_t}{M_y}.$$

Откуда

$$K = \frac{M_y}{M_x M_t}\; a = \frac{1}{RC}; \tag{12.28}$$

Коэффициент передачи интегрирующего блока равен отношению масштаба переменной на его выходе, к произведению масштабов подынтегральной переменной и времени, умноженному на коэффициент a исходного уравнения.

Для интегрирующего блока, когда на его вход подается ряд переменных напряжений, коэффициент передачи для j-го входа

$$K_j = \frac{M_{\text{вых}}}{M_{\text{вх}j}\, M_t}\, a_j. \tag{12.29}$$

Если интегрирование производится в натуральном масштабе времени, т. е. $M_t = 1$, то

$$K_j = \frac{M_{\text{вых}}}{M_{\text{вх}j}}\, a_j.$$

Последнее выражение совпадает с выражением для коэффициента передачи суммирующего блока.

Если на один из входов суммирующего блока подается постоянная величина $b = \text{const}$, то ее масштаб

$$M_b = u_b/b = 100/b. \tag{12.30}$$

Тогда коэффициент передачи по этому входу

$$K_b = \frac{M_{\text{вых}}}{M_{\text{вх}}} a = \frac{M_{\text{вых}}}{100} b, \tag{12.31}$$

так как $a = 1$.

Для решения дифференциальных уравнений на АВМ в качестве исходных данных необходимо иметь значения переменных, входящих в состав этих уравнений, в начальный момент времени. Эти значения называют *начальными условиями*.

Напряжения, соответствующие начальным условиям, определяют исходя из выбранных масштабов переменных по формуле

$$u_{x_0} = \text{Sign}\,(x)\, M_x\, x_0, \tag{12.32}$$

где x_0 — начальное значение переменной x при $t = 0$; M_x — масштаб переменной x:

$$\text{Sign}\,(x) = \begin{cases} +1 \text{ при } x > 0; \\ -1 \text{ при } x < 0. \end{cases}$$

Функция sign (x) учитывает знак машинной переменной на выходе интегрирующего блока. Если на выходе блока получается отрицательная переменная, то в формуле вместо sign (x) ставится минус, если на выходе получается положительная переменная, то вместо sign (x) ставится плюс.

Полученное расчетным путем напряжение начальных условий устанавливается путем зарядки интегрирующего конденсатора до напряжения u_{x0} на выходе интегрирующего блока.

Одна из схем, с помощью которой можно осуществлять установку напряжений начальных условий, приведена на рис. 12.11.

§ 12.5. ПРИМЕРЫ ПОДГОТОВКИ ЗАДАЧ К РЕШЕНИЮ И ИССЛЕДОВАНИЮ НА АВМ

Составление структурной схемы для решения обыкновенного дифференциального уравнения с постоянными коэффициентами.

Пример. Задано дифференциальное уравнение второго порядка с постоянными коэффициентами

$$2\frac{d^2x}{dt^2} + 3\frac{dx}{dt} + 4x = 0$$

и начальные значения переменных при $t = 0$, $x_0' = 2$; $x_0 = 5$.

Требуется подготовить это уравнение для решения на АВМ.

1. Предположим, что решение уравнения должно происходить в натуральном масштабе времени, т. е. $M_t = 1$. Зададимся следующими максимальными значениями переменных $x_{\max} = 5$; $x'_{\max} = 20$; и определим их масштабы

$$M_x = u_{x\,\max}/x_{\max} = 100/5 = 20;$$

$$M_{x'} = u_{x'\max}/x'_{\max} = 100/20 = 5;$$

$$M_t = \tau/t = 1.$$

2. Составим структурную схему задачи. Для этого исходное уравнение разрешим относительно старшей производной

$$\frac{d^2x}{dt^2} = -1{,}5\frac{dx}{dt} - 2x.$$

Чтобы получить напряжение, пропорциональное второй производной, необходимо сложить напряжения, соответствующие членам правой части уравнения, и затем с помощью интегрирующих блоков понизить порядок производной. При составлении структурной схемы следует учитывать, что с помощью интегросуммирующего блока можно выполнять операцию интегрирования суммы входных напряжений.

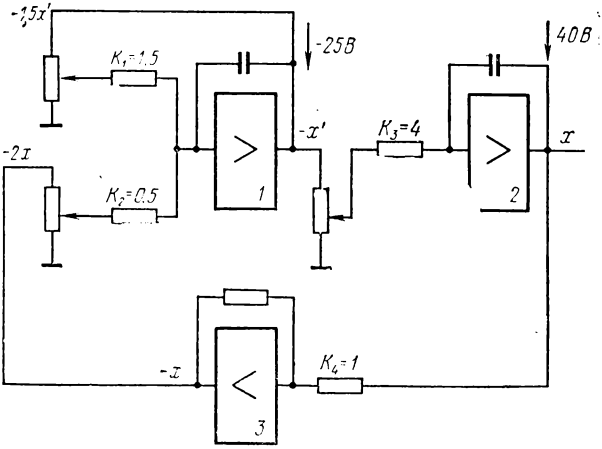


Рис. 12.25. Структурная схема для решения дифференциальных уровнений второго порядка

Если на вход интегросуммирующего блока *1* (рис. 12.25) подать напряжения, пропорциональные членам — 1,5 dx/dt и —2 x, то на выходе получится величина, пропорциональная $-x'$ (знак минус перед ней говорит о том, что каждый блок изменяет знак входного напряжения). Дальнейшее интегрирование первой производной с помощью интегрирующего блока *2* позволяет получить на его выходе величину, пропорциональную x, которая и является искомой величиной. На вход интегросуммирующего блока должны быть поданы величины $-x'$ и x. На выходе блока *1* получается величина $-x'$ и поэтому ее сразу же можно подать на верхний вход блока *1*. На выходе блока 2 x получается со знаком плюс, а на нижний вход блока *1* требуется подать $-x$. Поэтому с помощью инвертора *3* изменяем знак переменной $-x$ и подаем эту величину на нижний вход блока *1*. На этом составление структурной схемы заканчивается.

3. Рассчитаем коэффициенты передачи. Для этого определим значения коэффициентов передачи по каждому входу всех блоков в структурной схеме:

$$K_1 = \frac{M_{x'}}{M_{x'}M_t}a_1 = \frac{5}{5\cdot 1}1{,}5 = 1{,}5;$$

$$K_2 = \frac{M_{x'}}{M_x M_t}a_2 = \frac{5}{20\cdot 1}2 = 0{,}5;$$

$$K_3 = \frac{M_x}{M_{x'}M_t}a_3 = \frac{20}{5\cdot 1}1 = 4;$$

$$K_4 = 1.$$

При расчете K_3 величина $a_3 = 1$, так как этот блок выполняет только операцию интегрирования.

На структурной схеме значения коэффициентов передачи отмечаются над входными резисторами решающих блоков.

4. Определим напряжение начальных условий по формуле (12.32):

$$u_{x_0} = \text{Sign}\,(x)\, x_0\, M_x = +2 \times 20 = +40;$$

$$u_{x_0'} = \text{Sign}\,(x')\, x_0'\, M_{x'} = -5 \times 5 = -25.$$

Напряжение начальных условий $u_{x_0'}$ получается со знаком минус, так как на выходе первого решающего блока будет величина $-x'$ и вместо Sign (x') подставляется -1.

На структурной схеме начальные условия обозначаются стрелкой на выходе соответствующего блока; рядом со стрелкой указывается напряжение начальных условий.

5. Выбираем АВМ, на которой будет решаться задача.

На АВМ производим набор структурной схемы, установку коэффициентов передачи и напряжений начальных условий.

Исследование задачи сводится к подбору коэффициентов, при которых искомая переменная находится в нужных пределах. После этого экспериментально на модели измеряем все коэффициенты передачи и по формулам (12.27), (12.29) вычисляем искомые значения коэффициентов a_j заданного уравнения.

Составление структурной схемы для решения системы дифференциальных уравнений

Пример. Задана система дифференциальных уравнений

$$\frac{d^2x}{dt^2} + 5y - 2x = 3;$$

$$\frac{d^2y}{dt^2} - 5x - 2y = 5$$

и начальные значения переменных $x_0 = 5$; $x_0' = 2{,}5$; $y_0 = 1$; $y_0' = 4$.

Требуется подготовить систему для решения на АВМ.

1. Предположим, что решение системы должно происходить в замедленном масштабе времени, т. е.

$$M_t = \tau/t = 2.$$

Зададимся максимальными значениями переменных $x_{\max} = 5$; $x'_{\max} = 5$; $y_{\max} = 10$; $y'_{\max} = 25$ и определим масштабы переменных:

$$M_x = u_{x\,\max}/x_{\max} = 100/5 = 20;$$

$$M_y = u_{y\,\max}/y_{\max} = 100/10 = 10;$$

$$M_{x'} = u_{x'\max}/x'_{\max} = 100/5 = 20;$$

$$M_{y'} = u_{y'\,\max}/y'_{\max} = 100/25 = 4.$$

2. Составим структурную схему задачи. Для этого каждое из уравнений разрешим относительно старшей производной:

$$\frac{d^2x}{dt^2} = -5y + 2x + 3;$$

$$\frac{d^2y}{dt^2} = 5x + 2y + 5.$$

Структурные схемы (рис. 12.26) составим для каждого уравнения отдельно, а затем проведем связи между этими схемами в соответствии с решаемой системой.

В правой части первого и второго уравнений стоит по три члена. Поэтому необходимо использовать два интегросуммирующих блока на три входа. На вход интегросуммирующего блока *1* необходимо подавать y со знаком минус. На выходе блока *4* переменная y получается со знаком плюс. Поэтому для изменения знака введен инвертор *5*. На входы блоков *1* и *3* должны подаваться постоянные напряжения, так как в каждом из уравнений присутствует постоянный член.

3. Определим коэффициенты передачи усилителей по формулам (12.27), (12.29) и (12.31):

$$K_1 = \frac{M_{x'}}{100 M_t} b_1 = \frac{20}{100 \cdot 2} 3 = 0{,}3;$$

$$K_2 = \frac{M_{x'}}{M_y M_t} a_1 = \frac{20}{10 \cdot 2} 5 = 5;$$

$$K_3 = \frac{M_{x'}}{M_x M_t} a_2 = \frac{20}{20 \cdot 2} 2 = 1;$$

$$K_4 = \frac{M_x}{M_{x'} M_t} a = \frac{20}{20 \cdot 2} 1 = 0{,}5;$$

$$K_5 = \frac{M_{y'}}{M_x M_t} a_3 = \frac{4}{20 \cdot 2} 5 = 0{,}5;$$

$$K_6 = \frac{M_{y'}}{100 M_t} b_2 = \frac{4}{100 \cdot 2} 5 = 0{,}1;$$

$$K_7 = \frac{M_{y'}}{M_y M_t} a_4 = \frac{4}{10 \cdot 2} 2 = 0{,}4;$$

$$K_8 = \frac{M_y}{M_{y'} M_t} a = \frac{10}{4 \cdot 2} 1 = 1{,}25;$$

$$K_9 = 1.$$

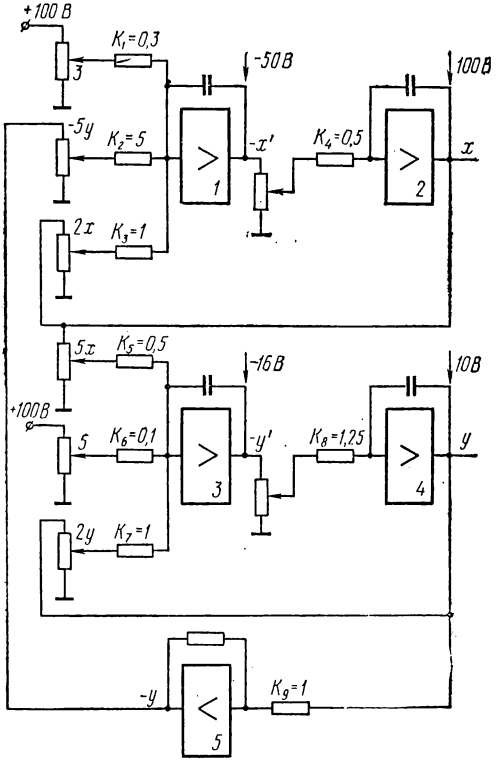


Рис. 12.26. Структурная схема для решения системы дифференциальных уравнений

4. Напряжения начальных условий определим по формуле (12.32):

$$u_{x_0} = \mathrm{Sign}\,(x)\, x_0 M_x = +20 \cdot 5 = 100;$$

$$u_{x_0'} = \mathrm{Sign}\,(x')\, x_0' M_{x'} = -20 \cdot 2{,}5 = -50;$$

$$u_{y_0} = \mathrm{Sign}\,(y)\, y_0 M_y = +10 \cdot 1 = 10;$$

$$u_{y_0'} = \mathrm{Sign}\,(y')\, y_0' M_{y'} = -4 \cdot 4 = -16.$$

5. Выбираем АВМ, на которой будет решаться система.

Затем определяем операционные усилители и резисторы в АВМ и их номера указываем на структурной схеме, которую набираем на машине. В этом примере решение получается на выходах блоков *2* и *4*.

Подготовка нелинейного дифференциального уравнения к решению на АВМ. Пример. Задано нелинейное дифференциальное уравнение

$$2\frac{d^3x}{dt^3} + 3\frac{d^2x}{dt^2} + f(x)\frac{dx}{dt} - 2x = 8$$

и начальные значения переменных $x_0'' = 40$; $x_0' = 10$; $x_0 = -50$. Нелинейный коэффициент $f(x)$ задан графически (рис. 12.27).

Требуется подготовить уравнение для исследования на АВМ в натуральном масштабе времени, т. е. $M_t = 1$.

1. Зададимся следующими максимальными значениями переменных $x''_{max} = 40$; $x'_{max} = 25$, $x_{max} = 50$ и определим масштабы переменных:

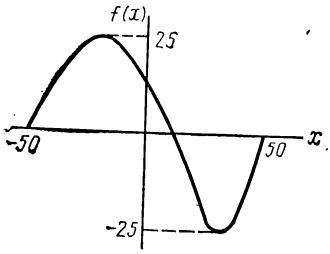


Рис. 12.27. Вид нелинейной функции

$$M_t = \tau/t = 1;$$

$$M_x = u_{x\,max}/x_{max} = 100/50 = 2;$$

$$M_{x'} = u_{x'\,max}/x'_{max} = 100/25 = 4;$$

$$M_{x''} = u_{x''\,max}/x''_{max} = 100/40 = 2{,}5;$$

$$M_{пр} = 0{,}01\,\frac{M_{1\,сомн}\,M_{2\,сомн}}{a} = 0{,}01\,\frac{4\cdot 4}{1} = 0{,}16;$$

$$M_{f(x)} = u_{f\,max}/f_{max} = 100/25 = 4.$$

2. Составим структурную схему для решения задачи. Для этого исходное уравнение запишем относительно старшей производной:

$$\frac{d^3x}{dt^3} = -1{,}5\,\frac{d^2x}{dt^2} - 0{,}5f(x)\,\frac{dx}{dt} + x + 4.$$

Порядок составления структурной схемы для нелинейных дифференциальных уравнений (рис. 12.28) аналогичен порядку составления структурной схемы решения линейного уравнения.

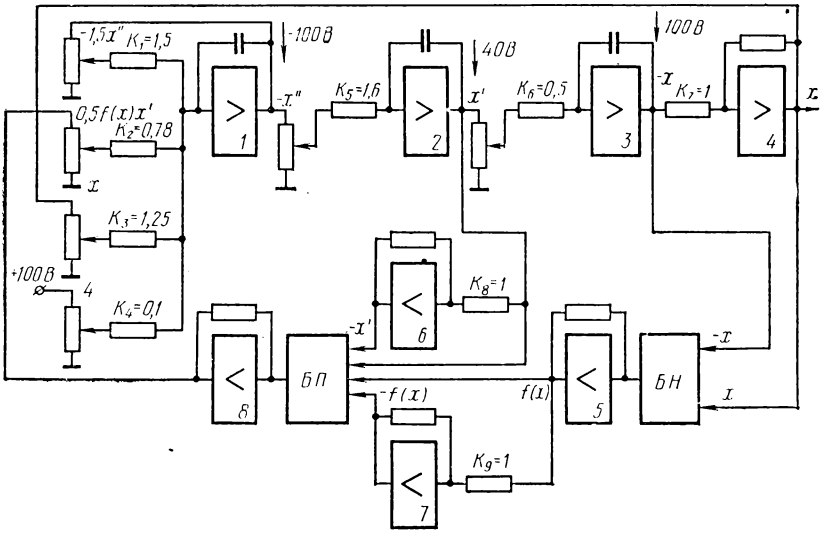


Рис. 12.28. Структурная схема для решения нелинейного дифференциального уравнения

Подаем величины правой части уравнения на вход интегросуммирующего блока *1* и с помощью интегрирующих блоков *2* и *3* последовательно понижаем порядок производной. На выходе блока *1* получается величина $-x''$, на выходе блока *2* — величина $+x'$, на выходе блока *3* — величина $-x$. Так как решается

уравнение третьего порядка, то для получения искомой величины x требуется три интегрирующих блока. Теперь с помощью дополнительных блоков получим все величины правой части уравнения и подадим их на входы интегросуммирующего блока *1*. Величина $-x''$, получаемая на выходе блока *1*, непосредственно подается на вход этого же блока. На вход интегросуммирующего блока *1* должна подаваться величина $+x$. Для этого с помощью инвертора *4* производится перемена знака, и, наконец, на вход блока *1* подается постоянное напряжение, которое моделирует постоянный член уравнения.

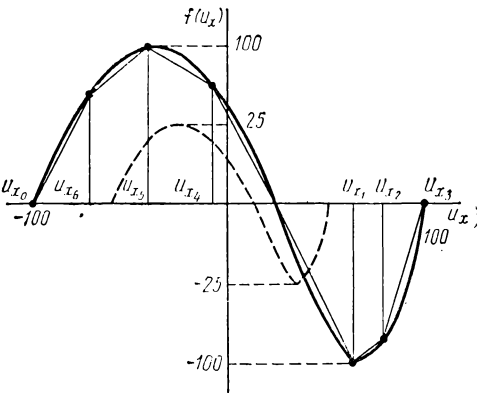


Рис. 12.29. Подготовка функции к набору на нелинейном блоке

Член уравнения $-0{,}5\, f(x)\, x'$ получается следующим образом. Для набора нелинейной зависимости используется нелинейный блок БН. Заданная нелинейная зависимость (см. рис. 12.27) аппроксимируется конечным числом линейных отрезков, которые могут лежать во всех четырех квадрантах. Поэтому на вход нелинейного блока подается x со знаком плюс и минус. С выхода усилителя *5* снимается функция $f(x)$. Операция перемножения $f(x)$ и x' выполняется с помощью блока перемножения БП, на вход которого сомножители должны подаваться со своим и обратным знаками. Поэтому $f(x)$ инвертируется с помощью инвертора *7*, а x' — с помощью инвертора *6*. На выходе усилителя *8* получается требуемое произведение.

3. Рассчитаем коэффициенты передачи

$$K_1 = \frac{M_{x''}}{M_{x''}}\ a_1 = \frac{2{,}5\cdot 1{,}5}{2{,}5} = 1{,}5;\quad K_2 = \frac{M_{x''}}{M_{\text{пр}}}\ a_2 = \frac{2{,}5\cdot 0{,}5}{0{,}16} = 0{,}78;$$

$$K_3 = \frac{M_{x''}}{M_x}\ a_3 = \frac{2{,}5\cdot 1}{2} = 1{,}25;\quad K_4 = \frac{M_{x''}}{100}\ b = \frac{2{,}5\cdot 4}{100} = 0{,}1;$$

$$K_5 = \frac{M_{x'}}{M_{x''}}\ a = \frac{4\cdot 1}{2{,}5} = 1{,}6;\quad K_6 = \frac{M_x}{M_{x'}}\ a = \frac{2\cdot 1}{4} = 0{,}5;$$

$$K_7 = K_8 = K_9 = 1.$$

При определении масштаба произведения коэффициент a был взят равным 1, а коэффициент 0,5, который стоит в уравнении перед произведением, учтен при расчете коэффициента передачи K_2.

4. Определим напряжения начальных условий:

$$u_{x_0} = \text{Sign}\,(x)\, x_0\, M_x = -(-50)\cdot 2 = 100;$$

$$u_{x'_0} = \text{Sign}\,(x')\, x'_0\, M_{x'} = +10\cdot 4 = 40;$$

$$u_{x''_0} = \text{Sign}\,(x'')\, x''_0\, M_{x''} = -40\cdot 2{,}5 = -100.$$

Напряжение u_{x_0} получается положительным, так как на выходе интегрирующего блока x знак минус и начальное значение x задано отрицательным.

5. Исходная нелинейная зависимость $f(x)$ для набора на блоке нелинейностей должна быть трансформирована в координаты, выраженные через напряжения. При этом обычно необходимо стремиться к тому, чтобы напряжения по обеим осям получались в машине максимально возможными, т. е. по 100 В (рис. 12.29). Затем график аппроксимируется с учетом заданной точности конечным числом линейных отрезков.

6. Выбираем АВМ, на которой будет решаться уравнение. В машине определим усилители, линейные и нелинейные элементы и их номера проставим на структурной схеме. Затем схему наберем на АВМ, установим коэффициенты передачи и напряжений начальных условий.

На нелинейном блоке наберем нелинейную зависимость и проведем решение и исследование уравнения.

§ 12.6. НЕКОТОРЫЕ ОСОБЕННОСТИ ПОДГОТОВКИ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ ДЛЯ РЕШЕНИЯ НА АВМ

Выравнивание переменных и приведение коэффициентов уравнений к одному порядку. Обычно коэффициенты передачи решающих блоков должны лежать в определенных для данного типа машины пределах, т. е. они не должны превышать K_{max} и должны быть не меньше K_{min} Выход коэффициента передачи за эти значения приводит к ошибкам при решении задачи. АВМ строятся на усилителях постоянного тока, которые допускают использовать решающие блоки с максимальным значением коэффициента передачи (от 10 до 50). Конкретное значение коэффициента передачи зависит от особенностей построения усилителей и методов, которые применяются для уменьшения их погрешностей. Значение K_{min} обусловливается тем, что коэффициенты передачи устанавливаются с помощью потенциометров, которые обладают определенной разрешающей способностью.

Численное значение коэффициента передачи блока, как было показано выше, определяется коэффициентами исходного уравнения и масштабами переменных. При подготовке уравнений к исследованию на АВМ могут встречаться случаи, когда при выбранных максимальных значениях переменных коэффициенты передачи будут располагаться за пределами допустимых величин. Кроме того, коэффициенты исходного уравнения могут изменяться в широком диапазоне, а это приводит к тому, что коэффициенты передачи блоков будут существенно отличаться друг от друга. Коэффициентами, имеющими малые значения в структурной схеме, можно пренебречь. Но часто эти связи необходимо сохранить. Это можно сделать за счет изменения масштабов переменных и времени.

Пример. Рассмотрим изменения масштабов переменных в системе дифференциальных уравнений третьего порядка:

$$\begin{aligned}
\frac{dx_1}{dt} &= a_{11}x_1 + a_{12}x_2 + a_{13}x_3;\\
\frac{dx_2}{dt} &= a_{21}x_1 + a_{22}x_2 + a_{23}x_3;\\
\frac{dx_3}{dt} &= a_{31}x_1 + a_{32}x_2 + a_{33}x_3.
\end{aligned} \qquad (12.33)$$

Для этой системы начальные условия будут x_{10}, x_{20}, x_{30}.

Предположим, что выбраны максимальные значения переменных и определены масштабы. При этом напряжения, моделирующие одни переменные, будут изменяться в очень маленьком диапазоне, а напряжения, моделирующие другие переменные, будут превышать максимально допустимые напряжения. Это

свидетельствует о том, что диапазоны изменения переменных системы различны и масштабы выбраны неверно.

Требуется изменить масштабы переменных таким образом, чтобы при решении задачи на машине напряжения, пропорциональные переменным, изменялись бы в одинаковых пределах.

Для этого вводим новые переменные:

$$x_1 = \gamma_1 x_1^*; \quad x_2 = \gamma_2 x_2^*; \quad x_3 = \gamma_3 x_3^*. \tag{12.34}$$

При подстановке новых переменных в исходную систему получаем:

$$\left.\begin{aligned} \frac{dx_1^*}{dt} &= a_{11} x_1^* + a_{12} \frac{\gamma_2}{\gamma_1} x_2^* + a_{13} \frac{\gamma_3}{\gamma_1} x_3^*; \\ \frac{dx_2^*}{dt} &= a_{21} \frac{\gamma_1}{\gamma_2} x_1^* + a_{22} x_2^* + a_{23} \frac{\gamma_3}{\gamma_2} x_3^*; \\ \frac{dx_3^*}{dt} &= a_{31} \frac{\gamma_1}{\gamma_3} x_1^* + a_{32} \frac{\gamma_2}{\gamma_3} x_2^* + a_{33} x_3^*. \end{aligned}\right\} \tag{12.35}$$

При подстановке также изменяются величины начальных условий:

$$x_{10}^* = x_{10}/\gamma_1; \quad x_{20}^* = x_{20}/\gamma_2; \quad x_{30}^* = x_{30}/\gamma_3.$$

Варьируя величины коэффициентов γ_1, γ_2, γ_3, можно добиться изменения переменных в одинаковых пределах. После решения преобразованной системы в машине для определения истинных значений x_1, x_2 и x_3 необходимо учитывать значения γ и определять их по формулам (12.34).

Пример. Рассмотрим случай, когда в исходном уравнении коэффициенты изменяются в широких пределах. Предположим, что на АВМ требуется решить уравнение

$$1000 \frac{dx^3}{dt^3} + 500 \frac{d^2x}{dt^2} - 25 \frac{dx}{dt} + 5x = 1.$$

Начальные значения переменных x_0, x_0', x_0''.

Допустим, что для решения этого уравнения выбрана машина, в которой на решающем блоке можно устанавливать $K_{\min} = 0{,}1$ и $K_{\max} = 10$. В соответствии с методикой подготовки задачи к решению на АВМ необходимо задаться максимальными значениями переменных и определить масштабы. Пусть максимальные значения переменных $x''_{\max} = x'_{\max} = x_{\max} = 100$,

тогда масштабы переменных

$$M_{x''} = M_{x'} = M_x = 1,$$

а масштаб времени

$$M_t = 1.$$

Разрешим уравнение относительно старшей производной

$$\frac{dx^3}{dt^3} = -0{,}5 \frac{d^2x}{dt^2} + 0{,}025 \frac{dx}{dt} - 0{,}005x + 0{,}001$$

и составим структурную схему (рис. 12.30) для решения этого уравнения.

Определим коэффициенты передачи блоков при выбранных максимальных значениях переменных;

$$K_1 = \frac{M_{x''}}{M_{x''} M_t} a_1 = 0{,}5; \quad K_2 = \frac{M_{x''}}{M_{x'} M_t} a_2 = 0{,}025;$$

$$K_3 = \frac{M_{x''}}{M_x M_t} a_3 = 0{,}005; \quad K_4 = \frac{M_{x''}}{100 \cdot M_t} b = 0{,}00001;$$

$$K_5 = K_6 = K_7 = 1.$$

Видно, что K_1 отличается на порядок от K_2 и на два порядка от K_3. Кроме того, K_2, K_3 и K_4 получились меньше допустимого минимального значения коэффициента передачи 0,1 для выбранной машины. Для того чтобы коэффициенты передачи мало отличались один от другого и лежали в заданных пределах, необходимо выравнивать порядки коэффициентов исходного уравнения. Это осуществляется за счет изменения масштаба времени.

Введем $M_t = 0$, 1, т. е. $t = 10\,\tau$ и подставим значение t в исходное уравнение:

$$\frac{d^3x}{d\tau^3} + 5\frac{d^2x}{d\tau^2} - 2{,}5\frac{dx}{d\tau} + 5x = 1.$$

После такого преобразования коэффициенты передачи решающих блоков при тех же масштабах переменных, но при новом значении масштаба времени бу-

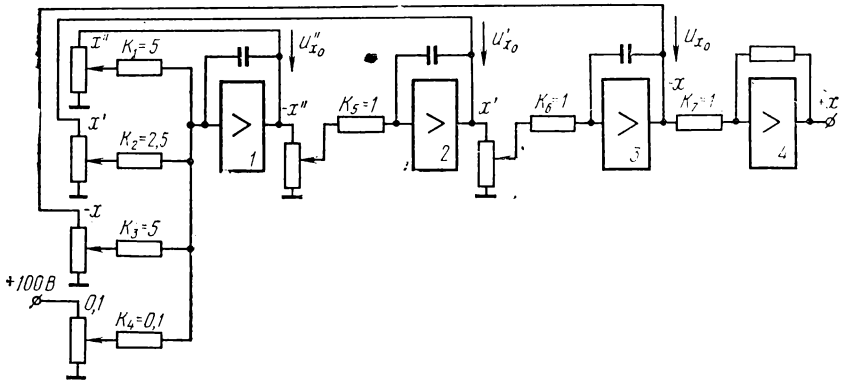


Рис. 12.30. Структурная схема для решения дифференциального уравнения третьего порядка

дут: $K_1 = 5$; $K_2 = 2{,}5$; $K_3 = 5$; $K_4 = 0{,}1$; $K_5 = K_6 = K_7 = 1$, т. е. имеют одинаковый порядок и лежат в заданных пределах. Установить такие коэффициенты передачи на решающих блоках несложно.

Выравнивание коэффициентов передачи за счет изменения масштаба времени возможно в том случае, когда порядок коэффициентов исходного уравнения возрастает (или уменьшается) с ростом порядка производной. При уменьшении значения коэффициента с ростом порядка производной M_t выбирается больше единицы.

Решение дифференциальных уравнений с переменной правой частью. Предположим, что задано дифференциальное уравнение n-го порядка с переменной правой частью:

$$\frac{d^n x}{dt^n} + a_{n-1}\frac{d^{n-1}x}{dt^{n-1}} + \ldots + a_1 x = y.$$

В этом уравнении y может быть функцией времени или аргумента, т. е. $y = f(x)$; $y = f(t)$, где y — непрерывная однозначная функция.

Для воспроизведения произвольных нелинейных зависимостей обычно используют нелинейные функциональные блоки (см. рис. 12.18).

Примеры составления структурных схем с использованием нелинейных функциональных блоков для набора функций $f(x)$ были рас-

смотрены ранее. Рассмотрим теперь воспроизведение функций вида $y = f(t)$.

Пример. Получить функцию вида $y = f(t)$ можно с помощью схемы, изображенной на рис. 12.31, которая состоит из интегрирующего блока *1*, инвертора *2* и нелинейного блока БН с суммирующим усилителем *3*. Заданная нелинейная зависимость набирается на нелинейном блоке БН, на вход которого должны по-

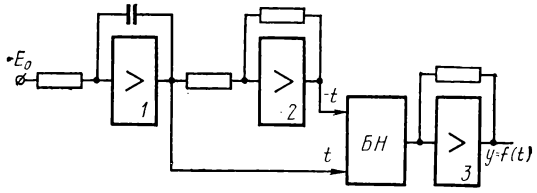


Рис. 12.31. Схема для воспроизведения зависимости $y=f(t)$

ступать переменные напряжения, пропорциональные времени. Это достигается следующим образом. На вход интегрирующего блока *1* подается постоянное напряжение $-E_0$, при этом блок выполняет следующую операцию:

$$u = -\frac{1}{RC}\int_0^t (-E_0)\,dt = \frac{E_0}{RC}\,t.$$

Если $E_0/RC = 1$, то $u = t$, т. е. напряжение на выходе интегрирующего блока *1* имитирует время (рис. 12.32). На выходе инвертора получается $u = -t$. Напряжения, пропорциональные $\pm t$, поступают на вход нелинейного блока БН и на его выходе будет $y = f(t)$.

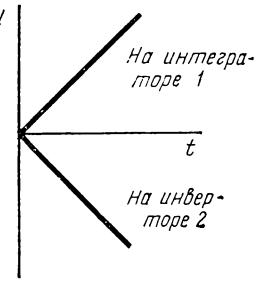


Рис. 12.32. График изменения напряжения на интеграторе и инверторе

Для набора функций $f(x)$ и $f(t)$ используются нелинейные блоки. В ряде случаев для набора зависимостей определенного вида, которые рассматриваются ниже, можно избежать применения нелинейных блоков и применять структурные схемы, моделирующие эти зависимости.

Пример. Предположим, что правая часть дифференциального уравнения y задана полиномом второй степени по аргументу t:

$$y = b_0 + b_1 t + b_2 t^2,$$

где b_j — постоянный коэффициент, принимающий как положительные, так и отрицательные значения.

Реализация этого полинома производится с помощью схемы, изображенной на рис. 12.33. На вход интегрирующего блока *1* подается постоянное напряжение E_0, интегрирование которого позволяет получить линейный член

$$u(t) = -\frac{1}{R_1 C_1}\int_0^t E_0\,dt = -\frac{1}{R_1 C_1}\,E_0 t.$$

Это напряжение инвертируется блоком *3* и подается на вход суммирующего блока *4* и на вход интегрирующего блока *2*. Интегрирование линейного члена

позволяет получить составляющую второго порядка:

$$u(t)=\frac{1}{R_2C_2}\int_0^t\frac{1}{R_1C_1}E_0\,tdt=\frac{E_0}{R_1R_2C_1C_2}\,\frac{t^2}{2},$$

которая также подается на вход суммирующего блока.

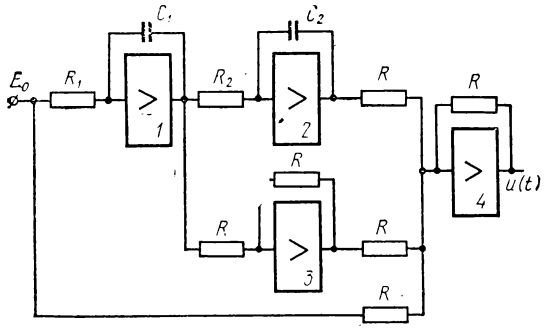


Рис. 12.33. Структурная схема формирования полинома

Напряжение на выходе суммирующего блока описывается уравнением:

$$u(t)=-\left(E_0+\frac{1}{R_1C_1}\,t+\frac{1}{2R_1R_2C_1C_2}\,t^2\right).$$

Если в схему добавить еще один интегратор, то можно получить полином третьей степени, и т. д.

Пример. Рассмотрим схему для получения зависимости $y=e^{+\alpha t}$.

Чтобы составить структурную схему, воспроизводящую заданную зависимость, продифференцируем правую и левую части уравнения:

$$y'=\alpha e^{\alpha t}=\alpha y.$$

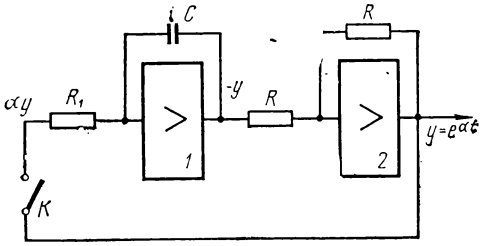


Рис. 12.34. Схема для формирования зависимости $y=e^{\alpha t}$

Структурная схема, моделирующая эту зависимость, показана на рис. 12.34. При замыкании ключа K на выходе инвертора *2* получается напряжение, пропорциональное $y=e^{+\alpha t}$. Аналогично можно получить зависимость $y=e^{-\alpha t}$. При этом в схеме (рис. 12.34) будет отсутствовать инвертирующий блок и напряжение с выхода интегратора *1* через ключ K подается на его вход.

Если правая часть уравнения задана в виде $y=e^{-x}$, то ее можно воспроизвести с помощью специальной структурной схемы.

Дифференцирование правой и левой части равенства приводит к уравнению

$$y'=-\frac{dx}{dt}e^{-x}=-\frac{dx}{dt}y.$$

Схема, моделирующая эту зависимость, показана на рис. 12.35. Она состоит из блока перемножения БП, интегрирующего блока *1* и инвертора *2*. Величины

x' и $-x'$ берутся из структурной схемы решения левой части исследуемого уравнения.

Пример. Получить функции $y_1 = \sin(x)$ и $y_2 = \cos(x)$ можно с помощью одной структурной схемы. Продифференцировав левые и правые части этих выражений, получим

$$y_1' = x' \cos(x); \quad y_2' = x' \sin(x).$$

Схема, позволяющая получить такие зависимости, показана на рис. 12.36.

Блок перемножения $БП_1$ осуществляет перемножение двух переменных величин x' и $y_2 = \cos(x)$. Интегрирование произведения усилителем *2* дает на выходе переменную y_1. Так как $y_1 = \sin(x)$, то его перемножение на x' с помощью блока $БП_2$ и последующее интегрирование с помощью усилителя *5* позволяет получить переменную $y_2 = \cos(x)$.

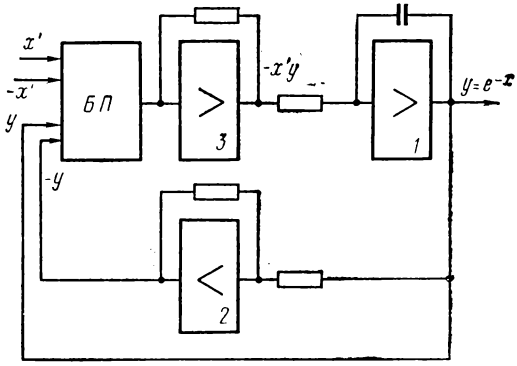


Рис. 12.35. Структурная схема для моделирования зависимости $y = e^{-x}$

Рассмотрим случай, когда правая часть дифференциального уравнения содержит переменное возмущение и его производные. Для моделирования правых частей при этом так же могут быть использованы специальные структурные схемы.

Пусть задано уравнение

$$\frac{d^2x}{dt^2} + a_1 \frac{dx}{dt} + a_2 x = b_1 \frac{dy}{dt} + b_2 y,$$

при начальных условиях x_0' и x_0.

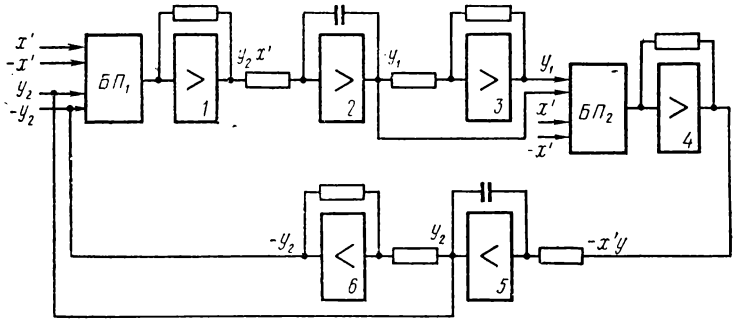


Рис. 12.36. Структурная схема для моделирования зависимостей $y_1 = \sin(x)$ и $y_2 = \cos(x)$

Для составления структурной схемы исходное уравнение разрешим относительно старшей производной:

$$\frac{d^2x}{dt^2} = b_1 \frac{dy}{dt} + b_2 y - a_1 \frac{dx}{dt} - a_2 x.$$

Чтобы сформировать член $b_1 dy/dt$, необходимо использовать дифференцирующий усилитель.

Структурная схема для формирования правой части уравнения имеет вид, показанный на рис. 12.37. Поскольку дифференцирующий усилитель обладает

малой помехозащищенностью и его применение может привести к большим погрешностям при решении задачи, то желательно составить такую структурную схему так, чтобы избежать его применения.

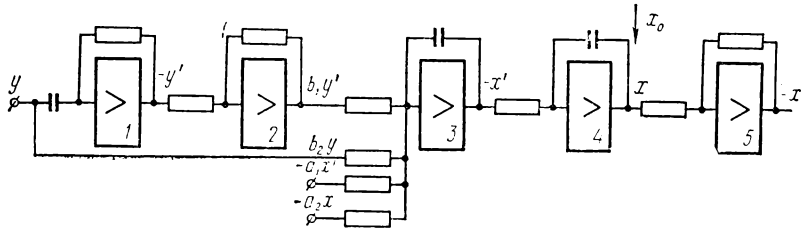


Рис. 12.37. Структурная схема с дифференцирующим ОУПТ для уравнения с производной в правой части уравнения

Один из методов составления такой структурной схемы заключается в том, что исходное уравнение переписывается относительно старших производных как переменной x, так и переменной y:

$$\frac{d^2x}{dt^2} - b_1\frac{dy}{dt} = b_2y - a_1\frac{dx}{dt} - a_2x.$$

Для этого уравнения структурная схема (рис. 12.38) составляется следующим образом. Величины, пропорциональные членам в правой части уравнения, суммируются и одновременно интегрируются усилителем *1*. На его выходе получается сумма членов левой части уравнения:

$$-\frac{dx}{dt} + b_1y.$$

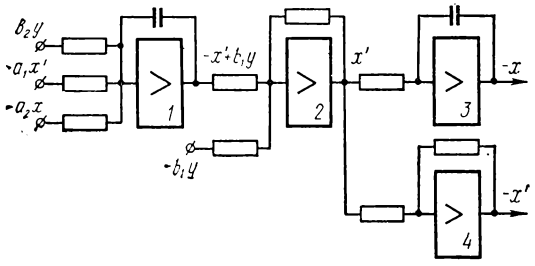


Рис. 12.38. Структурная схема уравнения с производной в правой части с суммированием воздействия

Для выделения x' на вход суммирующего усилителя *2* подается член уравнения $-b_1y$. Затем сигнал с выхода усилителя *2* интегрируется усилителем *3*.

Иногда от операции дифференцирования избавиться невозможно. В таких случаях могут быть использованы приближенные методы дифференцирования, основанные не реализации различных математических зависимостей. Возьмем уравнение вида

$$y = -x - \int_0^t y\,dt + Ky$$

и продифференцируем его. В результате это уравнение можно записать следующим образом:

$$\frac{dy}{dt}(1-K) + y = -\frac{dx}{dt}.$$

Если $K = 1$, то

$$y = -dx/dt.$$

С помощью схемы (рис. 12.39), воспроизводящей данное уравнение, можно выполнять операцию дифференцирования. Эта схема состоит из суммирующего усилителя *1*, инвертора *2* и интегратора *3*. На вход схемы подается переменная *x*.

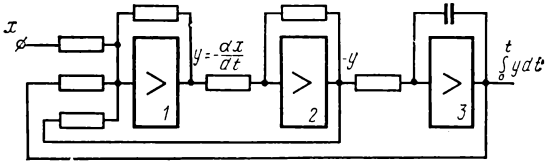


Рис. 12.39. Схема для реализации операции дифференцирования

Выходное напряжение снимается с усилителя *1*. При реализации этой схемы необходимо учитывать, что она имеет контур с положительной обратной связью, состоящей из инвертирующего и суммирующего усилителей. Когда $K = 1$, то схема возбуждается.

§ 12.7. КРАТКИЕ СВЕДЕНИЯ ОБ ОТЕЧЕСТВЕННЫХ АВМ

Отечественной промышленностью серийно выпускаются АВМ различных типов, которые условно можно разделить на машины общего назначения и специализированные. АВМ находят широкое применение при исследовании динамических процессов в различных объектах, системах автоматического регулирования и в управлении производственными процессами.

Наиболее важными характеристиками АВМ являются: вид решаемых дифференциальных уравнений, их максимальный порядок, допустимая длительность интегрирования и точность решения.

Современные АВМ снабжаются специальными устройствами, позволяющими многократно повторять процесс решения. При повторном решении можно изменять начальные условия, что дает возможность за сравнительно короткое время просмотреть большое количество вариантов и выбрать среди них наилучший.

Некоторые АВМ снабжены средствами для оптимизации, что позволяют им автоматически находить для исследуемой системы такое сочетание параметров, при котором величина, зависимая от этих параметров, принимает минимальное или максимальное значение.

Специализированные машины обеспечивают решение либо одной задачи, либо определенного класса задач.

АВМ, которые решают только одну задачу, как правило, используют в управлении производственными процессами. Например, существуют управляющие АВМ, которые рассчитывают дозировку шихты ферросплавной печи, управляют процессами доменной печи. В последнее время управляющие АВМ находят все более широкое применение в самых разнообразных производственных процессах. Растет и парк этих машин.

Машины общего назначения направлены на решение целого класса

задач. Среди них можно выделить машины, предназначенные для решения линейных и нелинейных дифференциальных уравнений, а также машины для решения дифференциальных уравнений в частных производных.

Рассмотрим характеристики некоторых АВМ, предназначенных для решения линейных и нелинейных дифференциальных уравнений, которые нашли наибольшее распространение и являются типичными представителями машин этого класса.

АВМ, предназначенные для решения линейных дифференциальных уравнений. Для решения линейных дифференциальных уравнений используют, например, моделирующие установки типа ИПТ-5 (интегратор постоянного тока), МПТ-9 и др.

Электронная моделирующая установка типа МПТ-9. Эта установка представляет машину структурного типа и предназначена для исследования динамики сложных объектов и систем автоматического регулирования и управления, движение которых описывается обыкновенными линейными дифференциальными уравнениями с постоянными и переменными коэффициентами. Максимальный порядок решаемых уравнений — 16. Длительность процесса интегрирования 400 с.

Моделирующая установка выполнена в виде отдельных секций и блоков. Основным элементом ее схемы является усилитель постоянного тока с коэффициентом усиления $10 \cdot 10^6 — 20 \cdot 10^6$.

Установка допускает аппроксимацию заданных графиков переменных коэффициентов ступенчатыми кривыми, имеющими различные интервалы разбивки по оси времени. Величины этих интервалов изменяются программным переключением шаговых искателей основных блоков переменных коэффициентов. Постоянные коэффициенты задаются делителями напряжений трехзначным десятичным числом в диапазоне от 0 до 1.

Моделирующая установка пригодна для сопряжения исследуемых приборов управления и автоматического регулирования и совместной работы с ними как в режиме одноразового решения, так и в режиме автоматического повторения решения с индикацией результата на трубке электронно-лучевого индикатора. Имеется также возможность фиксации искомых переменных остановкой процесса интегрирования с последующим продолжением прерванного процесса. Решение можно регистрировать на шлейфовом осциллографе, самописце и др.

Основные функциональные блоки объединены в секции по 12 блоков операционных усилителей или блоков переменных коэффициентов в каждой. Делители напряжения размещаются в двух секциях управления по 24 в каждой. Управление функциональными блоками сосредоточено в блоках управления секций операционных усилителей и в пульте управления моделирующей установки.

Погрешность при моделировании устойчивых динамических систем не превосходит 10%.

АВМ, предназначенные для решения нелинейных дифференциальных уравнений. Наиболее широкое распространение для решения обыкновенных нелинейных дифференциальных уравнений получили АВМ

серии МН (модель нелинейная): МН-7, МН-10, МН-11, МН-14, МН-17, МН-18. Рассмотрим характеристики и возможности некоторых из них.

Электронная моделирующая установка типа МН-7. Нелинейная электронная моделирующая установка МН-7 является АВМ, предназначенной для исследования динамики систем и объектов автоматического регулирования, движение которых описывается обыкновенными дифференциальными уравнениями, имеющими небольшее количество нелинейных зависимостей. Максимальный порядок решаемых уравнений — 6. Длительность процесса интегрирования — 220 с. Для решения более сложных нелинейных систем возможна параллельная работа двух и более таких установок.

В установку МН-7 входят решающий блок, электронно-лучевой индикатор И-5М и блок питания ЭСВ-6.

В решающем блоке имеется 16 усилителей постоянного тока, выполняющих операции интегрирования, суммирования и масштабных преобразований, четыре диодных элемента, которые при совместном включении с усилителями позволяют моделировать типовые нелинейные зависимости (петля гистерезиса, сухое трение, зона нечувствительности и т. д.).

Машина имеет 24 делителя напряжения, 12 из которых позволяют менять коэффициенты передачи усилителей от 0,01 до 1 и 12 — от 1 до 10.

В комплекте машины имеется четыре блока перемножения двух переменных и четыре блока воспроизведения однозначных нелинейных функций от одной переменной. Одновременно при решении задачи возможно использование не более четырех нелинейностей.

В машине МН-7 может осуществляться либо одноразовое решение, либо автоматическое повторение решения. Кроме того, машина имеет схему совпадения, с помощью которой в процессе решения могут проводиться автоматическое сравнение двух переменных, простое изменение в структурной схеме задачи или же фиксация решения.

Результат решения может наблюдаться визуально на экране осциллографа И-5М или же регистрироваться внешними измерительными приборами (шлейфовым осциллографом, самописцем).

Погрешность блоков при изменении входных величин с частотами 0 до 10 Гц не превышает 1%.

Электронная моделирующая установка типа МН-11. Эта установка предназначена для решения обыкновенных дифференциальных уравнений до девятого порядка включительно. Она содержит девять интегрирующих усилителей, 38 суммирующих усилителей, шесть блоков перемножения, три блока переменных коэффициентов, а также четыре трехдекадных делителя напряжения.

Машина МН-11 имеет следующие режимы работы:

1) автоматическое отыскание решения задачи методом минимизации, когда поочередное изменение переменных величин производится на основе анализа предыдущих решений;

2) электрическое моделирование систем обыкновенных линейных или нелинейных дифференциальных уравнений до девятого порядка

с ручным подбором нескольких величин при реализации нескольких алгоритмов поиска или решения уравнений, представленных в нормальной форме Коши.

Машина может отыскивать шесть неизвестных параметров по шести заданным условиям для одной системы дифференциальных уравнений до девятого порядка.

Наблюдение за процессом поиска осуществляется с помощью электронно-лучевого индикатора, на экране которого одновременно можно наблюдать до четырех переменных.

Переход от задачи к задаче происходит за счет изменения схемы шнуровой коммутации, а также установки на коммутационном поле машины требуемых по расчету сменных резисторов, конденсаторов и диодных элементов.

Схема управления машины обеспечивает режим периодизации с частотой до 100 полных решений в секунду.

Электронная моделирующая установка типа МН-14. Нелинейная АВМ МН-14 относится к классу машин средней мощности стационарного типа и предназначена для моделирования сложных динамических систем, описываемых обыкновенными нелинейными дифференциальными уравнениями до 20-го порядка с большим числом нелинейностей.

В состав машины входят:

10 сдвоенных блоков интегрирующих усилителей, позволяющих одновременно осуществлять 20 операций интегрирования;

15 сдвоенных блоков суммирующих усилителей, позволяющих одновременно производить 30 операций суммирования;

10 сдвоенных инвертирующих блоков;

3 блока постоянного запаздывания;

10 вспомогательных усилителей, набор резисторов конденсаторов и диодных ячеек, позволяющих выполнять некоторые линейные, нелинейные и логические операции.

7 сдвоенных блоков переменных коэффициентов, осуществляющих ступенчатую аппроксимацию 12 функций;

50 блоков перемножения — деления, которые выполняют операции перемножения, деления и извлечения квадратного корня;

20 блоков универсальных нелинейностей, позволяющих воспроизводить нелинейные функции одной переменной.

Кроме того, в состав машины входят блоки для набора типовых нелинейных зависимостей систем автоматического регулирования, блоки для воспроизведения тригонометрических функций.

Длительность процесса интегрирования в МН-14 может находиться в пределах от 1 до 10 000 с. Набор задачи производится на съемном наборном поле.

Регистрация решения осуществляется с помощью электронно-лучевого индикатора, рассчитанного на четыре входа.

Измерение переменных величин производится как стрелочными, так и цифровыми вольтметрами, которые входят в состав машины.

Для фиксации решений к машине можно подключить самописцы и двухкоординатный регистратор.

ЛИТЕРАТУРА

Агеев М. И. Основы алгоритмического языка АЛГОЛ-60. ВЦ АН СССР, 1965.

Алгоритмический язык АЛГОЛ-60. Пересмотренное сообщение. «Мир», 1965.

Анисимов Б. В. и др. Основы вычислительной техники и программирования. МВТУ, 1970.

Анисимов Б. В., Четвериков В. Н. Основы теории и проектирования цифровых вычислительных машин. «Высшая школа», 1971.

Анисимов Б. В., Голубкин В. Н. Аналоговые вычислительные машины. «Высшая школа», 1971.

Балуев А. Н. и др. Сборник упражнений по АЛГОЛ-60. Л., Изд-во ЛГУ, 1967.

Беллман Р. Динамическое программирование. ИЛ, 1960.

Брудно А. Л. АЛГОЛ. «Наука», 1967.

Бусленко Н. П. Моделирование сложных систем. «Наука», 1968.

Бухтияров А. М., Зикевская Л. М., Фролов Г. Д. Сборник задач по программированию. «Наука», 1970.

Вентцель Е. С. Исследование операций. «Советское радио», 1972. Вычислительная система IBM-360. Принципы работы. «Советское радио», 1969.

Гермейер Ю. Б. Введение в теорию исследования операций. «Наука», 1971.

Голенко Д. И. Моделирование и статистический анализ псевдослучайных чисел на электронных вычислительных машинах. «Наука», 1965.

Грубов В. И., Кирдан В. С. Электронные вычислительные машины и моделирующие устройства. Киев, «Наукова думка», 1969.

Гутер Р. С., Резниковский П. Т., Резник С. М. Программирование и вычислительная математика. Вып. 1, 2. «Наука», 1971.

Данциг Дж. Линейное программирование, его применения и обобщения. «Прогресс», 1966.

Демидович Б. П., Марон И. А. Основы вычислительной математики. Физматгиз, 1963.

Демидович Б. П., Марон И. А., Шувалова Э. З. Численные методы анализа. Физматгиз, 1963.

Джермейн К. Программирование на IBM/360. «Мир», 1971.

Каган Б. М., Каневский М. М. Цифровые вычислительные машины и системы. «Энергия», 1970.

Коган Б. Я. Электронные моделирующие устройства и их применение для исследования систем автоматического регулирования. Физматгиз, 1963.

Корн Г., Корн Т. Электронные аналоговые и аналого-цифровые вычислительные машины. «Мир», 1967.

Кофман А., Дебазей Г. Сетевые методы планирования и их применение. «Прогресс», 1968.

Лавров С. С. Универсальный язык программирования (АЛГОЛ-60). «Наука», 1972.

Левин Л. Методы решения технических задач с использованием аналоговых вычислительных машин. «Мир», 1966.

Мак-Кракен Д. Д., Дорн У. С. Численные методы и программирование на ФОРТРАНе. «Мир», 1969.

Меррей-Шелли Р. Программирование на ФОРТРАНе. «Энергия», 1971.

Папернов А. А. Логические основы цифровой вычислительной техники. «Советское радио», 1972.

Первин Ю. А. Основы ФОРТРАНа. «Наука», 1972.

Петров А. В. и др. Сборник задач и упражнений для программирования в автокоде «Инженер», МВТУ, 1971.

Прохоров В. И., Погорелко И. А., Яковлев А. А. Основы программирования для электронных цифровых вычислительных машин. «Высшая школа», 1967.

Решение задач надежности и эксплуатации на универсальных ЭЦВМ. Под ред. Шишонок Н. А. «Советское радио», 1967.
Савинков В. М. Программирование для ЭВМ «Минск-22»: «Статистика», М., 1972.
Система обработки экономической информации на ЭВМ «Минск-22». «Статистика», 1969.
Сэксон Д. КОБОЛ. «Статистика», 1970.
Фиакко А., Мак-Кормик Г. Нелинейное программирование. «Мир», 1972.
Черкасова М. П. Сборник задач по численным методам. Минск, «Вышэйшая школа», 1967.
Юдин Д. Б., Гольштейн Е. Г. Линейное программирование. Физматгиз, 1963.
Языки программирования. Под ред. Женюи Ф. «Мир», 1972.

ОГЛАВЛЕНИЕ

Алексей Викторович Петров,
Владимир Евтихиевич Алексеев,
Михаил Александрович Титов,
Валентин Иванович Суровцев,
Петр Николаевич Шкатов

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА В ИНЖЕНЕРНЫХ И ЭКОНОМИЧЕСКИХ РАСЧЕТАХ

Редактор Л. П. Андрианова, переплет художника В. З. Казакевича, художественный редактор Н. К. Гуторов, технический редактор С. П. Передерий, корректор В. В. Кожуткина

Сдано в набор 19/IX 1974 г. Подп. к печати 17/II 1975 г.
Т—03169. Формат 60×90¹/₁₆. Бумага типографская № 3.
Объем 19 печ. л. Усл. печ. л. 19. Уч.-изд. л. 19,69.
Изд. № СТД—210. Тираж 89 000 экз. Цена 85 коп.
План выпуска литературы издательства
«Высшая школа» (вузы и техникумы) на 1975 г.
Позиция № 146.
Москва К-51, ул. Неглинная, 29/14,
Издательство «Высшая школа»

Московская типография № 4 Союзполиграфпрома
при Государственном комитете Совета Министров СССР
по делам издательств, полиграфии и книжной торговли,
Москва, И-41, Б. Переяславская ул., дом 46
Зак. 1153.