ВЫСШЕЕ ОБРАЗОВАНИЕ

УЧЕБНИК

ВЫЧИСЛИТЕЛЬНАЯ И МИКРОПРОЦЕССОРНАЯ ТЕХНИКА В УСТРОЙСТВАХ ЭЛЕКТРИЧЕСКИХ ЖЕЛЕЗНЫХ ДОРОГ

·ТРАНСПОРТ·

ВЫСШЕЕ ОБРАЗОВАНИЕ
УЧЕБНИК

ВЫЧИСЛИТЕЛЬНАЯ И МИКРОПРОЦЕССОРНАЯ ТЕХНИКА В УСТРОЙСТВАХ ЭЛЕКТРИЧЕСКИХ ЖЕЛЕЗНЫХ ДОРОГ

Под редакцией Г.Г. Марквардта

Утверждено
Управлением учебных заведений МПС
в качестве учебника для студентов вузов
железнодорожного транспорта

МОСКВА "ТРАНСПОРТ" 1989

УДК 681.3:621.33

Вычислительная и микропроцессорная техника в устройствах электрических железных дорог: Учебник для вузов ж.-д. трансп./В. В. Андреев, П. Б. Куликов, Г Г. Марквардт и др.; Под ред. Г Г Марквардта.— М.: Транспорт, 1989.—287 с.

Приведены общие сведения об архитектуре современных ЭВМ. Изложены основы и способы программирования на алгоритмических языках Бейсик и Фортран IV, используемых решений задач тягового электроснабжения и электроподвижного состава. Рассмотрены вопросы применения микропроцессорных систем и микроЭВМ в автоматизированных системах управления тяговым электроснабжением и электроподвижным составом. Книга предназначена в качестве учебника для студентов высших учебных заведений железнодорожного транспорта и может быть полезна инженерно-техническим работникам, занятым в области автоматизации управления устройствами тягового электроснабжения и электроподвижного состава.

Ил. 103, табл. 21, библиогр. 36 назв.

Книгу написали: введение, гл. 1, 6 и параграф 7.7 — Г Г Марквардт, гл. 3 — В. П. Феоктистов, гл. 4 — В. В. Андреев, гл. 5 — П. Б. Куликов, гл. 2 и параграфы 7.1—7.6 — Е. К. Рыбников.

Рецензенты доктора технических наук Н. Д. Сухопрудский и Е. П. Фигурнов

Заведующий редакцией В. П. Репнева

Редакторы Н. В. Зенькович и С. А. Каткова

В $\frac{3202010000\text{-}360}{049(01)\text{-}89}$ 90-89

ISBN 5-277-0595-1

ВВЕДЕНИЕ

Научно-технический прогресс и связанная с ним интенсификация экономики приводят к необходимости проектирования и эксплуатации объектов, насыщенных дорогими и сложными устройствами. Это в полной мере относится и к железнодорожному транспорту, в частности, к электрифицированным железным дорогам.

Возрастающие темпы электрификации железных дорог, связанные с большими капитальными вложениями, требуют коренного пересмотра методов проектирования электрификации с целью повышения экономической эффективности и надежности работы электрифицированных железных дорог.

Выполнение этих требований возможно лишь при высокой автоматизации процессов управления устройствами электроснабжения и электроподвижным составом.

Основными направлениями экономического и социального развития СССР на 1986—1990 годы и на период до 2000 года предусмотрен рост объема производства вычислительной техники, запланировано наращивание масштабов применения современных высокопроизводительных электронно-вычислительных машин (ЭВМ) всех классов.

Решение поставленных перед железнодорожным транспортом задач невозможно без широкого применения ЭВМ, в использовании которых можно выделить два основных направления:

чисто вычислительные работы (при проектировании и эксплуатации), связанные с выбором экономичных вариантов электрификации и оптимальных режимов работы электроснабжающих устройств;

управление режимами работы отдельных устройств электрической железной дороги и всей системой ее электроснабжения в целом.

Некоторая условность приведенного разделения определяется тем, что во многих случаях для воздействия на управляемые объекты ЭВМ выполняют подчас очень большую вычислительную работу. По числу взаимодействующих объектов, их сложности и многообразию связей между ними электрифицированная железная дорога по современной терминологии относится к разряду сложных или больших систем [1]

Полное аналитическое описание процессов, протекающих на электрифицированных железных дорогах, невозможно. Поэтому без применения ЭВМ очень трудно обеспечить грамотное проектирование и решать ряд эксплуатационных задач.

В управлении режимами работы системы тягового электроснабжения и электроподвижного состава все более широкое применение находят микропроцессорные устройства. Они чрезвычайно быстро совершенствуются, стоимость их постоянно снижается. В недалеком будущем они без сомнения станут основными элементами устройств автоматики, управления и защиты электрических железных дорог. Будущий инженер-электрификатор, работая проектировщиком или эксплуатационником, не сможет обойтись без ЭВМ. Поэтому хорошая подготовка такого инженера к работе на ЭВМ и к использованию микропроцессорной техники особенно необходима.

Важность данной задачи подчеркивается тем, что за рубежом как в общей энергетике, так и на электрифицированных железных дорогах уже действуют основанные на этой базе системы автоматизированного управления.

Большие работы в этом направлении ведутся в СССР. Во Всесоюзном научно-исследовательском институте железнодорожного транспорта (ВНИИЖТ), Московском институте инженеров железнодорожного транспорта (МИИТ), Ростовском институте инженеров железнодорожного транспорта (РИИЖТ), Всесоюзном заочном институте инженеров железнодорожного транспорта (ВЗИИТ), Днепропетровском институте инженеров железнодорожного транспорта (ДИИТ) проведены исследовательские, проектные и экспериментальные работы по созданию автоматизированной системы управления (АСУ) устройствами тягового электроснабжения. Уже эксплуатируются два автоматизированных энергодиспетчерских пункта, разработанных РИИЖТом. Учеными ВЗИИТа создан действующий макет контроля и управления тяговой подстанцией. Другие транспортные вузы также ведут подобные работы, координируемые РИИЖТом. Специалисты ВНИИЖТа, МИИТа, ВЗИИТа изучают возможности применения микропроцессорной техники в области управления и диагностики электроподвижного состава.

Рассмотрение всех перечисленных вопросов выходит за рамки данного курса, посвященного изучению основных технических средств вычислительной техники и их использованию на электрифицированных железных дорогах. Детальному изучению возможностей ЭВМ и микропроцессорной техники посвящены другие специальные дисциплины.

Настоящий учебник предназначен для студентов специальности «Электрификация железнодорожного транспорта». В нем общие вопросы вычислительной техники по возможности увязаны с профилем работы будущего инженера, который должен уметь предельно четко ставить задачу и полностью формализовать ее решение. Для этого необходимо научиться строить и обосновывать выбранную модель объекта, составлять алгоритм, а затем и программы решения задачи. Этот процесс несомненно дисциплинирует умственную деятельность обучающегося, делает ее более целенаправленной и позволяет теснее связать специальные дисциплины с вычислительной техникой.

Глава 1

МОДЕЛИ И АЛГОРИТМЫ

1.1. Моделирование как этап познания

Решая любую конкретную задачу мы, как правило, начинаем с наблюдения связанных с этой задачей явлений или поведения некоторых систем, проводим их анализ и выявляем факторы, имеющие в данном случае существенное значение.

Далее, используя полученный фактически материал о протекании изучаемого процесса и взаимодействий различных влияющих на него факторов, мы отвлекаемся от самого изучаемого явления и составляем свое представление об определяющих его закономерностях.

Объективность установленных закономерностей проверяется сопоставлением их с данными, полученными экспериментально. Такой путь решения задачи проходят все исследователи при решении новых задач.

В современной терминологии вторая стадия решения задачи называется процессом создания модели изучаемого явления. После опытной проверки созданной модели (третья стадия) далее она используется для различного рода практических выводов и расчетов.

Особое значение имеет исследование систем математическими методами.

Для таких исследований необходимо построить соответствующую математическую модель. Из сказанного ранее вытекает, что при построении модели вообще, а следовательно, и математической модели в частности отбрасываются факторы, не имеющие существенного значения при решении конкретной задачи. Из этого вытекает, что при одинаковой природе явления математические модели для решения разных задач могут иметь существенно разный вид.

Важно, однако, то, что при решении любой задачи математическими методами реальному объекту ставится в соответствие определенный математический объект, который называется его *моделью*. Математические модели делят на аналитические и имитационные [1].

1.2. Формализация простых процессов в системе электрической тяги. Аналитические модели

В аналитических моделях процессы функционирования системы или ее элементов записываются в виде функций и логических условий.

Если уравнения аналитической модели могут быть решены в явном виде относительно искомых величин, то такие модели исследуют-

ся аналитическими методами. Если же уравнения аналитической модели в общем виде не могут быть решены, то прибегают к численным методам получения результатов с применением ЭВМ. В некоторых случаях только использование ЭВМ позволяет получить необходимые результаты. Это относится к тем случаям, когда система уравнений математической модели теоретически разрешима численными методами при конкретных начальных данных, но вычислительный процесс связан с очень большим числом операций.

Ниже даны примеры составления моделей, в которых показано, как для одного и того же объекта для решения разных задач строятся разные математические модели.

Пример 1. Рассмотрим фидерную зону однопутного участка, нагруженного пакетом из трех движущихся в одну сторону поездов (рис. 1.1). Поставим две задачи:

определить среднее напряжение на локомотиве за время хода его по произвольно выбранному участку пути ab;

определить максимальные значения резких изменений токов локомотивов при отключении от сети одного из них.

Обе задачи имеют практическое значение. Рещение первой зависит от способа выбора длины участка ab и связано с корректировкой пропускной способности участка по уровню напряжения или с проверкой выполнения требования ПТЭ железных дорог к уровню напряжения.

Для упрощения обеих задач будем считать, что напряжения на подстанциях A и B поддерживаются равными и неизменными независимо от их нагрузки. Примем также, что за рассматриваемый промежуток времени поезда не переходят с одного элемента профиля на другой.

В первой задаче речь идет об усреднении напряжения за относительно большой промежуток времени 2—10 мин. Установившаяся скорость поезда даже при больших отклонениях напряжения изменяется значительно быстрее. Это позволяет принять токи локомотивов соответствующими установившимся значениям скорости движения поездов. Но при движении по одному и тому же элементу профиля ток локомотива при изменении напряжения остается практически неизменным. Это объясняется тем, что в пределах сравнительно малых изменений скорости, вызванных изменением напряжения, сопротивление движению и равная ему (при установившейся скорости) сила тяги изменяются незначительно. Следовательно, мало изменится и ток поезда. Последнее объясняется тем, что у локомотивов с двигателями последовательного возбуждения тяговое усилие зависит только от тока.

С учетом сделанных замечаний для решения первой задачи модель рассматриваемой системы может быть представлена в виде схемы замещения (рис. 1.2)

Аналитическую зависимость потери напряжения в контактной сети от подстанции до точки расположения второго поезда можно записать в виде

$$\Delta U_2 = R_{21} I_1 + R_{22} I_2 + R_{23} I_3$$

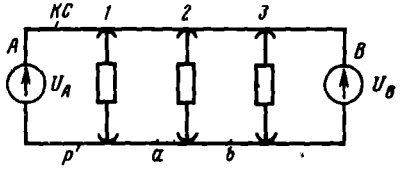


Рис. 1.1. Схема расположения нагрузок в фидерной зоне:
U_A, U_B — напряжения на подстанциях A и B; КС — контактная сеть; P — рельсы; 1, 2, 3— нагрузки

В более общем случае, когда общее число нагрузок равно m, а потерю напряжения надо определить до точки расположения локомотива k, получим:

$$\Delta U_k = \sum_{j=1}^{m} R_{jk} I_j, \qquad (1.1)$$

где R_{jk} — взаимное сопротивление между точками j и k.

При $j=k$ сопротивление R_{kk} называется входным сопротивлением.

Нетрудно показать, что

$$R_{jk} = \left\{ \begin{matrix} rl_j(l-l_k)/l \text{ при } j \leqslant k \\ rl_k(l-l_j)/l \text{ при } j \geqslant k \end{matrix} \right\}, \quad (1.2)$$

где r — сопротивление 1 км тяговой сети.

Для решения первой задачи надо по формулам (1.1) и (1.2) найти ряд значений ΔU_k при перемещении поезда k от точки a до точки b с учетом перемещения других поездов и изменения всех нагрузок. По полученной таким образом зависимости $\Delta V_k(t)$ находят среднее значение этой величины за время хода поезда по отрезку ab [24].

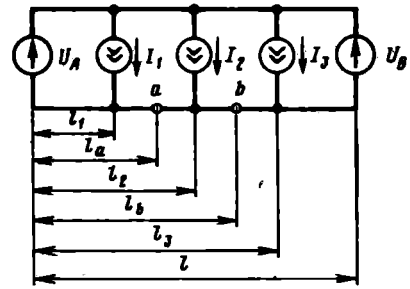


Рис. 1.2. Схема замещения для расчета токов в установившемся режиме:
I_1, I_2, I_3 токи локомотивов

Решение второй задачи связано с рассмотрением резких изменений напряжения в отдельные моменты времени. Поскольку скорости поездов не могут изменяться мгновенно, при решении этой задачи нельзя принять токи поездов неизменными, но можно считать, что скорости поездов в течение электрического переходного процесса остаются равными их начальным значениям. В магнитной цепи тяговых двигателей имеются массивные участки, в которых при изменении магнитного потока возникают вихревые токи, замедляющие изменение потока. В первом приближении для моментов, близких к начальному, можно принять, что магнитные потоки в двигателях неотключаемых локомотивов остаются неизменными.

Поскольку ранее было принято, что скорости локомотивов в рассматриваемый период не меняются, то остаются неизменными и э. д. с. двигателей $E_д$, поскольку $E_д = C\Phi v$.

С учетом сказанного для решения второй задачи можно использовать модель, которая в виде схемы замещения представлена на рис. 1.3. Для неустановившегося процесса, возникающего при отключении от сети одного из локомотивов, аналитическая модель записывается в виде системы дифференциальных уравнений. Здесь нет необходимости рассматривать составление этой системы и ее решение. Формализация задачи практически уже выполнена, как только составлена схема замещения.

Таким образом при решении разных задач для одного и того же объекта пришлось построить совершенно разные математические модели, хотя объект исследования был один и тот же.

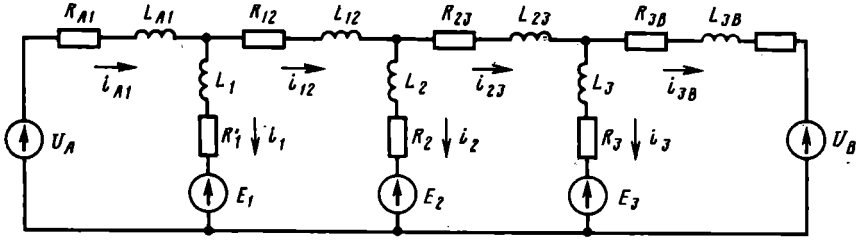


Рис. 1.3. Схема замещения для расчета токов в неустановившемся режиме в нормальный условиях:
U_A, U_B — напряжения на подстанциях; R_{A1}, R_{12}, R_{23}, R_{3B} — сопротивления участков тяговой сети; L_{A1}, L_{12}, L_{23}, L_{3B} — индуктивности соответствующих участков тяговой сети; i_{A1}, i_{12}, i_{23}, i_{3B} — мгновенные значения токов в соответствующих участках тяговой сети; R_1, R_2, R_3 — сопротивления силовых цепей электровозов; L_1, L_2, L_3 — индуктивности силовых цепей электровозов; i_1, i_2, i_3 — мгновенные значения токов электровозов; E_1, E_2, E_3 — э. д. с. электровозов

Пример 2. Рассмотрим теперь короткое замыкание нагруженной тяговой сети (рис. 1.4). Поставим опять две задачи:

определить ток фидера по окончании неустановившегося процесса, вызванного коротким замыканием;

определить скорость нарастания тока фидера в момент возникновения короткого замыкания.

Обе эти задачи имеют практическое значение. Решение первой необходимо для проверки работы так называемой импульсной защиты, которая реагирует на разность установившихся значений тока — тока до начала неустановившегося режима и тока в конце его. Эта разность будет меньше, если короткое замыкание происходит в момент, когда сеть нагружена, следовательно, именно в таком случае возникает опасность несрабатывания защиты.

Подобная же задача, но при отсутствии нагрузки возникает при использовании токовой защиты и дистанционной защиты ВНИИЖТа. Вторая задача требует решения при использовании защиты тяговой сети постоянного тока, реагирующей на скорость нарастания тока в начальный момент неустановившегося процесса. Скорость нарастания тока, а вместе с ней и чувствительность защиты будут меньше в случае короткого замыкания нагруженной сети.

Составляя модель для решения первой задачи, надо учесть, что электрический процесс даже с учетом тормозящего действия вихревых токов в магнитопроводах тяговых двигателей протекает во много раз быстрее, чем процесс установления новой скорости движения локомотива в связи с изменением напряжения в месте расположения электровоза. Поэтому при построении модели можно считать скорость неизменной. Но в этом случае при изменении магнитного потока будет изменяться и э. д. с. электровоза. Так как установившееся значение потока зависит от тока, то при неизменной скорости движения локомотива э. д. с. его двигателей $E_э$ зависит от их токов $I_э$. Функция $E_э(I_э)$ может быть получена по кривой намагничивания тягового двигателя или по его тяговой и токовой характеристикам.

С учетом этого математическая модель объекта для решения первой задачи может быть построена на основе схемы замещения, приведенной на рис. 1.5. Математическая модель в этом случае может быть представлена следующими уравнениями, составленными на основе законов Кирхгофа:

$$\begin{aligned} R_{пэ} I_{фк} + R_э I_э &= U - E(I_э) \\ R_к I_к - R_э I_э &= E_э(I_э) - E_д(I_к); \\ I_{фк} &= I_э + I_к, \end{aligned} \qquad (1.3)$$

где $E_д(I_к)$— зависимость падения напряжения в дуге в месте короткого замыкания от тока; часто $E_д(I_к)$ принимают постоянной и равной 150—200 В.

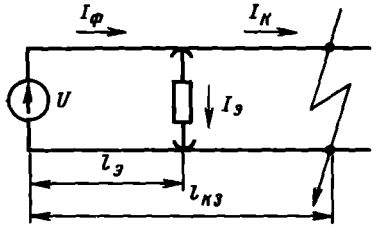


Рис. 1.4. Схема нагруженной тяговой сети при коротком замыкании:
$I_э$ — ток электровоза; $I_ф$, $I_к$ — токи на участках тяговой сети; U — напряжение на шинах подстанции

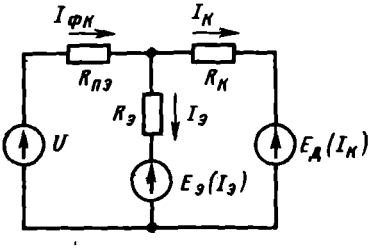


Рис. 1.5. Схема замещения для расчета установившихся токов в нагруженной тяговой сети при коротком замыкании:
$R_{пэ}$, $R_к$ — сопротивления участков тяговой сети; $R_э$ — сопротивление силовой цепи электровоза; $E_э$ $(I_э)$ — э. д. с. электровоза; $E_д(I_к)$ — падение напряжения в дуге

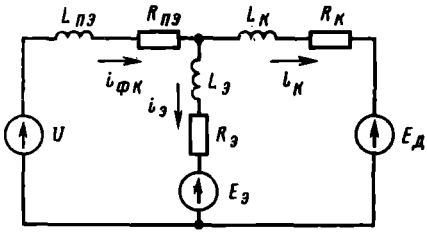


Рис. 1.6. Схема замещения для расчёта скорости нарастания тока в нагруженной тяговой сети при коротком замыкании:
$L_{пэ}$, $L_к$ — индуктивности участков тяговой сети; $L_э$ — индуктивность силовой цепи электровоза; $i_{фк}$, $i_к$ — мгновенные значения токов в тяговой сети; $i_э$ — мгновенное значение тока электровоза

Как видно, модель, представленная уравнениями (1.3), несложна. Однако решение системы уравнения без применения ЭВМ связано с большой затратой времени, так как функция $E_э(I_э)$ нелинейна и большей частью задается графически. На ЭВМ эта задача решается просто и быстро, как при представлении зависимости $E_э(I_э)$ в табличной форме, так и при аппроксимации ее каким-либо выражением.

Совершенно другая модель должна быть составлена для решения второй задачи. Здесь надо рассматривать неустановившийся процесс, поэтому, кроме сопротивлений, необходимо учитывать и индуктивность проводников.

Так как требуется определить скорость нарастания тока фидера в момент начала неустановившегося процесса, то можно, как было показано на примере 1, принять э. д. с. электровоза равным его значению до короткого замыкания. Схема замещения при построении математической модели для решения данной задачи приведена на рис. 1.6.

Для составления аналитических выражений воспользуемся вновь законами Кирхгофа.

$$\left.\begin{aligned} L_{пэ}\frac{di_{фк}}{dt} + R_{пэ}i_{фк} + L_э\frac{di_э}{dt} + R_э i_э = U - E_э;\\ L_к\frac{di_к}{dt} + R_к i_к - L_э\frac{di_э}{dt} - R_э i_э = E_э - E_д;\\ i_{фк} = i_э + i_к. \end{aligned}\right\} \qquad (1.4)$$

В начале неустановившегося процесса $i_э(0)=I_{эо}$, $i_к(0)=0$, $i_{фк}(0)=I_{эо}$.
Обозначим скорость нарастания токов в начальный момент:

$$\left.\begin{aligned} \frac{di_{фк}}{dt_0} = v_{Iфо};\\ \frac{di_к}{dt_0} = v_{Iко};\\ \frac{di_э}{dt_0} = v_{Iэо}. \end{aligned}\right\} \qquad (1.5)$$

Тогда вместо системы (1.4) получим

$$\begin{aligned} L_{пэ}v_{Iфо} + R_{пэ}I_{эо} + L_э v_{Iэо} + R_э I_{эо} = U - E_{эо};\\ L_к v_{Iко} - L_э v_{Iэо} - R_э I_{эо} = E_{эо} - E_д;\\ v_{Iфо} = v_{Iэо} + v_{Iко}. \end{aligned} \qquad (1.6)$$

Система из трех линейных уравнений содержит три неизвестных и поэтому легко решается.

Как видно, математические модели рассматриваемого объекта, составленные для решения различных задач, существенно различаются.

Пример 3. Рассмотрим в качестве объекта поезд и поставим две задачи: определить зависимость скорости его движения и пройденного пути от времени; определить динамические усилия, действующие в различных частях поезда.

В ряде работ показано, что решая первую задачу, можно представлять поезд в виде материальной точки, обладающей некоторой массой m. В этом случае математическая модель процесса движения поезда может быть представлена очевидными выражениями:

$$\left.\begin{aligned} m\frac{dv}{dt} &= F_э(v) - W(v); \\ s(t) &= \int_{t_0}^{t} v dt, \end{aligned}\right\} \qquad (1.7)$$

где v — скорость движения поездов; $F_э(v)$ — зависимость силы тяги электровоза от скорости; $W(v)$ — зависимость сопротивления движению поезда от скорости; $s(t)$ — зависимость пройденного поездом пути от времени.

Такова известная из теории тяги математическая модель процесса движения поезда.

Для решения второй задачи, в которой требуется определить силы, действующие внутри поезда, она явно непригодна, так как поезд был представлен материальной точкой.

Решение второй задачи очень сложно и здесь неуместно сколь-нибудь подробно останавливаться на различных моделях, предложенных для ее решения. Укажем лишь на то, что в составленной для ее решения модели необходимо учитывать рассредоточение общей массы поезда, неравенство скоростей движения вагонов в одинаковые моменты времени и др.

Пример 4. В качестве последнего примера рассмотрим простую подвеску провода (рис. 1.7) и поставим две задачи:

определить зависимость стрелы провеса провода от горизонтальной составляющей натяжения;

найти стрелу провеса провода при температуре t_x и нагрузке на один метр g_x, если известна стрела провеса f_1 при температуре t_1 и нагрузке g_1.

Составляя модель для решения первой задачи, можно воспользоваться тем, что в контактной сети стрелы провеса настолько малы, что длина любого отрезка провода незначительно отличается от длины его проекции на горизонталь.

С учетом этого можно, разрезав пролет в середине и заменив отброшенную правую его часть горизонтальной составляющей силы натяжения H, составить расчетную схему, приведенную на рис. 1.8. Взяв сумму моментов всех сил относительно точки A и приравняв ее нулю (поскольку система находится в равновесии), получим:

$$fH - \frac{l}{4}\frac{gl}{2} = 0,$$

откуда

$$f = \frac{gl^2}{8H}. \qquad (1.8)$$

Решение второй задачи связано с учетом деформации провода в связи с изменением его натяжения и температуры. В этом случае, конечно, нельзя считать, что

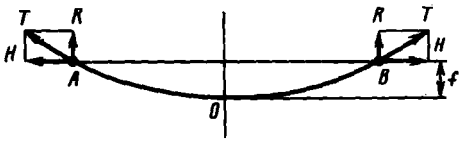


Рис. 1.7 Схема расположения провода при простой подвеске:
H — горизонтальные; R — вертикальные составляющие реакции T в точке подвеса провода; f — стрела провеса; A и B — точки крепления провода

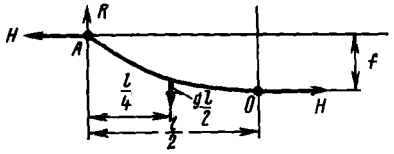


Рис. 1.8. Расчетная схема для определения стрелы провеса провода:
l — длина пролета; $gl/2$ — сила тяжести половины пролета провода

длина провода равна длине его проекции (длине пролета), т. е. неизменна. Длину провода в пролете длиной l с достаточной точностью можно определить по формуле:

$$L = l + \frac{8}{3}\frac{f^2}{l},$$

а удлинение провода при изменении метеорологических условий

$$\Delta L = L_x - L_1 \approx \frac{8}{3l}(f_x^2 - f_1^2). \qquad (1.9)$$

Это же удлинение можно найти, рассматривая температурное и упругое удлинение провода. Оно с малой погрешностью может быть представлено в виде

$$\Delta L = l\left[\alpha(t_x - t_1) + \frac{H_x - H_1}{ES}\right], \qquad (1.10)$$

где α — температурный коэффициент линейного расширения провода; E — модуль упругости; S — площадь поперечного сечения провода.

Приравнивая правые части выражений (1.9) и (1.10) и учитывая формулу (1.8), получаем аналитическую запись математической модели, составленной для решения второй задачи.

Приведенные примеры формализации свойств реального объекта иллюстрируют сделанное в начале параграфа утверждение, что математическая модель объекта может изменяться и, как правило, изменяется в зависимости от задачи, для решения которой она составлена.

1.3. Имитационное моделирование

В предыдущем параграфе рассмотрены модели некоторых объектов для решения конкретных задач. В силу простоты объектов их математические модели во всех случаях удалось дать в виде аналитических зависимостей. Для всех задач, кроме последней задачи примера 4, аналитические выражения разрешаются в явном виде относительно исследуемого параметра.

Применение ЭВМ для решения этих задач оправдано только в случае достаточно большого массива исходных данных. Большинство такого рода задач может быть решено при единичных расчетах на простых, а при многократно повторяющихся расчетах (по одним и тем же аналитическим выражениям) на программируемых калькуляторах.

Конечно, имеется множество аналитических выражений, решение которых возможно только на мощных ЭВМ.

Электрифицированная железная дорога представляет собой систему, состоящую из многих подсистем. Они имеют сложные взаимосвязи друг с другом, а вся система в целом связана с внешними системами, такими, как энергосистема, система организации движения поездов. Такого рода системы называются сложными системами. Сложными системами являются и отдельные ее части, такие, как система тягового электроснабжения, отдельные единицы электроподвижного состава и др.

В таких системах исследование свойств отдельных ее элементов хотя и необходимо, но недостаточно. Конечной задачей является анализ поведения всей системы в целом и изучение ее реакции на те или иные управляющие сигналы. Это необходимо в процессе проектирования системы и при управлении ею во время эксплуатации.

Моделированию сложных систем посвящено много специальных исследований, имеется обширная литература, посвященная им.

Уже давно хорошо осознано влияние характеристик отдельных элементов, составляющих систему, на свойства системы в целом. Однако только с появлением ЭВМ создались условия для проведения так называемого *системного анализа*. Для его проведения разработан специальный метод, названный имитационным моделированием.

При имитационном моделировании разрабатывается порядок численных и логических операций (моделирующий алгоритм), с помощью которого с той или иной степенью приближения воспроизводится процесс в оригинале. Моделирующий алгоритм, реализуемый на ЭВМ, дает возможность при заданных начальных данных и параметрах воспроизвести формализованный процесс в сложной системе.

В области электрификации железных дорог имитационное моделирование [2] в последнее время стало широко применяться в проектных и эксплуатационных расчетах тягового электроснабжения, поэтому ниже речь будет идти в основном об этой части общей системы электрической тяги. Несомненно, что разработка и создание новых локомотивов, представляющих сложные системы, открывают широкие перспективы применения имитационного моделирования и в области электроподвижного состава.

Как указывалось, при имитационном моделировании приближенное воспроизведение реального процесса осуществляется на основе не только аналитических зависимостей, но и при помощи моделирующего алгоритма [1]. Следовательно, имитационное моделирование является не теорией, а методологией решения проблемы.

Имитационное моделирование включает конструирование модели реальной системы, постановку эксперимента на модели для изучения системы и выбор стратегии решения задач. Сам процесс моделирования поэтому тесно связан с системным анализом (анализом динамики реального процесса, составлением и структуризацией модели).

При имитационном моделировании какого-либо процесса воспроизводится состояние объекта, изменяемое во времени. Новое (обновленное) состояние его зависит от предыдущего и внешних воздействий на объект.

Основным преимуществом имитационных моделей по сравнению с аналитическими — возможность решения задач исключительной сложности, каковыми являются многие задачи проектирования и оптимального управления на электрифицированной железной дороге и в системе тягового электроснабжения. Имитационную модель мы можем использовать для математического эксперимента,

обычно более быстрого и дешевого, чем эксперимент в натуре, который иногда вообще невозможно провести.

Еще одним существенным преимуществом имитационной модели является то, что введение в нее случайных возмущений не вносит в модель принципиальных усложнений, а учет их в аналитических моделях сложных систем приводит часто к непреодолимым трудностям.

При выборе параметров многих устройств эллектрифицированных железных дорог или проверке их работоспособности в эксплуатации, как правило, приходится исследовать воздействия на эти устройства случайных процессов.

Случайным процессом изменения какой-либо величины $x(t)$ (тока, напряжения, температуры и др.) называется такой процесс, который при одних и тех же условиях протекает по разным законам.

Моделирование системы электроснабжения электрифицированных железных дорог поэтому почти всегда носит вероятностный характер. Это особое направление моделирования сложных систем, в которых случайности обусловлены не только внешними, но и внутренними причинами (в самой системе).

Всякий объект имеет практически бесконечное число взаимных связей с другими объектами. Иначе говоря, он функционирует в некоторой среде. Система тягового электроснабжения взаимодействует со всеми элементами электроэнергетической сети (рис. 1.9) и ее

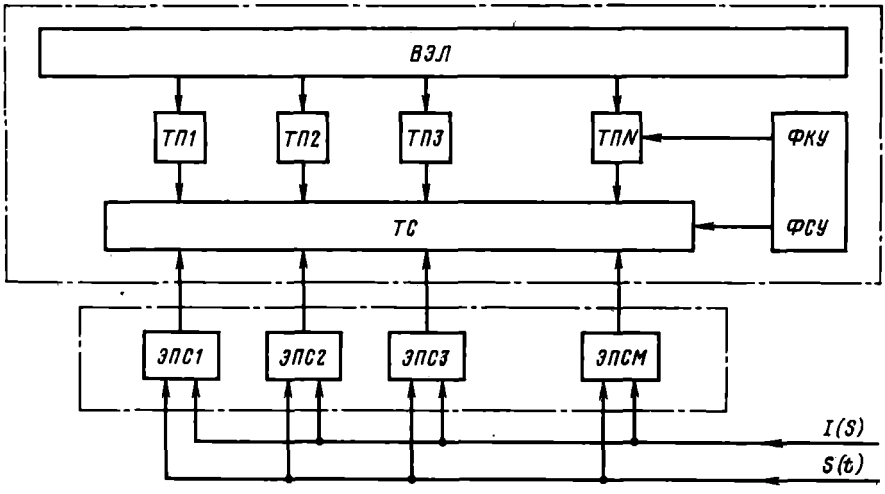


Рис. 1.9. Структурная схема системы электроснабжения:
ВЭЛ — система внешнего электроснабжения; *ТП1*, ..., *ТПN* — подстанции; *ТС* — тяговая сеть; *ЭПС1*, ..., *ЭПСМ* — электровозы; *ФКУ* — управляющее воздействие, определяющее режимы работы подстанции; *ФСУ* — управляющее воздействие, формирующее схему секционирования и питания контактной сети; $I(s)$ — заданные токи в зависимости от координаты расположения электровоза; $s(t)$ — заданные координаты расположения электровозов в зависимости от времени

потребителями, с электроподвижным составом, с системой организации движения поездов и др. Надо заметить, что определение самой системы электроснабжения электрических железных дорог и окружающей среды неоднозначно и подвержено изменениям по мере развития методов расчета и технических средств. Поэтому надо проявлять большую осторожность при определении границ изучаемой системы и устанавливать их в зависимости от цели исследования.

Как уже указывалось, модель надо ориентировать на решение определенных вопросов, а не стремиться имитировать реальную систему в подробностях, которые не отражаются на результатах.

Возможности моделирования зависят от:

внутреннего математического обеспечения (математическое и техническое обеспечение процесса счета, программы и устройства, обеспечивающие диалог с машиной);

внешнего математического обеспечения (методы формирования моделей, принципы постановки машинных экспериментов, методы анализа моделей и принятия решений).

Совокупность алгоритмов и программ, входящих во внутреннее и внешнее математическое обеспечение, используемая при машинных экспериментах, составляет имитационную систему. Несмотря на ряд проведенных работ в данной области и практическое их использование, разработка такой системы еще далеко не закончена.

Модель системы электроснабжения и ее программная реализация должны строиться по общим правилам, т. е. иметь модульную структуру. В противном случае любая корректировка или развитие модели потребует составления каждый раз новых программ. В этой связи очень продуктивен развитый Н. П. Бусленко [1] агрегативный подход, при котором сложная система математически расчленяется на конечное число подсистем.

В результате исследуемая система формализуется в виде унифицированной агрегативной системы, что значительно расширяет возможности формального описания объектов. В этом случае становится возможным применить системный анализ к объектам большой сложности.

В связи со сказанным в моделирующем алгоритме можно выделить три подалгоритма:

моделирующий какой-либо процесс;

учитывающий взаимодействие элементарных подпроцессов;

обеспечивающий согласованную работу подалгоритмов при реализации модели на ЭВМ.

Агрегативный подход благодаря модульной структуре модели и дискретному характеру обмена сигналами позволяет шире использовать внешнюю память машины, что в ряде случаев снимает ограничения по сложности задачи. Исследование сложных систем с помощью имитационных моделей, учитывающих случайные факторы, называют статистическим моделированием, хотя имитационное моделирование применимо и для детерминированных процессов.

Заметим, что статистическое моделирование — это эффективный метод изучения сложных процессов с помощью имитационных моделей.

Метод статистического моделирования позволяет вычислить значение любой функции, зависящей от всего хода рассматриваемого процесса, заданного на множестве реализаций процесса функционирования изучаемой случайной системы. Для этого надо только предусмотреть соответствующую подпрограмму в имитационной модели.

Применительно к модели системы электроснабжения это означает, что решение любой новой задачи при наличии модели можно будет получить в минимальные сроки, составив простейшую дополнительную подпрограмму.

Для обеспечения правильного чередования событий в ЭВМ организуются так называемые системные часы, которые в заданных единицах времени Δt хранят значения времени (системное время).

При моделировании на ЭВМ могут быть два способа задания времени: с фиксированным интервалом (фиксированный шаг) или с переменным (шаг до следующего события). Это относится и к обновлению состояния объекта (системному времени). Само моделирование протекает с фиксированным шагом.

Для модели системы электроснабжения с непрерывным обновлением состояния, по-видимому, удобен фиксированный шаг. При этом некоторую информацию (например, токи локомотивов) можно вводить с переменным шагом.

К основным агрегатам при моделировании системы электроснабжения можно отнести часть системы внешнего электроснабжения, силовую часть тяговых подстанций, защиты, систему регулирования мощности и напряжения, тяговую сеть, устройства компенсации, вольтодобавочные устройства, э. п. с., агрегат, моделирующий график движения поездов (последовательность выпуска поездов разных категорий и весов, время хода их между раздельными пунктами, длительности стоянок, межпоездные интервалы на всех остановочных пунктах).

Возможности и результаты моделирования, как указывалось, зависят от используемых технических средств. В качестве них могут применяться аналоговые машины, обладающие высокой скоростью, а также ЭВМ, обладающие высокой точностью. Наиболее перспективными считаются гибридные вычислительные системы.

В связи с развитием микропроцессорных систем гибридные машины особенно перспективны. Возможность создания аналоговых блоков на цифровом принципе делает равноточными обе системы.

В полной мере оценить свойства этих машин можно будет после изучения последующих глав. Здесь же дано только наименование типов машин и констатация их свойств.

При статистическом моделировании определение воздействия случайно изменяющейся нагрузки на какой-либо элемент (трансфор-

матор, выпрямитель, провод) производится за определенный перио времени. Этот период надо выбрать так, чтобы погрешность расчет лежала в заданном (доверительном) интервале с вероятностью н менее заданной (доверительной).

В результате эксперимента на имитационной модели, выполнен ного для какого-то периода времени (как правило, не превышающег сутки), определится конкретная функция изменения рассматриваемо го параметра во времени. Эта конкретная функция называется реа лизацией случайного процесса. Чем больше случайных реализаци будет рассмотрено при моделировании, тем выше вероятность того что погрешность расчета не выйдет за установленный доверительны интервал, т. е. статистическая устойчивость рассчитываемого пара метра (максимальной температуры какого-либо элемента, степен теплового износа изоляции обмотки трансформатора и др.) буде достаточно высока.

Специальные приемы определения необходимого числа реализа ций по заданным доверительным интервалу и вероятности рассматри вать не будем, поскольку эти вопросы изучаются в курсах теории вероятностей и математической статистики.

В настоящее время работы по моделированию ведутся почти во всех транспортных учебных и научных институтах. Важнейшей задачей на перспективу является создание единой имитационной системы. Комплексное решение возникающей при этом проблемы возможно, если моделирующие системы имеют в основе единую формальную математическую схему, на которую должны быть ориентированы алгоритмы и программы, составляющие внешнее математическое обеспечение имитационной системы.

Создание такой имитационной системы позволит решать с ее помощью круг вопросов, который сейчас трудно ограничить. В связи с этим следует отметить широкие возможности, которые дает имитационное моделирование при проверке системы электроснабжения и э. п. с. на чувствительность (по какому-либо критерию) к изменению тех или иных параметров их работы.

Использование имитационной системы, моделирующей систему электроснабжения электрических железных дорог, не ограничивается только проектными или эксплуатационными расчетами. Она может служить базой для проведения близких к натурным математических экспериментов при испытании устройств или режимов управления.

1.4. Алгоритм

В вычислительной технике под алгоритмом понимают точную последовательность операций по переработке исходных данных с целью получения результата при решении конкретной задачи.

Эта последовательность (предписание) должна быть настолько строго формализована, что действуя по ней, вычислитель, имея исход-

ные данные (не вникая в существо задачи и ее физический смысл), придет после выполнения некоторого числа операций к правильному решению.

Рассмотрим примеры алгоритмов решения некоторых задач.

Пример 1. Требуется определить изменение скорости движущегося по подъему поезда. Исходными данными для решения задачи являются: масса поезда m; скорость движения поезда в рассматриваемый момент v_0; основное сопротивление движению поезда — W_0^*, сила тяги электровоза F_0^*, подъем i_{00}; ускорение силы тяжести g.

Решение этой задачи можно выполнить в следующем порядке:

1. Определить силу сопротивления движению поезда от подъема

$$W_i = gm/1000.$$

2. Найти значение полного сопротивления движению поезда

$$W_п = W_0 + W_i.$$

3. Найти результирующую силу, действующую на поезд,

$$F_р = F_0 - W_п.$$

4. Определить дальнейший ход решения задачи в зависимости от значения $F_р$: если $F_р$ положительна, следует перейти к выполнению операций, начиная с п. 5; если $F_р$ отрицательна, надо перейти к выполнению операций, начиная с п. 7; если $F_р = 0$, перейти к выполнению операций, начиная с п. 9.

5. Найти ускорение $a = F_р/m$.

6. Записать решение «Скорость поезда увеличивается», ускорение равно a и перейти к п. 10.

7. Найти замедление $a = -F_р/m$.

8. Записать решение «Скорость поезда снижается», замедление равно a и перейти к п. 10.

9. Записать решение «Скорость поезда не меняется» и перейти к п. 10.

10. Записать «Решение задачи закончено».

Пример 2. Определить сопротивление n параллельно соединенных резисторов. В качестве исходных данных заданы сопротивления отдельных резисторов r_1, r_2, r_3, r_n.

Для решения задачи введем переменную s_i с пределами изменения индекса i от 0 до n. Далее составим следующий алгоритм решения задачи:.

1. Положим $s_0 = 0$.
2. Положим $i = 1$.
3. Найдем $g_i = 1/r_i$.
4. Найдем $s_i = s_{i-1} + g_i$.
5. Проверим соотношение $i < n$. Если оно выполняется, увеличить i на 1 и вернуться к п. 3. Если $i = n$, то перейти к п. 6.
6. Искомое сопротивление $r_{рез} = 1/s_i$.

Конец.

Из приведенных примеров видно, что по составленным алгоритмам решать задачи данного класса при любых исходных данных может любой человек, умеющий выполнять элементарные вычислительные операции. Другими словами, если для определенного класса задач составлен алгоритм их решения, то в дальнейшем любая конкретная задача может быть решена чисто механически без затрат умственной энергии. В этом заключается особая ценность алгоритма.

---

* Предполагаем, что W_0 и F_0 заранее определены по соответствующим зависимостям, как $W_0(v)$ и $F_0(v)$.

Это свойство алгоритмов позволяет организовать на ЭВМ решение очень сложных задач.

Любой алгоритм должен удовлетворять следующим трем требованиям:

обладать четкостью составляющих его указаний для однозначного понимания их исполнителем;

алгоритм должен быть применим не к одной какой-либо задаче, а к некоторому множеству задач, отличающихся значениями исходных данных;

алгоритм при его выполнении должен приводить к результату, являющемуся решением задачи.

Заметим, что данное в начале этого параграфа определение алгоритма не является математическим определением. Тем не менее оно позволяет успешно использовать его для решения практических задач. Для решения ряда математических задач оказалось необходимым строгое математическое определение понятия алгоритма, в связи с чем уже создана и продолжает разрабатываться специальная теория алгоритмов.

Контрольные вопросы

1. Что такое модель объекта исследования?
2. Что такое математические модели, в частности, аналитические?
3. В чем различие аналитических и имитационных моделей?
4. В каких случаях целесообразно применять имитационное моделирование?
5. Дайте определение алгоритма.
6. Приведите пример алгоритма решения какой-либо задачи.

Глава 2

ОБЩИЕ ПРИНЦИПЫ РАБОТЫ И ПОСТРОЕНИЯ ЦИФРОВЫХ ЭВМ

2.1. Общие сведения о типах ЭВМ и их классификация

В зависимости от способа представления информации для обработки ее с помощью ЭВМ вычислительные машины принято делить на два класса: *аналоговые* (АВМ) и *цифровые* (ЦВМ).

В АВМ величины, характеризующие вычисляемые процессы и над которыми производятся математические преобразования, изменяются во время решения непрерывно и представлены в машине обычно в форме непрерывно изменяющихся электрических напряжений.

В ЦВМ все величины, характеризующие вычисляемые процессы, представлены в виде электрических напряжений, имеющих дискретную (импульсную) форму и два уровня: максимальный и нулевой. Это соответствует двоичной форме представления чисел в ЦВМ.

Для дальнейшего деления этих двух классов на группы необходимо выбрать признак деления, который может отражать производительность и объем оперативной памяти, габариты, стоимость, назначение и т. д.

С развитием технологии изготовления элементов ЭВМ, появлением интегральных схем стираются грани между такими важными характеристиками, как производительность, емкость оперативной и внешней памяти, уменьшаются габариты и стоимость ЭВМ.

Поэтому классификацию ЭВМ целесообразно провести с точки зрения удобства применения разных их типов в той или иной сфере деятельности человека.

В настоящее время парк ЭВМ в СССР составляет ЭВМ третьего и четвертого поколения (первое поколение ЭВМ — ламповые, 1946 г.— середина 50-х годов; второе — полупроводниковые, середина 50-х—60-х годов; третье — на интегральных микросхемах разной степени интеграции, середина 60-х —70-х годов; четвертое — на больших интегральных схемах, середина 70-х годов — настоящее время).

Условно выделим две основные группы: большие и малые ЭВМ. Причем это деление подходит для обоих классов ЭВМ, но в дальнейшем будем рассматривать только цифровые ЭВМ, называя их компьютерами (от англ. COMPUTER).

Большие ЭВМ — это ЭВМ третьего и четвертого поколений, основное назначение которых связано с выполнением обработки и хранением больших объемов информации, проведением сложных расчетов в ходе решения научно-исследовательских задач. Работа на таких ЭВМ требует высокой квалификации и опыта. К ним относятся все ЭВМ единой серии ЕС.

Малые ЭВМ — это наиболее распространенный тип ЭВМ, который объединяет компьютеры как третьего, так и четвертого поколений. Общими чертами этих ЭВМ являются небольшие габариты и удобство эксплуатации. На этих ЭВМ могут работать после соответствующего обучения и неспециалисты по вычислительной технике. К ним относятся ЭВМ серии СМ (СМ-2, СМ-3, СМ-4, СМ-1420).

Среди малых ЭВМ выделяются подгруппы: мини-ЭВМ и микроЭВМ. *Мини-ЭВМ*— семейство ЭВМ, которые в основном служат средством коллективного пользования в научно-исследовательских подразделениях, лабораториях, на производстве, на участках и в автоматизированных линиях изготовления какой-либо продукции. Для их эксплуатации не требуется больших помещений, поскольку они состоят всего лишь из двух стоек и дисплея — устройства визуального отображения информации на электронно-лучевую трубку, которое при наличии клавиатуры ввода информации является устройством оперативного взаимодействия человека и ЭВМ.

На этих ЭВМ может работать специально обученный оператор, управляющий производственным процессом.

МикроЭВМ — это компьютеры, где в качестве элементной базы использована специальная интегральная микросхема-микропроцессор. Предназначены они для индивидуальной работы, хотя на них могут работать и несколько человек одновременно, и все оборудование занимает место в пределах конторского стола. Их могут использовать неспециалисты по вычислительной технике, имеющие некоторые основные знания по программированию и операторской работе. В СССР этот класс образует модели семейства «Электроника».

Среди микроЭВМ можно выделить персональные ЭВМ, домашние компьютеры, микрокалькуляторы и ЭВМ, встраиваемые (однокорпусные, одноплатные и однокристальные) в какой-либо технический объект.

Персональные ЭВМ (ПЭВМ) обладают рядом отличительных признаков, самым важным из которых является создание для человека удобных условий работы. Достигается это наличием в ЭВМ большого числа программ, обслуживающих процесс вычисления, а также программ, специально разработанных для пользователей. К персональным ЭВМ относятся ДВК-2, ДВК-2М, ДВК-4, «Искра 226», ЕС-1840, «Искра 1030».

Домашние компьютеры — микроЭВМ, которые в качестве периферийных устройств ввода-вывода и отображения информации используют бытовые приборы: магнитофон, телевизор. Для работы на них необходимы минимальные знания, которые затем пополняются при работе на ЭВМ с обучающими программами. В СССР выпускаются домашние компьютеры серий БК0010, БК0011, «Микроша» и др.

Микрокалькуляторы предназначены для небольших расчетов и удобны в эксплуатации главным образом из-за малых габаритов. Для работы на программируемых микрокалькуляторах необходимо

знать принципы программирования на машинно-ориентированном языке, примененном в данном типе калькулятора.

МикроЭВМ, встраиваемые в какой-либо технический объект, выпускаются промышленностью без периферийного оборудования и оснащаются им в зависимости от требуемых условий работы. К этим микроЭВМ относятся различного рода микропроцессорные контроллеры (одноплатный К1-20), однокристальные микроЭВМ (КМ1816ВЕ48), агрегативные средства („Электроника 60").

Для повышения надежности и производительности вычислительных машин их объединяют в вычислительные системы с одновременно работающими несколькими однородными или неоднородными (по типу) ЭВМ. Они могут дублировать друг друга, могут находиться в резерве, могут параллельно решать разные задачи или, разбив одну из задач на части, обрабатывать эти части тоже параллельно. Пользователи подключаются к вычислительным системам через каналы связи и имеют на рабочем месте дисплей. С помощью него осуществляется ввод и вывод информации.

По принципу организации своей структуры вычислительные системы подразделяются на многомашинные и мультипроцессорные (многопроцессорные).

Основной задачей *мультипроцессорной системы* является решение задач, поддающихся расчленению на несколько задач, параллельно. За счет этого существенно сокращается время решения задачи. Параллельное решение задач эффективно при применении большого числа матричных и векторных операций. В тех случаях когда задача не поддается распараллеливанию, применяется другой принцип организации систем — конвейерный. При этом одна задача разбивается на ряд элементарных частей, каждая из которых решается на своем процессоре. Каждый процессор вступает в работу после окончания работы предыдущего и выполняет только определенную функцию. Управляющая программа может настраивать для решения каждой задачи требуемый набор процессоров, т. е. каждая задача образует свою структуру вычислительной системы. Так возникает понятие виртуального, т. е. условного компьютера, структура которого определяется структурой задачи. К мультипроцессорной системе отечественного производства относится система «Эльбрус». Она объединяет до 10 одинаковых универсальных процессоров (кроме вспомогательных) и позволяет получить производительность от 1,2 млн. до 12 млн. операций в секунду.

Мультипроцессорные системы строятся внутри семейства мини-ЭВМ СМ, которые ориентированы на работу в определенной области техники (управление технологическими процессами, обработка данных в сейсморазведке и т. п.).

С появлением и широким внедрением персональных ЭВМ для расширения их вычислительных возможностей и производительности ПЭВМ можно объединять в вычислительные сети.

2.2. Цифровые ЭВМ. Программы. Схемы алгоритмов

Электронно-вычислительная машина представляет собой комплекс взаимосвязанных устройств, предназначенных для выполнения вычислительных операций под управлением программы, преобразующей исходные данные в результате решения задачи.

Обрабатываемые в ЭВМ данные представлены в виде двоичных кодов, преобразованных в электрические сигналы, таким образом, она представляет собой электронную *цифровую вычислительную машину*.

Совокупность электронных схем, обрабатывающих поступающие данные, называют процессором. Если все эти обрабатывающие схемы размещены в одной интегральной схеме, то последнюю называют микропроцессором. Дополнив его запоминающим устройством и устройствами ввода-вывода информации, получают микроЭВМ.

Когда употребляются слова «информация» или «данные», то имеются в виду сведения, которые вводятся в ЭВМ. Данные могут быть представлены с помощью букв, цифр, различных символов и помещены в основную память ЭВМ.

Программа состоит из совокупности команд. Команды, которые могут выполняться на ЭВМ, составляют ее *систему команд*. Программа, которая должна быть выполнена, и обрабатываемые данные помещаются в основную память. Данные и программа преобразуются в совокупность электрических сигналов с помощью устройства ввода.

Для контроля взаимодействия между различными частями ЭВМ служит устройство управления *УУ* (рис. 2.1). Для временного хранения данных *УУ* может использовать набор регистров. Один из регистров называется регистром команд и служит для размещения кода выполняемой команды, выбираемой из основной памяти. Команда, находящаяся в регистре команд, сообщает устройству управления следующую информацию:

адрес ячейки памяти, в которой находятся данные;

место в памяти, куда эти данные должны быть помещены;

действие, которое должно быть выполнено над данными.

После выполнения команды счетчик команд, имеющийся в устройстве управления, выдает адрес следующей выбираемой из памяти команды.

Для того чтобы устройство управления могло выполнять команды, предусмотрено арифметико-логическое устройство *АЛУ*. Оно может выполнять арифметические операции (сложение, вычитание) и логические операции (обрабатывать коды чисел с помощью логических операций И, ИЛИ). В распоряжении *АЛУ* имеется также вспомогательная память в виде накапливающего регистра, который принято называть аккумулятором. *АЛУ* в соответствии с указанием *УУ* обрабатывает данные о команде, находящейся в регистре команд, и результат помещает в аккумулятор.

В больших ЭВМ, которые используются для обработки данных и для научных расчетов, устройство управления, арифметико-логическое устройство и память принято относить *к центральному процессору* (вычислителю) В микроЭВМ роль памяти выполняют связанные с *УУ* и *АЛУ* регистры.

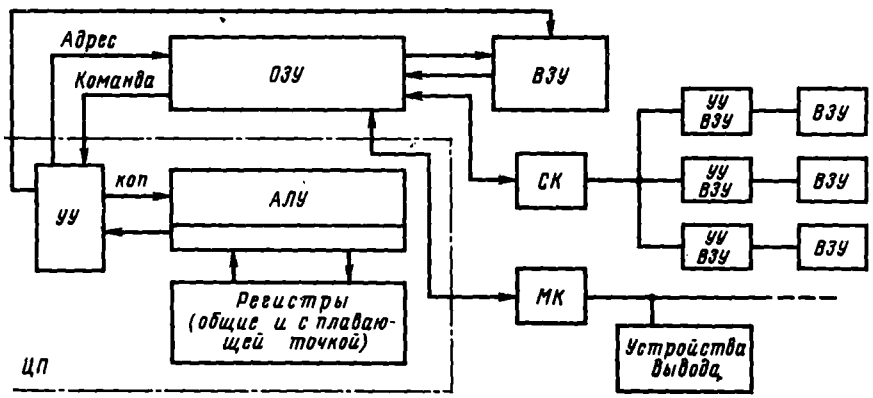
Для представления результатов обработки данных в удобной для восприятия человеком форме служат устройства вывода данных. По команде ЭВМ или оператора результаты обработки выводятся из ЭВМ на экран дисплея или с помощью печатающего устройства на бумагу.

Память ЭВМ служит для хранения команд программы обработки данных и самих данных. Память вычислительной машины состоит из основной, внешней и вспомогательной, образуемой несколькими регистрами.

Основная память служит для хранения команд, образующих программу, и обрабатываемых данных. В соответствии с этим основная память делится на память программ и данных. Последняя получила название оперативного запоминающего устройства (*ОЗУ*).

Если требуется организовать хранение данных и программ с целью их последующего использования, то применяется внешнее запоминающее устройство — накопители на магнитной ленте или дисках.

Команды и данные, расположенные в основной памяти, должны быть легкодоступны. Память состоит из блоков одинакового размера, называемых словами или *ячейками памяти*. Ячейка памяти состоит из элементов памяти. В каждом элементе может храниться одно двоичное число (1 или 0). Совокупность нулей и единиц, находящихся в элементах памяти, представляет собой содержимое ячейки памяти. Каждое слово (ячейка) памяти имеет свой *адрес*. Например, в большинстве микроЭВМ максимальное число адресов составляет $2^{16}=$ $=65\,536$, или 2^{32}. Этим числом ограничивается размер основной



ис. 2.1. Структурная схема цифровой ЭВМ

памяти, так как адрес описывается шестнадцатиразрядным двоичным словом.

Для передачи информации в другие устройства и записи информации, поступающей из других устройств, производится считывание слов из ячеек. При этом содержимое ячеек не меняется, а при необходимости слово может быть снова взято из той же ячейки. При записи в эту ячейку другого слова хранившееся в ней слово стирается.

Для того чтобы ЭВМ могла производить обработку информации, в ее память необходимо загрузить программу, по которой ЭВМ автоматически производит требуемые операции над данными, т. е. обрабатывает их.

Для обеспечения высокой производительности больших ЭВМ в центральную ЭВМ вводят два устройства, называемых мультиплексным *МК* и селекторным каналами *СК*. По своей структуре — это специализированные процессоры, которые управляют вводом или выводом информации, обеспечивают связь ЭВМ с внешней памятью и устройствами ввода-вывода информации.

Мультиплексный канал обеспечивает передачу информации для внешних устройств, которые характеризуются низкой скоростью работы (устройства ввода-вывода с перфокарт, перфолент, печатающие устройства, графопостроители, клавиатуры дисплеев). Мультиплексный канал одновременно обслуживает несколько параллельно работающих устройств, так как по сравнению с ними имеет более высокое быстродействие.

Селекторный канал обеспечивает ввод и вывод информации внешних запоминающих устройств *ВЗУ*, которые имеют большую скорость передачи. К таким устройствам относятся накопители на магнитных дисках, магнитной ленте. Селекторный канал работает в монопольном режиме и обслуживает только одно устройство через устройство управления им *УУ*.

Некоторые технические характеристики основных моделей ЕС ЭВМ отечественного производства представлены в табл. 2.1.

Из сравнения табличных данных можно заключить, что производительность «старших» моделей повысилась на несколько порядков и составляет для мощных ЭВМ 5 млн. операций в секунду; емкость оперативной памяти увеличилась также на несколько порядков и составляет у мощных ЭВМ несколько мегабайт.

Автоматическое управление процессом решения задачи достигается на основе *принципа программного управления*, являющегося основным при построении ЭВМ. Согласно этому принципу весь вычислительный процесс организуется на основе выполнения ЭВМ ряда команд, заложенных в памяти ЭВМ.

Другим важнейшим принципом является *принцип хранимой в памяти программы*. Согласно этому принципу программа, закодированная в цифровом виде, хранится в памяти наравне с числами. В командах указываются не сами участвующие в операциях числа, а адреса ячеек основной памяти, в которых они находятся, и адрес

Таблица 2.1

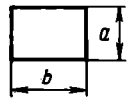
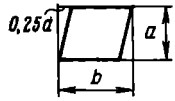
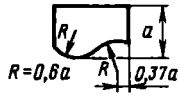
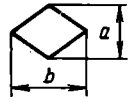
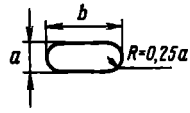
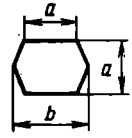
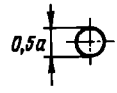
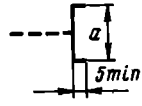
Модель	Производительность, тыс. операций в секунду	Емкость оперативной памяти, Кбайт
	Первая очередь «Ряд 1»	
ЕС 1020	10	64—256
ЕС 1030	50	256—1024
ЕС 1050	510	512—1024
	Модернизированная очередь «Ряд 1»	
ЕС 1022	80	128—512
ЕС 1033	200	256—512
ЕС 1052	700	1000—8000
	Вторая очередь «Ряд 2»	
ЕС 1035	125—200	512
ЕС 1045	650—860	1024—4000
ЕС 1065	5000	16324
	Третья очередь «Ряд 3»	
ЕС 1036	400	2000—4000
ЕС 1046	1300	4000—8000
ЕС 1066	5000	8000—16000

ячейки, куда помещается результат операции. Так как команды представляются в машине в форме чисел, то над командами, как над числами, машина может производить операции (модификация команд). Такой прием широко применяется при программировании задач по обработке информации.

Чтобы запрограммировать ЭВМ для выполнения определенной задачи, необходимо формально описать последовательность выполняемых операций, которые приводят к ее решению. Такое формальное описание называется *схемой алгоритма*. Схема алгоритма в графической форме изображает основную идею построения схемы вычислений. Для изображения схем алгоритмов используются специальные графические символы (табл. 2.2).

Для удобства описания и установления связей между отдельными частями (блоками) схемы алгоритма блоки помечаются путем указания сквозной нумерации на них. Последовательность выполнения пунктов алгоритма, описываемого схемой, устанавливается путем упорядоченного размещения блоков на схеме и объединения их линиями, которые называются линиями потока информации. Основными направлениями для линий потока информации являются направления сверху вниз и слева направо. По отношению к блоку линии потока могут быть входящими и выходящими. Число входящих линий для блока принципиально не ограничено. Выходящая линия может быть только одна. Исключение составляют логические блоки сравнения, имеющие не менее двух выходящих линий потока, каждая из которых соответствует одному из возможных исходов про-

Таблица 2.2

Наименование	Обозначение	Отображаемая функция
Блок вычислений		Выполнение действий, в результате которых изменяются значения данных
Блок ввода-вывода данных		Ввод-вывод данных или результатов независимо от носителя информации
Документ		Вывод данных или результатов на бумажный носитель информации (на печатающее устройство)
Блок сравнения		Проверка условия и принятие решения
Подпрограмма		Предварительно определенный процесс, например подпрограмма
Пуск-останов		Начало и конец алгоритма
Модификация		Выполнение действий, изменяющих пункты алгоритма
Соединитель		Связующее звено или соединитель (используется для связи частей схемы, размещенных на той же самой странице; внутри кружка проставляется номер)
Межстраничный соединитель		Соединитель частей схемы, размещенных на разных страницах
Комментарий		Примечания, дополнения, комментарии к различным блокам схемы алгоритма. Направленные соединения в схеме
Соединения		Направленные соединения в схеме

верки логического условия. При большом числе пересекающихся линий, многократных изменений их направлений допускается разрывать линии потока информации, размещая на обоих концах разрыва специальные символы (см. табл. 2.2). При разработке сложных систем алгоритмов необходимо составлять несколько схем различных уровней детализации. Схема первого уровня отображает весь алгоритм целиком. Схема второго уровня раскрывает логику отдельных блоков схемы первого уровня и т. д. В качестве примера рассмотрим схему алгоритма поиска максимума и минимума в последовательности чисел.

По структуре принято различать линейные алгоритмы с последовательно расположенными блоками, разветвляющиеся алгоритмы, в которых присутствуют один или несколько блоков сравнения, и циклические алгоритмы, обеспечивающие получение результата путем многократного повторения некоторой последовательности действий.

На рис. 2.2 представлен сложный алгоритм, включающий в себя все названные типы. Последовательно выполняемые действия изображаются блоками *1, 2, 10*. Блоки *3—9* можно считать одним сложным блоком и включить в эту последовательную структуру.

Действия, изображаемые блоками *4, 5, 6, 7*, представляют разветвленный алгоритм сравнения каждого i-го числа с числами, находящимися в ячейках M или Q. В зависимости от результата сравнения происходит разветвление, т. е. выполнение действий в блоке *5* (сравниваемое число больше находящегося в ячейке M) или *7* (сравниваемое число меньше находящегося в ячейке Q). Поскольку действия *4, 5, 6, 7* должны быть выполнены для каждого из N чисел, то эти блоки составляют тело (ядро) циклического алгоритма, изображаемого блоком *3* (начало цикла — присваивание переменной цикла i — начального числового значения), блоком *8* (изменение значения переменной цикла для последующего повторения действий *4, 5, 6, 7* с новым номером числа $i=i+1$), блоком *9* (проверка условия для окончания или продолжения циклических действий). Условие в данном алгоритме определяется заданным количеством проверяемых чисел N. При $i>N$ количество циклических действий будет превышать количество заданных чисел и обработку необходимо закончить, т. е. перейти к блоку *10*.

Циклические алгоритмы являются наиболее распространенными. Практически любой сложный алгоритм состоит из нескольких вложенных друг в друга циклических алгоритмов, в каждом из которых тело цикла может содержать несколько последовательно расположенных или вложенных друг в друга циклических алгоритмов.

Общий порядок следования основных частей цикла (начало цикла, тело цикла, изменение номера числа повторений цикла, проверка условия окончания цикла), не обязательно должен быть таким, как рассмотренный выше. Например, проверка условия окончания цикла может предшествовать всем остальным действиям. Так, итера-

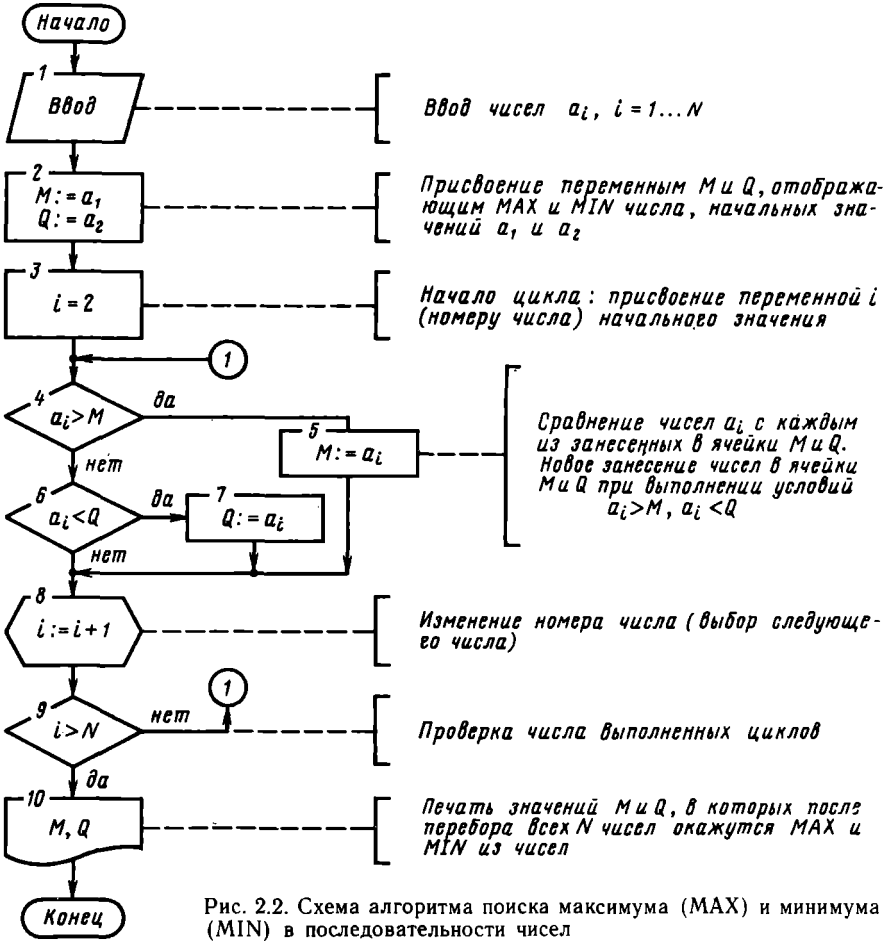


Рис. 2.2. Схема алгоритма поиска максимума (MAX) и минимума (MIN) в последовательности чисел

ционные циклы (в которых заранее неизвестно число повторений) имеют аналогичную структуру, только выход из цикла осуществляется не после того, как цикл повторится заданное число раз, а при выполнении более общего условия, связанного с проверкой значения монотонно изменяющейся в цикле величины. Очень часто эта величина характеризует точность, достигнутую на очередном шаге итерационного процесса, реализуемого алгоритмом.

Рассмотренные типы структур являются готовыми формами, с помощью которых описываются все конкретные процессы обработки информации.

Эти формы составляют основу структурированной формы представления алгоритмов [3].

2.3. Представление информации и чисел в цифровых ЭВМ. Системы счисления. Арифметические операции в цифровых ЭВМ

Вычислительная машина является устройством обработки информации. *Информация* — это любые сведения о различных явлениях, происходящих в общественной жизни, природе, а также характеризующие состояние некоторого технического объекта.

Информация, зафиксированная на некотором носителе информации, называется *сообщением*. Сообщение — это форма представления информации. Семантика сообщения оценивает такие его свойства, как содержательность, целесообразность, ценность, полезность информации. Обмен информацией в автоматизированных системах происходит при помощи сигналов.

Сигнал — это физически изменяющаяся величина, соответствующая передаваемому сообщению; сигналы могут быть электрическими (токи, напряжения и т. п.), световыми, ультразвуковыми и т. п. Носителями сигналов являются физические величины (электрические токи, напряжения и т. п.). Определяющие параметры передаваемых сигналов в виде временных функций (частота, амплитуда, фаза, длительность и последовательность импульсов) называются информационными параметрами в том случае, когда посредством этих параметров передается информация.

Различают *аналоговые сигналы* (информационные параметры внутри заданного диапазона принимают любые значения) и *дискретные сигналы* (информационные параметры могут принимать лишь определенные дискретные значения). Дискретные сигналы подразделяются на цифровые и многопозиционные.

Цифровые сигналы могут передаваться в основном последовательно или параллельно. У параллельных сигналов все информационные параметры передаются по n различным сигнальным линиям. У последовательных сигналов все n информационных параметров следуют по общей сигнальной линии один за другим в определенной последовательности.

В ЦВМ применяется двухпозиционный (двоичный) сигнал, у которого информационным параметром является уровень сигнала, принимающий во времени два значения, соответствующие нулю и единице. Такие сигналы легко получать технически, имея устройство с двумя устойчивыми состояниями (триггер).

Относительная простота технического осуществления двухуровневых сигналов и определила применение двоичной системы счисления.

Система счисления — способ представления чисел посредством цифровых знаков. Всякая система счисления характеризуется *основанием* — количеством цифр, используемых для записи чисел. Так, десятичная система счисления использует для записи чисел десять цифр (от 0 до 9). Эта система счисления — позиционная, т. е. значе-

ние цифры (ее вес) зависит от положения (позиции) в числе, например:

$$\begin{array}{r} 347 = 3\times 100 = 300 \\ +4\times 10 = 40 \\ +7\times 1 = 7 \\ \hline 347 \end{array}$$

Значение каждой позиции есть позиционный коэффициент. Каждый из них должен быть умножен на степень основания системы счисления, которая называется весом.

Любое число x можно выразить в следующей обобщенной форме, которая представляет разложение числа по степеням (весам) основания q системы счисления:

$$x = a_n q^n + a_{n-1} q^{n-1} + \ldots + a_0 q^0 + a_{-1} q^{-1} + \ldots + a_{-m} q^{-m}$$

Здесь a_i — цифры системы счисления.

Представим число в самых распространенных системах счисления: в десятичной, двоичной, восьмеричной и шестнадцатеричной

$$132 = 1\cdot 10^2 + 3\cdot 10^1 + 2\cdot 10^0 = 1\cdot 2^7 + 0\cdot 2^6 + 0\cdot 2^5 + 0\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 0\cdot 2^0 = 2\cdot 8^2 + 0\cdot 8^1 + 4\cdot 8^0 = 8\cdot 16^1 + 4\cdot 16^0$$

или $132_{10} = 10000100_2 = 204_8 = 84_{16}$.

Из сравнения видно, что трехзначное десятичное число представляется восьмизначным двоичным и двузначным шестнадцатеричным. Большая громоздкость двоичной системы делает ее неприемлемой для использования людьми, а тем более при программировании. Поэтому большинство программистов пользуются восьмеричной или шестнадцатеричной системой счисления, оставляя самой ЭВМ задачу преобразования таких чисел в их двоичный эквивалент. Преобразование двоичных чисел в восьмеричные и шестнадцатеричные легко выполняются. Для этого необходимо разбить двоичное число на группы по три цифры (если преобразуем в восьмеричный эквивалент) или по четыре цифры (если преобразуем в шестнадцатеричный эквивалент), начиная с наименьшей значащей цифры, а затем заменить каждую двоичную группу соответствующим ей числом. Например, если имеется двоичное число 010011100001, то его восьмеричный и шестнадцатеричные эквиваленты:

$$\begin{array}{llll} 010 & 011 & 100 & 001_2 = 2341_8 \\ 0100 & 1110 & 0001_2 = 4E1_{16} \end{array}$$

Для записи чисел в шестнадцатеричной системе используются цифры от 0 до 9 и буквы латинского алфавита от A до F (табл. 2.3).

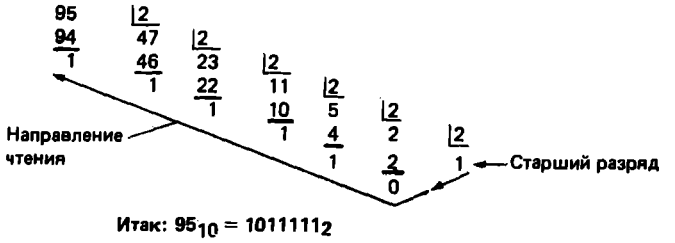
Для перевода целого числа из одной системы счисления в другую его нужно последовательно делить на основание той системы счисления, в которую оно переводится. Деление ведется в той системе, из которой осуществляется перевод, и до тех пор, пока частное (остаток

Таблица 2.3

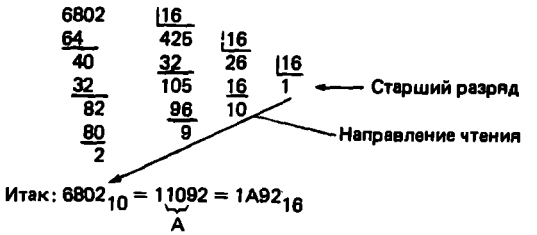
Десятичная система	Двоичная система	Шестнадцатеричная система	Десятичная система	Двоичная система	Шестнадцатеричная система
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

от деления) не будет меньше основания, на которое число делят. Число в новой системе счисления запишется в виде последнего частного и всех остатков деления, начиная с последнего. Последний остаток (частное) дает старший разряд (цифру) числа.

Перевод из десятичной в двоичную систему числа 95_{10}:



Перевод из десятичной в шестнадцатеричную систему числа 6802_{10}:



Для перевода правильной дроби (или дробной части числа) из одной позиционной системы счисления в другую ее надо последовательно умножить на основание той системы, в которую она переводится. Перемножаются только дробные части числа. Дробь в новой системе запишется в виде целых частей получаемых произведений, начиная с первого сомножителя. Переведем из десятичной в двоичную систему дробь $0{,}3125_{10}$:

Итак: $0{,}3125_{10} = 0{,}0101_2$

При переводе смешанных чисел отдельно переводят целую и дробную части.

Арифметические действия в двоичной системе (табл. 2.4) чрезвычайно просты и, следовательно, легко реализуются в вычислительных машинах. Арифметические действия в двоичной системе счисления выполняются так же, как и в десятичной. Но если в десятичной переносится и занимается сразу по десять единиц, то в двоичной — только по две единицы, т. е. по величине, равной основанию системы счисления.

В цифровых ЭВМ используют естественную (с фиксированной запятой) и нормальную (с плавающей запятой) формы представления чисел.

В естественной форме число представляется в виде целой и отделенной от нее запятой дробной части числа. Для каждой части числа отводится строго определенное количество разрядов.

При нормальной форме число представляется в виде

$$N = mg^p,$$

где $0{,}1 \leqslant m < 1$ — мантисса числа; p — порядок числа; q — основание системы счисления.

Порядок p определяет положение запятой в числе и показывает, на сколько разрядов влево (в случае отрицательного порядка) или вправо (в случае положительного порядка) нужно перенести запятую в мантиссе, чтобы получить число в естественной форме.

В нормальной форме запись содержит знак порядка числа, порядок (без нуля целых и запятой), знак и величину мантиссы. Знак «плюс» изображается нулем, а «минус»— единицей.

Преимуществом нормальной формы является более широкий диапазон представления чисел. Однако эта форма требует дополнитель-

Таблица 2.4

Сложение	Вычитание	Умножение	Деление
$0+0=0$	$0-0=0$	$0\cdot0=0$	$0/0=?$
$0+1=1$	$1-0=1$	$0\cdot1=0$	$0/1=0$
$1+0=1$	$1-1=0$	$1\cdot0=0$	$1/0\to\infty$
$1+1=10$	$10-1=1$	$1\cdot1=1$	$1/1=1$

ных разрядов для хранения порядка числа и усложняет арифметические действия над числами с плавающей запятой, так как этому предшествует операция выравнивания порядков, что увеличивает время выполнения арифметических действий. Для машинного выполнения арифметических действий все числа представляются в виде комбинации электрических сигналов, отображающих двоичные коды чисел.

Код — набор правил, раскрывающих способ представления данных (чисел). Часто используется в значении «язык». Например, термин в «машинном коде» означает «на машинном языке».

В цифровой ЭВМ все арифметические операции сводятся практически к двум: сложению и вычитанию в комбинации с такой специфической операцией над кодами, как поразрядный сдвиг, которая широко используется при умножении и делении.

Операция вычитания осуществляется с помощью специальных кодов.

Один из таких кодов — обратный код. *Обратный код* $\overline{N}$ числа есть разность, получаемая от вычитания основного числа N из некоторой константы C: $\overline{N}=C-N$, где $C=q-q^m$, m — разрядность мантиссы числа, представленного в нормальной форме.

Для двоичной системы счисления ($q=2$, $m=5$) получим $C=2-2^5=10{,}00000_2-0{,}00001_2=1{,}11111_2$. С использованием обратных кодов разность получают сложением уменьшаемого с обратным кодом вычитаемого.

Обратный код в двоичной системе счисления отличается от самого числа заменой единиц на нули и нулей на единицы. В электрических устройствах памяти с двумя устойчивыми состояниями (триггерах) имеются два противоположных по знаку выхода — прямой и инверсный. С последнего снимается обратный код, поэтому специальной операции нахождения обратного кода производить не требуется. Операция умножения в цифровой ЭВМ выполняется многократным сложением со сдвигом, операция деления выполняется умножением на число, обратное делителю, или многократным вычитанием делителя из делимого со сдвигом.

Например, требуется перемножить два числа в двоичном коде 1001 и 1011

```
   1001
 x
   1011
 ------
   1001
  1001    <-- сдвиг влево на один разряд
 0000     <--
1001      <--
-------
1100011   <-- сложение
```

Кроме арифметических операций, вычислительная машина выполняет и логические операции, которые лежат в основе решения задач

управления. Логические операции выполняются по правилам булевой алгебры.

Известны три основные логические функции или операции: ИЛИ, И, НЕ.

Операция ИЛИ является логическим сложением и поэтому может обозначаться знаком «+». Ее называют также дизъюнкцией и обозначают знаком $\vee$ Следует, однако, различать смысл знаков, применяемых для обозначения логических функций и одинаковых с арифметическими по форме написания знаков.

Логическая единица означает факт свершения какого-либо события, а нуль означает, что это событие не произошло.

Связь между логическими переменными и их функцией обычно выражается в виде таблицы истинности (рис. 2.3). Из таблицы истинности видно, что функция ИЛИ (рис. 2.3, *а*) принимает значение нуль только тогда, когда все переменные равны нулю. Элементы, реализующие логические функции, имеют специальные графические изображения (см. внизу на рис. 2.3).

Операция И — это операция логического умножения, поэтому она часто обозначается так же, как произведение в элементарной алгебре, ее называют также конъюнкцией и обозначают знаками $\wedge$ или &.

Из таблицы (рис. 2.3, *б*) следует, что функция И принимает значение единицы только при единичном значении переменных.

Операция НЕ называется логическим отрицанием или инверсией. Эта операция обозначается чертой над обозначением переменной. На схемах логический элемент НЕ изображается так, как показано на рис. 2.3, *в*.

На основе этих трех основных логических функций строятся более сложные, которые затем преобразовываются и упрощаются. Это необходимо для того, чтобы при аппаратной реализации логических функций сократить объем необходимых элементов. Очень распространенной является арифметическая операция, реализующая арифметическую операцию сложения двух двоичных чисел (рис. 2.3, *г*). Обычно устройство арифметического сложения должно осу-

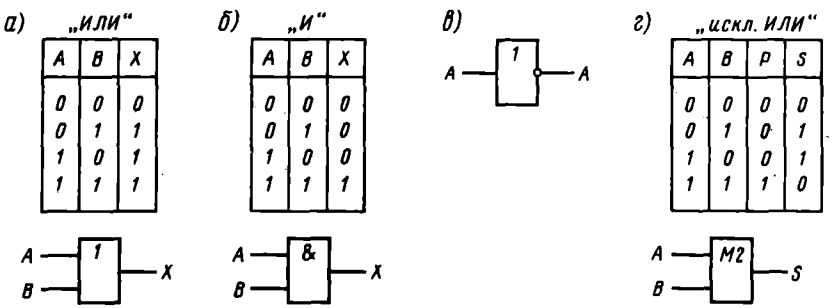


Рис. 2.3. Таблицы истинности и графические изображения логических элементов

ществлять следующие операции: $0+0=0$; $0+1=1$; $1+0=1$; $1+1=10$, где 10 означает 2 в двоичной системе счисления.

Первый столбец аналогичен операции И и дает перенос из данного разряда в следующий (вспомогательная переменная p). Столбец s (сумма) дает значение логической функции, называемой «исключающее ИЛИ», которая в отличие от операции ИЛИ означает «только A или только B». Эта операция является не основной, а производной и эквивалентна арифметическому сложению двух двоичных чисел. Результат ее называется «суммой по модулю два» и записывается в следующей форме: $s=A\oplus B=AVB$. Это выражение дает сумму двух двоичных чисел и бит переноса в следующий разряд, однако не учитывает перенос, полученный в предыдущем разряде.

2.4. Кодирование алфавитно-цифровой информации. Единицы информации. Адресация памяти ЭВМ

С помощью двоичных знаков можно кодировать и хранить в памяти машины любую алфавитно-цифровую информацию. Для этого каждому символу необходимо присвоить свой код. Запись десятичных цифр и алфавитных символов в виде двоичных символов называется *кодированием*. С помощью кода, состоящего из n разрядов, можно представить 2^n различных символов. Следовательно, число двоичных разрядов, используемых для кодирования, зависит от числа кодируемых символов.

Кодирование алфавитно-цифровой информации необходимо в связи с вводом в машину и хранением в ее памяти текстов программ, написанных на различных алгоритмических языках, использующих для записи цифры, буквы и некоторые специальные символы; применением ЭВМ для решения статистических планово-экономических задач, требующих обработки как числовой, так и текстовой информации. Для кодирования информации в машинах серии ЕС используется восьмиразрядный двоичный код ДКОИ [10, 20], в машинах серии СМ ЭВМ — семиразрядный код КОИ-7.

Каждый символ в коде ДКОИ представляется восьмиразрядным числом, требующим для своего размещения в памяти ЭВМ восемь информационных разрядов. Наименьший адресуемый элемент, состоящий из восьми информационных разрядов, называется байтом. Несколько последовательных байтов, используемых для хранения кодов, образуют ячейку памяти ЭВМ. Один байт состоит из восьми битов; бит — двоичная цифра 0 или 1, или двоичный разряд. Для оценки больших объемов информации, емкости памяти ЭВМ используются единицы измерения 1 Кбайт, равный 1024 байтам, или 1 Мбайт, равный 1024 Кбайтам.

Определенная последовательность битов называется словом независимо от того, имеет ли оно какой-либо смысл или нет. Обычно слово определяет, сколько битов могут сразу (параллельно) содержаться

в арифметических устройствах, устройствах памяти и других блоках ЭВМ.

Центральный процессор ЭВМ обладает возможностью временно хранить несколько слов, полученных им из памяти. Эти слова хранятся в специальных элементах, называемых регистрами. Работа процессора возможна только при наличии различных регистров в его составе.

В зависимости от того, в каком регистре находится группа битов, изменяется смысл содержащейся в ней информации. В памяти могут находиться *команды*, подлежащие выполнению процессором, а также *операнды*, или данные, т. е. исходная информация, подлежащая обработке. Отделение команд от данных осуществляется именно центральным процессором. Для этого команды направляются в один из регистров — регистр команд (РК), а данные направляются в регистр данных. Регистр команд функционально входит в устройство управления, а регистр данных — это один из регистров арифметико-логического устройства (АЛУ). Регистр команд сохраняет бинарное (двоичное) слово команды, согласно которой данный процессор действует. Это слово называется текущей командой.

Имеется еще и счетчик команд (РС), функция которого — образовывать бинарное слово, представляющее в памяти адрес следующей команды. Обычно счетчик команд имеет инкрементное устройство, которое добавляет единицу к текущему адресу и таким образом получается следующий адрес. По этому адресу в памяти содержится либо операнд, либо следующая команда программы.

Адрес представляет двоичное число, равное длине машинного слова. Максимальная емкость памяти определяется числом, которое может представить машинное слово. Например, при восьмиразрядном слове емкость памяти равна 256 ячейкам, при 16-разрядном слове — 64 Кбайтам. Дальнейшее увеличение емкости памяти требует расширения адресной части команды. В малоразрядных микропроцессорах для этого приходится использовать два, три и более машинных слов (байтов) в составе команды.

Совокупность ячеек, к которым может обращаться процессор, называется *адресным пространством*, или *полем памяти*. Адресное пространство графически изображается в виде столбца из нескольких строк, равного числу фактически имеющихся ячеек памяти с адресами от 0000 до 1111. Нумерация ячеек производится обычно сверху вниз.

Программа для микропроцессорной системы создается в виде последовательности команд. Программа должна находиться в ячейках адресного пространства в машинных кодах. Каждая команда представляет длинную последовательность нулей и единиц. Программирование и последующий ручной ввод этой программы в память представляют утомительную операцию. Значительно облегчает эту задачу использование восьмеричной и шестнадцатеричной систем кодирования.

Восьмеричная (октальная) и шестнадцатеричная (гексадецимальная, HEX) системы кодирования применяются для упрощения восприятия длинных последовательностей нулей и единиц, представляющих двоичные числа. Для бинарного кодирования символов какого-либо алфавита требуется различное число бит. Например, для кодирования числа от 0 до 9 требуются четыре бинарных разряда (четыре бита). Именно поэтому практически во всех микропроцессорах длина машинного слова кратна четырем (8, 12, 16 и т. д.). Для кодирования алфавитных символов недостаточно четырех, пяти и даже шести бинарных разрядов, так как это дает соответственно 16, 32, 64 различные кодовые комбинации. Для кодирования букв русского алфавита (33) и латинского (26) необходимо семибитное кодирование, дающее 128 комбинаций. Плодотворной оказалась идея использовать в ЭВМ машинное слово длиной из восьми разрядов (1 байт).

Четыре бинарных разряда позволяют получить 16 различных кодовых комбинаций, из которых используется только 10 (от 0000 до 1001), остальные как десятичные числа не имеют смысла. Однако шестнадцатеричный код широко применяется для символьного кодирования программ для микропроцессоров. Кодовые комбинации, соответствующие числам 10 и более, условно обозначаются буквами латинского алфавита (см. параграф 2.3). Чтобы не выписывать длинную последовательность нулей и единиц в программах, записывают их шестнадцатеричный эквивалент. Например, операция 1111 1001 1100 0101 в шестнадцатеричном коде запишется как F9C5. Возможность записи и чтения трех или четырех битов как одного символа представляет огромное преимущество для адресации и программирования.

Для ввода-вывода информации, в частности при аналого-цифровом преобразовании, бинарный код неудобен, поскольку требует либо большого числа линий связи (при параллельной передаче), либо длительного цикла обмена (при последовательной передаче). В основном по этой причине в большинстве автоматизированных систем для ввода-вывода информации применяют бинарный (двоично кодированную десятичную систему БКД) или двоично-десятичный код. Иными словами, используется алфавит из десяти цифр: 1, 2, 3, ..., 9, 0, каждая из которых закодирована четырьмя двоичными разрядами. При записи многоразрядных чисел сохраняются традиционные десятки, сотни и т. д., но каждый десятичный разряд по-прежнему кодируется четырьмя двоичными разрядами, например, 1987 = = 0001 1001 1000 0111. Как видно из примера, двоично-десятичный код менее экономичен, чем простой двоичный, но, несмотря на это, он более удобен, так как вводить исходную информацию и получать ее можно в привычной для человека десятичной системе счислений.

Обычно применяют взвешенные БКД. Это значит, что каждому из четырех битов присваивается определенная характеристика, представляющая собой значение каждого разряда БКД, равное целой

степени основания системы счисления (т. е. в данном случае двух) для соответствующей позиции разряда. Эту характеристику разряда кратко называют весом разряда. Простой двоичный код является кодом с естественными весами разрядов, так как значение каждого бита определяется последовательностью чисел 0, 1, 2, 3, отсчитываемых справа налево, т. е. 2^3, 2^2, 2^1, 2^0. Иными словами, такой код имеет веса 8—4—2—1 и поэтому он кратко называется «код 8421». Имеются коды с другими весами.

Когда в программах микропроцессорных систем встречаются числовые данные, например значения операндов, то они могут быть представлены в любой форме. Чтобы не возникло путаницы, запись в бинарной (двоичной) системе кодирования принято сопровождать буквой B. Восьмеричную систему идентифицируют буквой Q. После десятичной цифры ставят букву D. Шестнадцатеричное число, которое начинается с цифр 0—9, должно завершаться символом H или HEX. Шестнадцатеричные числа, начинающиеся с символов A — F, обязательно дополняются слева нулем. Например, двоичное число 1010 11111 должно быть записано в виде 0AFH.

2.5. Программное обеспечение ЭВМ. Язык Ассемблера

Все средства ЭВМ принято делить на две основные категории: аппаратные и программные [4, 5, 16].

Аппаратные средства — это разнообразные механические, электромеханические, магнитные и электронные элементы, из которых состоит машина. *К программным средствам*, называемым также программным обеспечением (ПО), относят все программы, совместимые с машиной и предназначенные для данной машины.

Обычно в ПО включают также документацию, руководства и описания, языки программирования с соответствующими трансляторами (специальные программы). Промежуточная категория, называемая аппаратно-программными средствами, включает в себя «материализованные» программы, т. е. ПЗУ, в которых хранятся программы и управляющие микропрограммы.

Известно, что команды программы и данные хранятся в оперативной памяти и закодированы в двоичной форме. Следовательно, естественный (внутренний) машинный язык есть язык двоичных нулей и единиц. Программа, представленная в терминах этого языка, называется *машинной*, или *объектной программой*. При всей простоте машинного языка он совершенно не устраивает пользователей, поэтому сейчас разработано множество разнообразных внешних языков, на которых ведется программирование и общение с ЭВМ.

Соответствующие исходные программы пользователей приходится переводить (транслировать) на машинный язык. Как бы ни усложнялись и ни изменялись внешние языки, действиями аппаратных средств управляют только машинные команды. Поскольку современ-

ные ЭВМ представляют собой сложное сочетание аппаратных и программных средств, то использованию машины как инструмента для решения конкретных задач помогает поставляемое вместе с машиной программное обеспечение. Оно превращает ЭВМ в удобное средство для пользователей.

Различают три класса программных средств: базовое программное обеспечение, операционные системы, прикладное программное обеспечение.

Базовое ПО состоит из набора автономных программ и подпрограмм, предназначенных главным образом для связи с пользователем и автоматизации его труда при разработке прикладных программ.

Так, например, редактор текста упрощает и ускоряет создание и изменение исходных программ, представленных в виде последовательности символов. Пользователь взаимодействует с редактором посредством удобных содержательных приказов, по которым машина вводит или удаляет строки, символы, а также ищет символьные цепочки.

Язык Ассемблера (язык символического кодирования) по существу эквивалентен машинному языку и явно отражает структуру ЭВМ. Однако все элементы программ при этом представляются в символической форме.

Ассемблерные программы оказываются наиболее эффективными, и пользователь может непосредственно управлять вводом-выводом информации.

В зависимости от конфигурации (набора и объема средств ЭВМ) могут быть реализованы три разновидности Ассемблеров: базовый Ассемблер, рассчитанный на минимальную конфигурацию, расширенный Ассемблер и макроАссемблер.

Ассемблер базовый занимает малый объем памяти (4 Кбайт) и выдает закодированное сообщение об ошибках. Расширенный и макроАссемблер занимают объем памяти (8 Кбайт) и выдают подробные сообщения об ошибках. В макроАссемблере имеются сложные команды (макрокоманды).

Программа-загрузчик предназначена для записи объектных программ в основную память (из устройства ввода) и запуска их на исполнение. Абсолютный (двоичный) загрузчик помещает программу в фиксированную область памяти, а перемещающий загрузчик — в произвольную область памяти.

Редактор связей реализует функцию объединения нескольких независимо ассемблируемых программ в единый загрузочный модуль.

Программа отладки (отладчик) представляет собой удобное средство отладки (локализации и удаления ошибок) прикладных программ.

Наиболее эффективным оказывается диалоговый (интерактивный) режим отладки.

В составе базового ПО могут быть трансляторы с языков программирования высокого уровня. Они различаются по числу проходов исходных программ, по требуемой минимальной конфигурации и степени стандартизации. Минимальный объем оперативной памяти в зависимости от возможностей языка составляет 8—16 Кбайт.

Язык программирования высокого уровня — это алгоритмический язык для записи программ независимо от типа ЭВМ. Этот язык состоит из операторов (обобщенных команд), у которых, как правило, имеется несколько слов в одной команде.

В мини-ЭВМ и микроЭВМ обычно реализуются языки высокого уровня Бейсик и Фокал, а в последних моделях Фортран, Паскаль, СИ, АДА и др.

Различают два типа трансляторов — компиляторы и интерпретаторы. Первые целиком считывают текст программы и переводят его в машинные коды. Программу в кодах, как правило, записывают на какой-либо внешний носитель информации. Затем для выполнения программы ее с этого носителя загружают в память ЭВМ.

Интерпретатор поочередно «просматривает» строки программы в том порядке, в котором они должны выполняться, и вызывает нужные подпрограммы, входящие в состав интерпретатора. Подпрограммы сразу выполняются ЭВМ. Таким образом в памяти постоянно должны находиться интерпретатор и исходный текст прикладной программы. Интерпретатор транслирует программу постоянно при каждом ее выполнении, и время этой трансляции входит во время работы прикладной программы.

Так как компилятор транслирует исходный текст программы один раз — при переводе в машинные коды, то скорость выполнения прикладных программ при этом во много раз больше, чем при интерпретаторе. Подавляющее большинство языков высокого уровня рассчитано на компилятор, а самым распространенным интерпретирующим языком является Бейсик.

В состав базового ПО входят также диагностические программы. Имеется комплекс программ для проверки процессора, оперативной памяти и всех периферийных устройств. Обычно имеется тест-программа общей проверки машины, а также специализированные программы для локализации неисправного блока. Для управления программами базового ПО предусматривается сравнительно простая программа — монитор.

Операционные системы (ОС) представляют собой набор программ (системных), которые, кроме рассмотренного базового программного обеспечения, осуществляют эффективное управление ресурсами ЭВМ. К главным ресурсам машины относят процессор, основную память и периферийные устройства. Основными функциями ОС являются: планирование работы процессора, распределение и защита памяти, управление периферийными устройствами, обработка внутренних и внешних прерываний, управление дан-

ными и библиотеками программ, загрузка и выполнение прикладных программ, связь с пользователем.

Желательно, чтобы ОС занимала как можно меньше оперативной памяти, поэтому ее программы хранятся в накопителе на магнитном диске и передаются в оперативную память, как только в этом возникает необходимость. Однако некоторая часть ОС, которая управляет функциями самой ОС, постоянно находится в оперативной памяти. Эта часть называется ядром, исполнительной программой, организующей программой или супервизором.

Прикладное программное обеспечение создается пользователем или для пользователя и определяется требованиями конкретного применения. В некоторых случаях прикладное ПО поставляется вместе с ЭВМ.

В зависимости от имеющихся средств автоматизации программирования разработка прикладных программ ведется на исходном языке одного из трех уровней: машинном языке, языке Ассемблера и языке высокого уровня.

Машинный язык и Ассемблер явно отражают аппаратные средства и обеспечивают максимальные возможности непосредственного управления работой ЭВМ. Оба эти языка низкого уровня оказываются машинно-зависимыми (машинно-ориентированными), а составленные на них программы обладают наименьшей мобильностью в смысле применимости к различным ЭВМ.

Языки высокого уровня (алгоритмические языки) почти не отражают структуру ЭВМ и считаются машинно-независимыми. Программы на этих языках характеризуются высокой мобильностью. В принципе любой алгоритм можно запрограммировать на машинном языке в двоичной или шестнадцатеричной формах. Однако этот язык на практике почти не применяется, за исключением тех ситуаций, когда нет других средств для автоматизации программирования.

Язык Ассемблера отличается от машинного языка тем, что операции, операнды и адреса представляются в символической форме, более удобной для пользователя.

Это значительно упрощает составление программы, позволяет сравнительно просто вносить в нее изменения производить отладку программы.

Программа-транслятор с языка Ассемблера называется ассемблирующей программой или, короче, Ассемблером. Если Ассемблер реализован на ЭВМ, которая может выполнять полученную объектную программу, его называют (естественным) резидентным Ассемблером.

Ассемблер, реализованный на ЭВМ, которая не может выполнять объектную программу, называется кросс-Ассемблером. Кросс-Ассемблеры для микроЭВМ и микропроцессоров обычно реализуются на мини-ЭВМ и в сетях ЭВМ с разделением времени.

Сформированные кросс-Ассемблерами объектные программы выводятся на некоторый носитель и загружаются с него в память ЭВМ, для которой они предназначены.

Выше было сказано, что Ассемблер в некоторой степени зависит от типа ЭВМ и поэтому программы, написанные на языке Ассемблера для машин серии ЕС ЭВМ и СМ, будут отличаться по кодам, написанию команд и др. Подробно описание Ассемблера для ЕС ЭВМ можно найти в специальной литературе, например [4, 6, 7]. Были сделаны попытки стандартизировать язык Ассемблера для микропроцессоров и микроЭВМ, однако и в нем имеются некоторые отличия для разных типов ЭВМ.

Рассмотрим один из вариантов языка Ассемблера применительно к программированию микропроцессоров КР580. В настоящем разделе приведены общие положения языка и правила оформления бланков для написания текста программы. Система команд микропроцессора будет описана отдельно. Как и любой алгоритмический язык, он состоит из серии команд, выполнение которых ЭВМ приводит к получению определенных результатов в преобразовании числовых кодов и в конечном итоге к решению каких-либо задач.

Команда языка Ассемблера — это последовательность буквенно-цифровых символов (обычно находящихся на одной строке), состоящая из трех полей: метки, кода операции и операндов (см. параграф 2. 8.). Поле метки используется не всегда. Набор букв и цифр, записанных в этом поле, называется меткой или именем. Метка представляет собой символический адрес какой-либо области памяти. Благодаря применению меток программисту при написании программы не надо помнить все используемые адреса.

В пределах одной программы нельзя использовать одну и ту же метку (в поле метки командных слов) несколько раз.

В поле кода операции записывают мнемоническое обозначение операции, которое однозначно соответствует команде. Как правило, эта запись представляет собой сокращенное название операции на английском языке (см. параграф 2.7). В этом же поле записываются мнемонические обозначения операций, резервирующих память для хранения одиночных констант (постоянных чисел) и массивов констант (см. пример 4, параграфа 2.8). Поле операндов содержит название или номера регистров, значения данных, адреса и метки, определяющие адреса, и другую информацию, служащую для указания операндов.

Комментарий Ассемблерной команды начинается после точки с запятой и служит для пояснения команды. Текст комментария полностью игнорируется при Ассемблировании (переводе с Ассемблера в объектный код), но печатается в листинге (текст на бумаге) программы. Желание пользователя экономить память (каждый символ — байт!) не должно приводить к сокращению ком-

ментариев в ущерб пониманию программы другими. Комментарии ради удобства даются малыми русскими буквами.

Кроме команд, которые предназначены для выполнения заданного пользователем алгоритма, имеются директивы (вспомогательные команды) или, как их называют, «псевдокоманды».

Директивы сообщают программе-Ассемблеру информацию, необходимую для формирования фиксированных таблиц, определения имен, резервирования памяти для переменных, размещенных в памяти программ и подпрограмм и задания формата листинга. Хотя мнемоники директив и появляются в поле кода, директивы не соответствуют машинным командам. Метки и комментарии в директивах необязательны, а содержимое поля операнда зависит от их функции.

Рассмотрим некоторые наиболее широко употребляемые директивы.

ORG — предписывает Ассемблирующей программе место расположения в области адресов памяти первой команды программы. Эта информация необходима Ассемблирующей программе, чтобы разместить программу в памяти и присвоить действительные адреса символическим меткам.

Символические имена и соответствующие им двоичные коды записываются, используя директиву EQV (присвоить).

например, MPGP: EQV 21H

Если Ассемблирующая программа встретит в тексте прикладной программы это символическое имя, она подставит вместо него соответствующее число.

Псевдооператоры DB и DW используются для записи в память восьми- и шестнадцатиразрядных констант соответственно.

Константы могут задаваться в виде чисел, символических имен и выражений, разделенных запятыми. Численные константы представляются в различных системах счисления, что указывается специальными обозначениями — дескрипторами. Например, в Ассемблере микропроцессора КР580 система счисления определяется буквой, завершающей число: H — шестнадцатеричная, O (или Q) — восьмеричная, D — десятичная, B — двоичная. Отсутствие дескриптора означает десятичное число. Шестнадцатеричные числа должны начинаться с цифр 0 — 9.

Для записи в память алфавитно-цифровых символов в поле операндов можно записать требуемую последовательность символов, заключив их в апострофы.

Псевдооператор DS резервирует ячейки памяти для хранения данных во время работы программы. Число ячеек указывается в поле операндов, их содержимое перед началом работы программы не определяется. END — псевдооператор записывается в конце программы.

При трансляции Ассемблер производит автоматическое распределение кодов программ и данных в памяти. Для этого дваж-

ды «просматривается» исходный текст. Сначала определяются значения всех символических имен, используемых в программе, т. е. составляется таблица, содержащая их имена, соответствующие им значения и адреса памяти. Затем вычисляются значения операндов и генерируются машинные коды.

2.6. Основные принципы работы микропроцессорной системы

Элементом, производящим обработку данных в микропроцессорных системах, является микропроцессор, выполненный в виде большой интегральной схемы (БИС) [7, 8, 9, 20].

Микропроцессор и ряд других вспомогательных схем, обеспечивающих его работу и всей системы в целом, образуют так называемый микропроцессорный модуль (рис. 2.4), к которому с помощью системных шин подключают периферийные модули.

Системные шины представляют набор соединительных проводников — линий, объединяющих определенные выводы всех периферийных модулей. По каждой линии может быть передано значение одного разряда двоичного кода в виде уровней напряжений +0,3 или +2,4 В, что соответствует логическому нулю или логической единице.

По роду передаваемой информации линии могут образовывать шину данных, шину адресов и шину управления.

Периферийными модулями являются различные запоминающие устройства и регистры для подключения внешних устройств (клавиатуры различных датчиков и исполнительных механизмов).

Регистр — совокупность триггеров, предназначенных для хранения двоичного числа определенной длины (например, восьмиразрядного). Регистры, входящие в устройства ввода-вывода МП, называются *портами ввода-вывода*. Микропроцессор *МП* серии КР580ИК80А предназначен для обработки восьмиразрядных двоичных чисел (слов или байтов), поэтому порты ввода-вывода тоже должны быть восьмиразрядными. Запоминающие устройства состоят их набора восьмиразрядных ячеек памяти. Обмен данными между микропроцессором и периферийными модулями происходит по шине данных, состоящей из восьми линий, обозначаемых D0—D7. По линии Д0 передается младший бит, по линии D7 — старший. Особенностью шины данных является ее двунаправленность, которая обозначает возможность передачи данных в разные моменты времени в различных направлениях, т. е. от микропроцессора или к нему. Эта возможность обеспечивается трехстабильными шинными формирователями, через которые периферийные модули подключаются к шинам данных. Входы формирователя, кроме состояний 0 и 1, могут принимать третье пассивное, или так называемое высокоимпедансное состояние, благодаря чему они оказываются отключенными от линий данных.

При работе модуль должен обмениваться данными с определенными ячейками памяти или портами. Для этого каждая ячейка памяти и порт ввода или вывода должны иметь индивидуальные номера — адреса. При обмене данными процессорный модуль устанавливает на адресной шине двоичный код, соответствующий адресу ячейки или порта. Число линий адресной шины определяется разрядностью адресной шины микропроцессора КР580ИК80А и равно 16. Это позволяет обращаться к $2^{16}=64536$ ячейкам памяти.

Обычно ЗУ состоит из одной или нескольких БИС памяти, каждая из которых имеет вход для приема сигнала ВМ (выбор модуля).

Расшифровка кода на адресной шине позволяет выбрать определенную БИС ЗУ с помощью соответствующего сигнала ВМ. Обращение к определенной ячейке памяти внутри БИС ЗУ происходит по сигналу с выхода внутреннего дешифратора, входы которого (адресные входы БИС ЗУ) подключаются к соответствующим линиям шины адресов. Микропроцессор позволяет подключить к шинам адресов до 256 портов ввода и до 256 портов вывода. Все входы *ВМ* портов ввода-вывода подключаются через схемы дешифраторов к восьми младшим разрядам адресной шины. Порты включаются в работу при появлении на шине адресов кодов, соответствующих их номерам. Дополнительным условием работы пор-

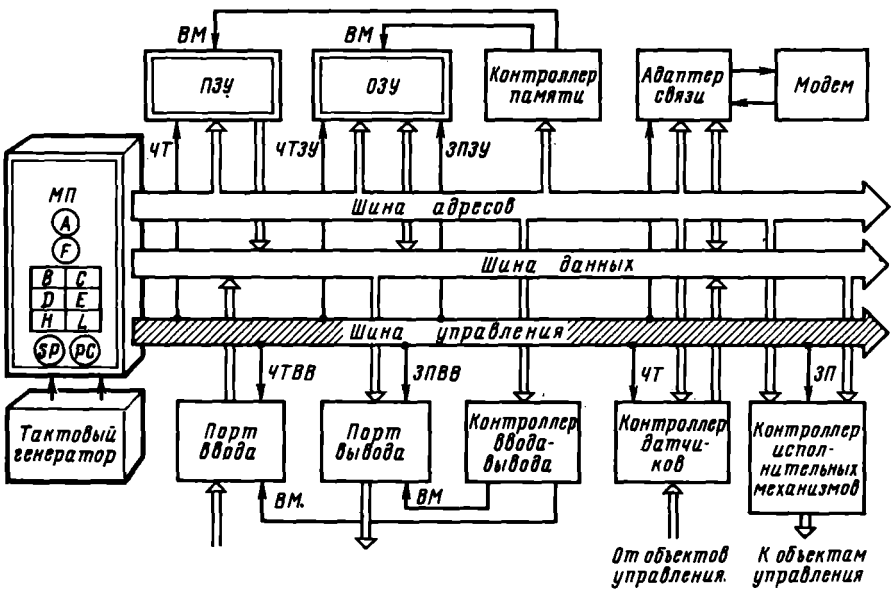


Рис. 2.4. Микропроцессорный модуль, внешние устройства и система ввода-вывода сигналов

та является наличие соответствующего сигнала на шине управления. По линиям шины управления от процессорного модуля к периферийным поступают сигналы выбора группы модулей (порты или модули памяти) и направления обмена данными: сигнал чтения из модулей запоминающих устройств *ЧТЗУ*, сигнал записи в модули запоминающих устройств *ЗПЗУ*, сигнал записи данных в порт вывода данных *ЗПВВ*, сигнал чтения из порта ввода *ЧТВВ* (см. рис. 2.4).

В микропроцессорной системе заданный алгоритм реализуется при выполнении программы, хранимой в ЗУ в виде последовательности команд. При этом исходными данными для программы являются данные, вводимые через порты ввода от датчиков или клавиатуры.

Промежуточные результаты вычислений хранятся в ЗУ или внутренних регистрах микропроцессора, а полученные результаты выводятся через порты вывода.

Программа хранится в постоянном запоминающем устройстве *ПЗУ* модуля *ЗУ*. *ПЗУ* — это БИС памяти, в которую необходимая информация (программа, константы) записывается в процессе изготовления микропроцессорной системы. Информацию из *ПЗУ* можно только считывать, и она не теряется при выключении электрического питания системы.

Промежуточные данные, результаты вычислений хранятся в оперативном запоминающем устройстве *ОЗУ*, в которое они могут и записываться, и считываться в процессе работы. При выключении питания данные в *ОЗУ* не сохраняются. Обычно *ПЗУ* имеет большой объем памяти и в нем записана небольшая программа-загрузчик, под управлением которой в начале работы системы в *ОЗУ* с какого-либо внешнего устройства (например, клавиатуры) загружается рабочая программа.

Выполнение любой команды микропроцессора начинается с чтения ее кода операции из *ЗУ* (в нем хранится программа). Для этого процессорный модуль устанавливает на адресных шинах код адреса ячейки памяти, в которой записан код операции команды, а на соответствующей линии шины управления — сигнал *ЧТЗУ*. В результате код операции команды выдается из ячейки памяти на шину данных и считывается процессорным модулем. Микропроцессор расшифровывает (декодирует) код операции, определяет необходимые и соответствующие этому коду действия и выполняет их. В процессе выполнения этой команды микропроцессор может неоднократно обращаться к памяти *ОЗУ* и *ПЗУ* для чтения и записи данных.

После выполнения текущей команды выполняется следующая, которая хранится в *ЗУ*, и т. д.

Микропроцессор имеет сложную внутреннюю структуру, но с точки зрения программиста его можно рассматривать состоящим только из семи восьмиразрядных регистров *A, B, C, D, E, H, L*

(см. рис. 2.4), регистра признаков результата выполнения операции F и двух шестнадцатиразрядных регистров *SP* и *PC*.

Регистр *A* называют *аккумулятором* и используют для хранения операнда, с которым работает арифметико-логическое устройство (АЛУ) микропроцессора. Результат работы АЛУ по окончании обработки данных вновь помещается в регистр *A*.

Шесть регистров *B*, *C*, *D*, *E*, *H*, *L* предназначены для хранения промежуточных данных. При исполнении некоторых команд регистры *B* и *C*, *D* и *E*, *H* и *L* объединяются в *регистровые пары* для хранения шестнадцатиразрядных данных. Назначение регистра признаков *F* будет рассмотрено при описании команд.

Для изучения системы команд и написания программ необходимо знать, как происходит формирование кода на шине адресов [8].

При обращении к памяти для чтения кода очередной команды из микропроцессора на шину адресов поступает содержимое шестнадцатиразрядного регистра *PC*, называемого *счетчиком команд*. В этом регистре к моменту окончания выполнения текущей команды всегда подготавливается адрес очередной команды программы. Во время выполнения программы микропроцессору необходимо обращаться к определенным ячейкам памяти для чтения и записи промежуточных данных. Это делается с помощью команд с *непосредственной адресацией*. Они имеют трехбайтовый формат, т. е. каждая команда занимает три последовательно расположенные в памяти ячейки. В первом байте команды хранится код операции, а во втором и третьем записан шестнадцатиразрядный адрес обращения к памяти. При выполнении такой команды микропроцессор последовательно считывает значения второго и третьего байтов во внутренние буферные регистры и затем при обращении к памяти для записи и чтения данных передает из этих регистров на шину адресов шестнадцатиразрядный адрес. Команды с непосредственной адресацией выполняются медленно, так как микропроцессору при их выполнении приходится дважды обращаться к памяти для побайтного чтения кода адреса.

При одно- и двухбайтовых командах, использующих *косвенную регистровую адресацию*, адрес требуемой ячейки памяти поступает на шину адресов из регистровых пар *BC*, *DE* или *HL*.

Кроме этого, возможна адресация к ячейкам памяти по содержимому шестнадцатиразрядного регистра *SP*, называемого указателем стека (рис. 2.5). Под стеком в микропроцессоре понимается любая область *ОЗУ*, служащая для хранения адресов констант и промежуточных данных, адресация к ячейкам которой осуществляется при помощи указателя стека *SP*. При обращении к ячейке *ЗУ*, расположенной в стековой области, на шину адресов засылается содержимое регистра *SP*. Перед выполнением команд, использующих регистр *SP*, в него должен быть предварительно записан код начала стековой области *ОЗУ* (код «верха» стека). С помощью стековых команд, образующих сте-

ковую адресацию, в стек можно переслать шестнадцатиразрядное число из любой регистровой пары или регистра счетчика команд *PC*.

Запись числа в память происходит побайтно: сначала записывается старший байт в ячейку с адресом, на единицу меньшим содержимого указателя стека (т. е. в ячейку с адресом $SP-1$), затем младший байт в ячейку с адресом $SP-2$. Таким образом по окончании записи содержимое указателя стека становится равным $SP-2$. Часто вместо выражения «содержимое указателя стека» используется выражение «положение указателя стека». В этом случае можно сказать, что указатель стека «смещается вниз» на две ячейки в сторону младших адресов памяти при занесении в стек содержимого регистровых пар или счетчика команд.

Система команд микропроцессора позволяет извлекать из стека его содержимое в любую регистровую пару или в счетчик команд PC. При этом сначала переписывается во внутренний регистр микропроцессора младший байт из ячейки памяти, затем в другой регистр регистровой пары переписывается старший байт из ячейки памяти с адресом $SP+1$. После выполнения команды указатель стека принимает значение $SP+2$, т. е. указатель стека оказывается автоматически «смещенным вверх» на две ячейки в сторону старших адресов памяти.

Достоинством команд с адресацией по указателю стека является то, что программист может не заботиться каждый раз о конкретных адресах ячеек памяти, куда записываются и откуда считываются данные. Необходимо соблюдать только определенную последовательность при записи данных в стек и их извлечении, т. е. читать данные из стека в последовательности, обратной той, которая была при записи. При этом говорят, что при работе со стеком используется принцип «последним пришел — первым вышел».

В любой момент времени в стек можно включить дополнительную информацию, но при извлечении первой всегда будет та, которая включена последней.

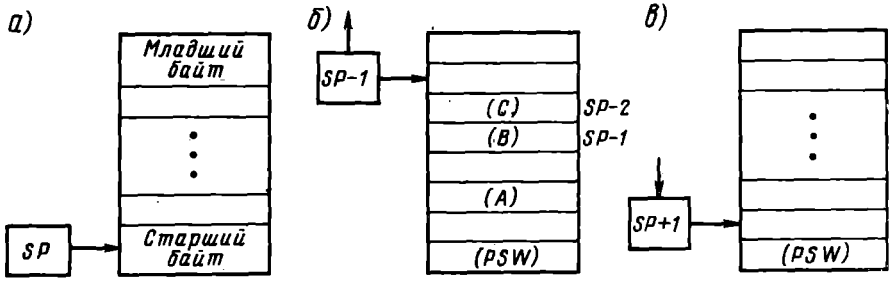


Рис. 2.5. Исходное состояние (*а*), загрузка (*б*) и выгрузка (*в*) стека

Для записи и хранения шестнадцатиразрядных чисел в памяти и внутренних регистрах восьмиразрядного микропроцессора можно использовать три регистровые пары — *BC*, *DE*, *HL*, указатель стека *SP* и счетчик команд PC. При этом в регистрах *B*, *D*, *H* регистровых команд хранятся старшие байты чисел, а в регистрах *C*, *E*, *L* — их младшие байты. В операциях со стеком как шестнадцатиразрядное число рассматривается также совокупность регистра *A* (старший байт) и регистра признаков (младший байт), именуемая *PSW* (см. рис. 2. 5). Для хранения в памяти шестнадцатиразрядному числу всегда отводятся две смежные ячейки. Запись чисел в эти ячейки происходит побайтно, причем в ячейку с меньшим адресом записывается младший байт, а в ячейку с бо́льшим адресом — старший байт числа.

2.7. Система команд микропроцессора КР580ИК80

Каждый микропроцессор характеризуется определенной *системой команд*, т. е. полным перечнем элементарных действий, которые способен производить микропроцессор. Управляемый этими командами микропроцессор выполняет очень простые действия, такие, как элементарные арифметические и логические операции, операции пересылки данных, сравнения двух величин и др. Представив алгоритм любой сложности в виде последовательности таких элементарных действий, можно написать программу выполнения этого алгоритма микропроцессором.

По формату (числу отведенных для нее разрядов) команды микропроцессора делятся на одно-, двух- и трехбайтовые. Байты команды последовательно друг за другом располагаются соответственно в одной, двух или трех ячейках запоминающего устройства микропроцессорной системы. Первый байт любой команды определяет *код операции*. Он представляет формат команды и те действия, которые должны быть произведены микропроцессором над данными (операндами) и в процессе ее выполнения.

Для успешного написания программ для микропроцессора необходимо хорошо знать весь перечень команд и представлять те действия, которые будут выполняться микропроцессором при выполнении каждой из них.

Код операции любой команды в ЗУ микропроцессорной системы представлен двоичным восьмиразрядным числом. Всего двоичным кодом можно представить $2^8 = 256$ различных комбинаций. Почти столько же команд имеет и микропроцессор (некоторые комбинации двоичных чисел не используются и поэтому команд несколько меньше).

Запомнить более 200 кодов команд, представленных в виде двоичных восьмиразрядных чисел, т. е. в виде набора единиц и нулей, почти невозможно. Поэтому каждому коду команды ставится

в соответствие мнемоническое название команды, которая является сокращением английских слов, описывающих ее действие. Мнемонический код команд позволяет легче запомнить их функции и облегчает написание программ.

Система команд микропроцессора КР580ИК80 может быть разбита на пять групп [8, 10]:

команды пересылок данных из памяти в микропроцессор;

команды арифметических операций, такие, как «сложить», «вычесть»;

команды логических операций над байтами;

команды перехода в заданную область памяти системы;

специальные команды ввода-вывода, сдвига содержимого регистра (аккумулятора), операций с шестнадцатибитными словами.

Полная система команд микропроцессора КР580ИК80 с мнемоническими обозначениями, позволяющими перейти к кодам, выраженным в шестнадцатеричной системе счисления, показана в табл. 2.5.

После мнемоники кода операций для двухбайтовых команд записывается восьмиразрядный операнд, обозначенный через D8, а для трехбайтовых команд — шестнадцатиразрядный адрес ячейки памяти или операнд, обозначаемый соответственно ADR и D16. Через M обозначается ячейка памяти, адресуемая в соответствии с описанием команды.

Группа команд пересылок — наиболее многочисленная группа. С их помощью происходит обмен данными между внутренними регистрами и ячейками памяти.

В команде сначала определяют приемник данных, а затем источник данных. Если источник или приемник данных в команде является ячейкой памяти M, то ее адрес всегда содержится в регистрах H и *L* блока регистров общего назначения микропроцессора. Старший байт всегда находится в регистре H, а младший — в регистре *L*.

Команда MOV R1, R (MOVe) содержит любые внутренние восьмиразрядные регистры микропроцессора R1 и R. При выполнении этой команды содержимое регистра R пересылается в регистр R1, причем в регистре R сохраняется прежнее значение данных. В качестве R1 и R могут быть определены ячейка памяти, регистр или регистры. Например, в команде MOV A, C символ A означает аккумулятор, C — регистр в микропроцессоре, а сама команда означает, что содержимое регистра C должно быть помещено в аккумулятор. Прежнее содержимое аккумулятора разрушается. Рассмотрим команду MOV M, C, где M — ячейка памяти, адрес которой хранится в регистровой паре H и L, куда он должен быть предварительно занесен. Эта запись означает, что содержимое регистра C должно быть передано в ячейку, адрес которой указан в регистрах H и L. Используются два восьмиразрядных регистра, поскольку адрес любой ячейки памяти занимает 16 разрядов.

Команда MVI R, D8 (MOVE Immediate) отличается от предыдущей команды тем, что в качестве источника данных используется восьмибитная константа, которая следует непосредственно за кодом операции. Приемником данных является или регистр, или ячейка памяти.

Например, запись MVI A, 02H означает, что двоичное число, шестнадцатеричный эквивалент которого 02, должно быть передано в аккумулятор. Можно записать эту команду и так: MVI A, 00000001B, при этом буква B указывает, что константа выражена в двоичной форме.

Для пересылок данных между аккумулятором и ячейками памяти в качестве адреса ячейки памяти может быть использовано также содержимое регистровых пар BC и DE. Тогда для записи в память данных из аккумулятора используются однобайтовые команды STAX B или STAX D, а при обратной пересылке LDAX B или LDAX D.

В командах STAX B (STore, Accumulatore eXtended addressing) и LDAX B (LoaD Accumulatore eXtended adderessing) последние два слова указывают на совместное использование двух регистров и поэтому в команду добавляется символ X. Адресация производится в соответствии с номерами регистров, но указывается один из них.

Команда STAX B означает, что надо запомнить содержимое аккумулятора в ячейке памяти, адрес которой хранится в регистровой паре BC. Команда LDAX D предписывает загрузить в аккумулятор содержимое ячейки памяти, адрес которой записан в регистровой паре DE.

Команды STA ADR и LDA ADR служат для записи данных из аккумулятора в ячейку памяти и обратной пересылки.

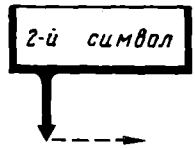
Адрес ячейки задается непосредственно; символами ADR обозначен двухбайтовый операнд. Адрес ячейки находится во втором и третьем байтах команды.

Команды LXI B, D16; LXI D, D16; LXI H, D16 (Load eXtended Immediate) служат для непосредственной записи в соответствующие регистровые пары шестнадцатиразрядного операнда D16.

Например, по команде LXI H, 3FF4H в регистровую пару HL загружается число 3FF4 за один цикл команд. В программе эта команда занимает меньше места по сравнению с загрузкой этого числа с помощью командMVI H, 3FH и MVI L, F4H.

Если надо, например, загрузить число 52080D в регистровую пару BC, то переводят десятичное число в шестнадцатеричную систему: 52080D→CB70H. Тогда команда получит вид LXI B, CBO7H, а в машинном коде ее запись будет следующей: 0170CB (порядок записи числа обратный).

Команды SHLD ADR, LHLD ADR (Store HL Direct и Load HL Direct) служат для пересылки данных между регистровой парой HL и ячейкой памяти, адресуемой по содержимому второго и третьего байтов команды.



	0	1	2	3	4	5	6	7
0	NOP	LXI B,&	STAX B	INX B	INR B	DCR B	MVI B,#	RLC
1	—	LXI D,&	STAX D	INX D	INR D	DCR D	MVI D,#	RAL
2	—	LXI H,&	SHLD *	INX H	INR H	DCR H	MVI H,#	DAA
3	—	LXI SP,&	STA *	INX SP	INR M	DCR M	MVI M,#	STC
4	MOV B,B	MOV B,C	MOV B,D	MOV B,E	MOV B,H	MOV B,L	MOV B,M	MOV B,A
5	MOV D,B	MOV D,C	MOV D,D	MOV D,E	MOV D,H	MOV D,L	MOV D,M	MOV D,A
6	MOV H,B	MOV H,C	MOV H,D	MOV H,E	MOV H,H	MOV H,L	MOV H,M	MOV H,A
7	MOV M,B	MOV M,C	MOV M,D	MOV M,E	MOV M,H	MOV M,L	HLT	MOV M,A
8	ADD B	ADD C	ADD D	ADD E	ADD H	ADD L	ADD M	ADD A
9	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB M	SUB A
A	ANA B	ANA C	ANA D	ANA E	ANA H	ANA L	ANA M	ANA A
B	ORA B	ORA C	ORA D	ORA E	ORA H	ORA L	ORA M	ORA A
C	RNZ	POP B	JMZ *	JNP *	CNZ *	PUSH D	ADI #	RST 0
D	RNC	POP D	JNC *	OUT N	CNC *	PUSH D	SUI #	RST 2
E	PRO	POP H	JPO *	XTNL	CPO *	PUSH H	ANI #	RST 4
F	RP	POP PSW	JP *	DI	CP *	PUSW PSW	ORI #	RST 6
	0	1	2	3	4	5	6	7

Таблица 2.5

8	9	A	B	C	D	E	F	
—	DAD B	LDAX B	DCX B	INR C	DCR C	MVI C,#	RRC	0
—	DAD D	LDAX D	DCX D	INR E	DCR E	MVI E,#	RAR	1
—	DAD H	LHLD *	DCX H	INR L	DCR L	MVI L,#	CMA	2
—	DAD SP	LDA *	DCX SP	INR A	DCR A	MVI A,#	CMC	3
MOV C,B	MOV C,C	MOV C,D	MOV C,E	MOV C,H	MOV C,L	MOV C,M	MOV C,A	4
MOV E,B	MOV E,C	MOV E,D	MOV E,E	MOV E,H	MOV E,L	MOV E,M	MOV E,A	5
MOV L,B	MOV L,C	MOV L,D	MOV L,E	MOV L,H	MOV L,L	MOV L,M	MOV L,A	6
MOV A,B	MOV A,C	MOV A,D	MOV A,E	MOV A,H	MOV A,L	MOV A,M	MOV A,A	7
ADC B	ADC C	ADC D	ADC E	ADC H	ADC L	ADC M	ADC A	8
SBB B	SBB C	SBB D	SBB E	SBB H	SBB L	SBB M	SBB A	9
XRA B	XRA C	XRA D	XRA E	XRA H	XRA L	XRA M	XRA A	A
CMP B	CMP C	CMP D	CMP E	CMP H	CMP L	CMP M	CMP A	B
RZ	RET	JZ *	—	CZ *	CALL *	ACI #	RCT 1	C
RC	—	JC *	IN N	CC *	—	SBI #	RST 3	D
RPE	PCHL	JPE *	XCHG	CPE *	—	XRI #	RST 5	E
RM	SPHL	JM *	EI	CM *	—	CPI #	RST 7	F
8	9	A	B	C	D	E	F	

N — номер порта ввода-вывода

& — двухбайтовый операнд D16

* — двухбайтовый операнд ADR

\# — однобайтовый операнд D8

Команды SPHL и PCHL служат для пересылки данных между регистровой парой HL с адресацией по указателю стека или счетчика команд.

Команды арифметических операций позволяют выполнять арифметические действия в арифметико-логическом устройстве. Микропроцессор КР580ИК80 может выполнять только сложение и вычитание, для выполнения других операций необходимы программы.

Рассмотрим команду ADD R (ADD). При сложении необходимо иметь два операнда, которые соответствуют двум слагаемым, сумма которых определяется. В микропроцессоре один из операндов всегда помещается в аккумулятор, который неявно адресуется самим кодом операции. Вслед за кодом операции необходимо указать, где находится второй операнд. Результат (сумма) помещается в аккумулятор.

Например, команда ADD H означает, что содержимое регистра H должно быть просуммировано с содержимым аккумулятора.

Если требуется сложить числа 12D и 84D и поместить результат в ячейку памяти с адресом 2AA3H, то решение запишется так:

MVI A, 12 MVI B, 84 — загрузка аккумулятора и регистра B числами;
ADD B — сложение;
LXI H, 2AA3H — загрузка HL адресом ячейки памяти;
MOV M, A — засылка суммы в ячейку памяти.

Команда ADC R (ADd with Carry) является разновидностью предыдущей команды, по ней происходит не только сложение двух операндов, но и сложение с признаком переноса, оставшимся от предыдущей операции.

Команда ADI D8 применяется для сложения восьмиразрядного операнда с содержимым аккумулятора.

Например, команда ADC M означает, что содержимое ячейки памяти, адрес которой записан в регистровой паре HL, и содержимое признака переноса должно быть просуммировано с содержимым аккумулятора.

Признак переноса находится в одном из битов регистра признаков F.

Для непосредственного сложения операнда с содержимым аккумулятора и с признаком переноса применяется команда ACI D8 (Add Immediate with Carry).

Команда SUB R (SUBtract) позволяет микропроцессору непосредственно вычесть содержимое одного из регистров или содержимое ячейки памяти из содержимого аккумулятора. Эта команда имеет разновидности SBB R (SuBtract with Borrow) — вычитание с займом, SUI D8 — вычитание непосредственного операнда и SBI D8 — вычитание непосредственного операнда с займом.

Например, по команде SUI 3AH вычитается шестнадцатеричное число 3A из содержимого аккумулятора, а результат операции помещается в аккумулятор.

С помощью команд логических операций можно выполнить следующие действия:

логические операции И, ИЛИ и исключающие ИЛИ с использованием двух операндов (один из операндов всегда помещается в аккумуляторе);

занесение в аккумулятор обратного кода числа;

сравнение двух чисел. Числа сравниваются одно с другим и результат анализируется по признакам «знак» и «нуль»;

сдвиг содержимого аккумулятора вправо или влево. Эта операция используется при умножении и делении чисел. Сдвиг двоичного кода числа на один разряд влево соответствует умножению числа на два.

Команда ANA R (ANd with Accumulator) используется для операции логического И над содержимым одного из регистров или ячейки памяти и содержимым аккумулятора.

Например, команда ANA M означает, что содержимое ячейки памяти, адрес которой находится в регистрах H и L, участвует в операции логического И вместе с содержимым аккумулятора:

```
содержимое ячейки памяти: 01110101
содержимое  аккумулятора: 10010100 «И»
                          --------
                Результат: 00010100
```

Команда ORA R (OR with Accumulator) используется для выполнения операции логического ИЛИ над содержимым одного из регистров или ячейки памяти микропроцессора и содержимым аккумулятора:

```
  содержимое регистра: 11101000
содержимое аккумулятора: 01001111 «ИЛИ»
                         --------
               Результат: 11101111
```

Команды ANI R8 и ORI R8 используются для выполнения аналогичных операций только со вторым байтом команды.

Команда XRA R (eXclusive oR with Accumulator) используется для выполнения операции, исключающей ИЛИ над содержимым одного из регистров или ячейки памяти и содержимым аккумулятора. Для операции непосредственно с операндом применяется команда XRI R8. Результатом выполнения этих команд является байт. Его отдельные разряды равны единице только тогда, когда соответствующие разряды операндов имеют противоположные значения.

Пример 1. Проверить состояние третьего бита регистра B.

```
MOV A, B — передача содержимого B в аккумулятор;
ANI 0000100B — логическая операция И.
```

Если в регистре B третий бит будет нулевым, то в результате получим 00000100B, т. е. признак нуля не устанавливается.

Пример 2. Изменить содержимое ячейки памяти с адресом 0400H. Принять содержимое ячейки равным 01110100B.

```
LXI H, 0400H — загрузка в XL числа 0400H;
MVI A, 00100000B — поместить число в аккумулятор;
XRA M — логическая операция, исключающая ИЛИ.
```

В результате операции изменится только содержимое пятого разряда:

```
0 1 1 1 0 1 0 0 – содержимое с адресом 0400
0 0 1 0 0 0 0 0 – число, которое используется для выполнения операции
---------------
0 1 0 1 0 1 0 0 – Результат в аккумуляторе
```

Команды CMPR и CPI D8 (CoMPare, ComPare Immediate) могут быть использованы для сравнения двух чисел. Одно всегда помещается в аккумулятор. При сравнении одно число вычитается из другого и результат проверяется на «нуль» и «знак»: положительный или отрицательный. Проверяется содержимое признаков переноса и нуля в регистре признаков микропроцессора.

Единственная разница между командами CMP и SUB заключается в том, что при выполнении первой команды результат операции не фиксируется в аккумуляторе. Его содержимое остается неизменным. Если в результате выполнения операции вычитания окажется, что операнды равны, то признак нуля Z устанавливается в единицу, если же значение операнда, хранимого в аккумуляторе, меньше значения второго операнда, то устанавливается в единицу признак переноса C в регистре F [11].

Команда RLC (Rotate Left in Carry) позволяет содержимое аккумулятора циклически сдвинуть влево на один разряд, т. е. все разряды сдвинуть влево на одну позицию. При этом восьмой бит перемещается в разряд признака переноса и в младший разряд «нуль» (рис. 2.6, *а*). Эта операция выполняется только над содержимым аккумулятора.

Пример. Загрузить в триггер переноса C значение разряда 6 содержимого регистра L.

```
MOV A, L — загрузка содержимого L в аккумулятор;
    RCL — циклический сдвиг, при котором разряд 6 займет положение 7;
    RCL — циклический сдвиг, при котором разряд 7 переместится в триггер C.
```

Команда RAL (Rotate Accumulator Left) используется для циклического сдвига влево содержимого аккумулятора на один разряд. Восьмой разряд перемещается в разряд переноса (рис. 2.6, *б*).

Команды RRC и RAR (Rotate accumulatore Right in Carry; Rotate accumulator Right through carry) позволяют циклически сдвинуть содержимое аккумулятора на один разряд вправо. При

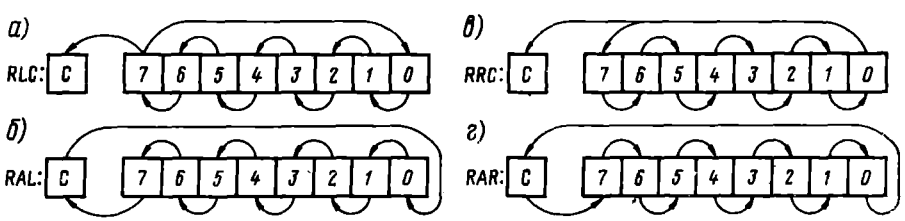


Рис. 2.6. Принципы выполнения команды циклического сдвига содержимого аккумулятора микропроцессора

первой команде содержимое разряда 0 поступает в разряд переноса и в разряд 7 аккумулятора (рис. 2.6, *в*). При второй команде 0 поступает в разряд переноса, а прежнее содержимое разряда переноса поступает в разряд 7 (рис. 2.6, *г*).

Команда DAA (Decimal Adjust Accumulator) выполняет коррекцию результата операции и в два приема корректирует восьмибитное содержимое в аккумуляторе на две бинарно кодированные десятичные цифры с правильной установкой триггера переноса C. Команда не корректирует результат двоичного вычитания.

Команды перехода в заданную область памяти (передачи управления) и работы с подпрограммами используются для изменения в счетчике команд возрастающей последовательности адресов при выполнении какой-либо программы микропроцессором. Если занести в счетчик команд PC код адреса, отличающийся от адреса очередной команды, то порядок выполнения программы может быть изменен. Это вызовет передачу управления работой микропроцессора другой части программы.

Такая передача управления может быть выполнена с помощью трехбайтовой команды безусловного перехода — JMP ADR (JuMP). По этой команде в регистр счетчика команд записывается величина ADR. Таким образом, следующей командой, которую будет выполнять микропроцессор вслед за командой JMP, будет команда, код операции которой записан в ячейке с адресом, равным значению ADR.

Безусловную передачу управления можно осуществить также по команде PCHL, в результате выполнения которой произойдет передача управления по адресу, хранящемуся в регистровой паре HL.

Кроме команд безусловного перехода, микропроцессор имеет восемь трехбайтовых команд условного перехода. При появлении команды условного перехода передача управления по адресу, указанному в команде, происходит только в результате выполнения определенного условия. Если условие не удовлетворяется, то выполняется команда, следующая за командой условного перехода.

Условия, с которыми оперируют команды условной передачи управления, определяются состоянием битов регистра признаков F.

Команда JC (Jump on Carry) осуществляет переход по программе только в том случае, когда в результате выполнения предыдущей операции имеет место переполнение, т. е. если признаком переноса является 1.

Команда JNC (Jump No Carry) обеспечивает переход только в том случае, если содержимое разряда переноса регистра признаков равно 0.

Пример.
```
BEGIN: ADI 01H
       JNC BEGIN
       MOV B, A
```

Первые две команды увеличивают содержимое аккумулятора на единицу до тех пор, пока не появится переполнение. При отсутствии сигнала переполнения (переноса) осуществляется переход по программе к ячейке памяти с символи-

ческим адресом BEGIN. Это и есть адрес команды ADI 01H. Когда появляется сигнал переноса, команда перехода не выполняется, и программа переходит к следующей команде.

Команды JZ, JNZ (Jump on Zero, Jump on No Zero) выполняются только в том случае, когда результат выполнения предыдущей операции нулевой (если признак 0 в регистре признаков имеет значение 1) и соответственно для второй команды — ненулевой (если признак 0 в регистре признаков имеет значение 0).

Пример. Загрузить в ячейку памяти 0800H число 100D и уменьшать его на 1 до тех пор, пока результат не станет равным нулю.

```
LXI H, 0800H — загрузить в HL число 0800H;
 MVI M, 100 — загрузить в ячейку памяти число 100;
MIN:DCR M — уменьшить содержимое ячейки памяти на единицу;
    JNZ MIN — перейти к ячейке памяти MIN, если содержимое ячейки памяти
              не равно 0.
```

Команды JM (Jump on Minus) и JP (Jump on Positive) выполняются только в том случае, если результат предыдущей операции является отрицательным числом (признак «знак» равен 1) и соответственно, если результат предыдущей операции является положительным числом (признак «знак» равен 0).

Команда JPE (Jump on Parity Even) выполняется, если двоичный код результата предыдущей операции содержит четное число единиц.

Команда JPO (Jump on Parity Odd) выполняется в том случае, когда двоичный код результата предыдущей операции содержит нечетное число единиц.

Команда CNZ (Call on No Zero) выполняется в том случае, если признак «нуль» в регистре признаков не равен 1.

Команда RC (Return on Carry) выполняется при наличии признака переноса.

Команда CALL (вызвать подпрограмму). Она содержит 3 байта: 1 байт — для кода операции и 2 байта для указания адреса первой команды подпрограммы. Этот адрес обычно задается символическим именем, например CALL TIME, где второе слово — символическое имя подпрограммы. По этой команде микропроцессор выполняет действия, необходимые для возврата в основную программу, а именно:

в счетчике команд фиксируется адрес команды в основной программе, которая следует за командой вызова;

содержимое счетчика команд (адрес возврата) загружается в стек; содержимое указателя стека изменяется для приема следующих команд, если потребуется;

в счетчик команд загружается адрес, задаваемый командой вызова.

После этого может начаться исполнение подпрограммы.

Команда RET (RETurn — возврат) является последней командой подпрограммы. По этой команде выполняется возврат к главной программе, подготовленный командой вызова. Команда возврата содержит только код операции. По этой команде счетчик команд

получает из стека адрес команды в основной программе, следующей за командой вызова, а содержимое указателя стека соответственно изменяется.

Каждый микропроцессор имеет ряд специальных команд, которые не передают и не обрабатывают информацию, но используются для управления работой микропроцессора.

Команда HLT (HaLT) останавливает выполнение текущей программы до тех пор, пока не появится запрос прерывания от устройства ввода-вывода.

Команды DI, EI (Disable Interrupt), (Enable Interrupt): по первой из этих команд микропроцессор игнорирует запросы прерывания до тех пор, пока не поступит вторая команда EI — разрешение прерывания.

Команда IN N осуществляет ввод данных через порт ввода, номер которого N. Приемником при этом является аккумулятор.

Команда OUT N осуществляет вывод содержимого аккумулятора в порт с номером N.

Команды PUSH B, PUSH D, PUSH H (поместить в стек) используются для засылки содержимого регистровых пар в стек. По команде PUSH PSW в стек засылаются данные из аккумулятора и регистра признаков (так называемые слова состояния программы).

Команды POP B, POP D, POP H, POP PSW (вытолкнуть из стека) используются для пересылки данных из стека в соответствующие регистры).

В системе команд микропроцессора имеются команды, позволяющие сложить два шестнадцатиразрядных числа (DAD B, DAD D,

Таблица 2.6

Команда	Операция	Влияние на биты регистра признаков
INR R	R+1→R	Влияет на все признаки, кроме C
INR YZ	YZ+1→YZ	Не оказывает влияния
DEC R	R−1→R	Влияет на все признаки, кроме C
DCX YZ	YZ−1→YZ	Не оказывает влияния
DAD YZ	HL+YZ→HL	Влияет только на признак C
RLC	Сдвиг влево	То же
RAL	Сдвиг влево через бит признака C	»
RRC	Сдвиг вправо	»
RAR	Сдвиг вправо через бит признака C	»

DAD H, DAD SP). Одно из этих чисел должно быть записано в регистровую пару HL, а другое — в регистровую пару BC, DE, HL или SP. Результат сложения помещается в пару HL.

Очень часто при написании циклических программ используются команды INR R, DEC R, INX YZ, DCX YZ, служащие для увеличения или уменьшения (инкремента и декремента регистровой пары) значения содержимого регистра, ячейки памяти или регистровой пары на единицу. YZ — содержимое регистровой пары BC, DE, HL или регистра SP.

Многие команды этой группы воздействуют на различные биты регистра признаков (табл. 2.6).

Команда NOP не производит никаких операций, однако поскольку она выполняется за определенный отрезок времени, ее можно использовать в программах для задания временных интервалов.

2.8. Организация режима прерывания и примеры программ

Специальные команды используются для организации прерываний текущей программы, выполняемой микропроцессором, по специальным сигналам от внешних устройств или датчиков. При поступлении запроса на прерывание микропроцессор переходит к выполнению программы обработки прерывания, которая реализует действия, являющиеся реакцией на внешние события. При появлении запроса на прерывание микропроцессор после выполнения очередной команды текущей программы считывает не, как обычно, код операции следующей команды из памяти, а код команды вызова подпрограммы, формируемый на шинах данных специальным блоком — контроллером прерываний [4, 8, 9, 10].

В качестве команды вызова подпрограмм используют однобайтовые команды RST 0—RST 7. В зависимости от номера команды ее выполнение ведет к передаче управления на одну из ячеек в начальной области памяти (табл. 2.7).

Именно с команды в этой ячейке и должна начинаться подпрограмма обслуживания прерывания.

Перевести микропроцессор в состояние прерывания можно только программно командой разрешения прерываний EI (возможно

Таблица 2.7

Команда	Адрес начала программы	Команда	Адрес начала программы
RST 0	0000	RST 4	0020
PST 1	0008	RST 5	0028
RST 2	0010	RST 6	0030
RST 3	0018	RST 7	0038

прерывание). О переходе в такое состояние микропроцессор информирует систему посредством установки сигнала высокого уровня (логическая единица) на выводе INT E (INTerruption Enable).

Сигнал INT E может быть переведен в логический нуль, т. е. возможность прерывания снята либо автоматически — при запросах от контроллера прерываний, либо программно — путем подачи команды запрещения прерываний DI (Disable Interruption). Когда микропроцессор принимает сигнал запроса прерывания от внешнего устройства, он прежде всего завершает выполнение текущей команды и в стек записывает адрес следующей команды программы, который будет адресом возврата из прерывания. После того как все текущие данные, которые надо сохранить, будут с помощью программы переписаны в стек, устройству, пославшему запрос, направляется сигнал разрешения прерывания INT A (INTerrupt Acknowledge). Сигнал INT A имеет нулевой уровень и передается по шине управления. Внешнее устройство должно обеспечить ввод в микропроцессор адреса, определяющего точку входа в программу обработки данного прерывания. Для этого устройство выдает на шину данных команду на повторный старт RST (ReSTart), с помощью которой указывает, какая программа из восьми возможных должна выполняться в данном случае. Эта команда указывает одну из областей памяти в первых 64 словах памяти. Каждая область имеет протяженность 8 байт и задается адресами, приведенными в табл. 2.7. Если программа обработки прерываний превышает восемь команд, то в область памяти включается команда безусловного перехода к остальной части программы, которая может быть расположена в любом месте основной памяти.

```
0038    PUSH  PSW
        PUSH  B
        PUSH  D
        PUSH  H
-------------------
Текст программы
      прерывания
-------------------
        POP   H
        POP   D
        POP   B
        POP   PSW
        EI
        RET
```

Рис. 2.7. Фрагмент записи программы прерывания с использованием команды RST 7

Конец программы обработки прерывания должен содержать команды POP для восстановления содержимого всех регистров и команду RET (возврат), по которой в счетчик команд заносится значение, имевшееся в нем в момент начала прерывания; только после этого происходит возврат в основную программу.

Команда RST может использоваться также при отладке программы (рис. 2.7).

Рассмотрим примеры программ для микропроцессора КР580ИК80, которые могут использоваться как фрагменты или подпрограммы для сложных программ управления устройствами электрических железных дорог.

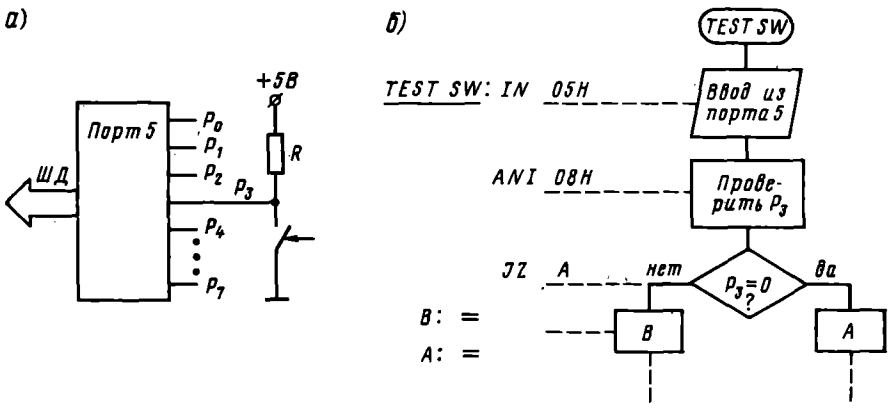


Рис. 2.8. Схема соединения двоичного датчика с портом ввода алгоритм программы «опрос двоичного датчика»

Пример 1. Типовая программа, которая часто встречается при автоматизации различных устройств, получила название «опрос двоичного датчика». На рис. 2.8, *а* приведена схема соединения датчика с портом ввода, причем он соединен с четвертым по счету входом порта. Требуется написать программу, которая проверяла бы состояние датчика и осуществляла переход к подпрограмме А в том случае, если сигнал на $P_3=0$, и к подпрограмме В, если сигнал на $P_3=1$.

Схема алгоритма этой программы изображена на рис. 2.8, *б* справа, а слева дан текст программы. Информация ввода из порта *5* передается в аккумулятор, где состояние входа P_3 проверяется с помощью маски всех остальных битов входного слова. Для этого в команде ANI задается число 08Н, двоичный код которого (маска) равен 00001000, т. е. единице в четвертом разряде. По команде логического И с вторым байтом определяется результат P_3, и если он нулевой, то по команде JZ определяется направление ДА и переходят к подпрограмме А. Если $P_3=1$, то программа продолжается в направлении НЕТ и переходит к подпрограмме В.

Пример 2. Широко применяется подпрограмма, которая получила название «ожидание события» (рис. 2.9). Схема устройства состоит из порта ввода и контакта *S*.

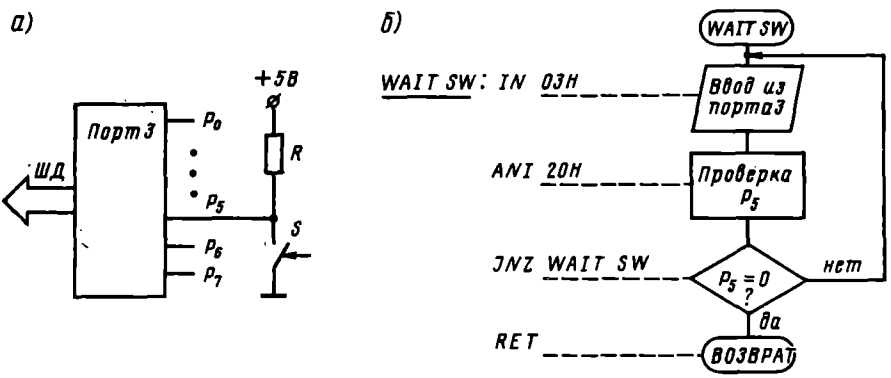


Рис. 2.9. Схема соединения двоичного датчика с портом ввода (*а*), алгоритм программы «ожидание события» (*б*)

Необходимо написать подпрограмму, которая постоянно опрашивает состояние входа P_5 до тех пор, пока оно не станет равным нулю, т. е. пока контакт не будет замкнут. Далее необходимо возвратиться в основную программу, если контакт замкнут Суть алгоритма сводится к следующему. Байт, присутствующий на входе порта ввода, считывается и засылается в аккумулятор. Состояние входа P_5 анализируется в аккумуляторе маскированием. Для этого по команде ANI происходит логическое сравнение с числом 20 Н, имеющим код с единицей в пятом бите. Если результат равен нулю, что может быть в случае $P_5=1$ (контакт не замкнут), то подпрограмма продолжается в направлении НЕТ, и операция проверки повторяется до тех пор, пока P_5 не станет равным нулю (контакт замкнут). Затем подпрограмма продолжается в направлении ДА и производит возврат к основной программе с помощью команды ВОЗВРАТ.

Пример 3. В микропроцессорных системах управления часто необходимо согласовывать время выполнения программы и время срабатывания исполнительного механизма устройства управления. Как правило, необходимо вводить в программы временные задержки различной длительности. Временная задержка может быть сформирована программой, в которой некоторое множество команд не выполняет никаких операций, но занимает машинное время. Для формирования временных задержек применяется метод, при котором в рабочий регистр загружается некоторое число и затем оно последовательно уменьшается на единицу до тех пор, пока не станет равным нулю. Чем большую задержку хотим получить, тем большее число нужно занести в регистр. В регистр В (рис. 2.10) загружается число X, значение которого определяется требуемой задержкой по времени. Из содержимого регистра В вычитается единица, и выполняется проверка его содержимого. Если результат вычисления станет равным нулю, то происходит возврат в основную программу. Если результат не равен нулю, то процесс вычитания единицы повторяется. Текст программы приведен рядом со схемой алгоритма.

Для того чтобы определить величину X, необходимо знать время выполнения каждой команды и таким образом определить, сколько раз данная последовательность команд должна повторяться, чтобы получить требуемую задержку времени. Многократно используются команды DCR B и JNZ. Необходимо учесть еще и время выполнения команды вызова этой подпрограммы CALL TIME и однократно выполняемые команды MVI B, X и RET

Определим, за сколько тактов подпрограмма выполнится однократно, принимая время одного такта равным 0,5 мкс. В инструкции к микропроцессору указано число тактов для выполнения каждой команды:

CALL TIME — 17 тактов → 8,5 мкс;
MVI B, X — 7 тактов → 3,5 мкс;
DCR B — 5 тактов → 2,5 мкс;
JNZ FORW — 10 тактов → 5 мкс;
RET — 10 тактов → 5 мкс.

Итого для однократного выполнения программы необходимо 17 мкс. Чтобы получить задержку времени 100 мкс, надо выполнить команды DCR и JNZ столько раз, чтобы этот процесс выполнялся за 100—17=83 мкс. Время выполнения этих команд 7,5 мкс. Решить такую задачу можно, выполнив команды 10 раз, что дает задержку 75 мкс, а 8 мкс можно добавить, использовав четыре команды NOP, длительность которой равна 2 мкс. Подпрограмма показана в табл. 2.8.

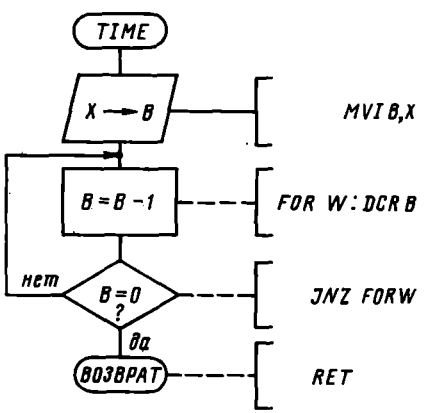


Рис. 2.10. Алгоритм программы организации временной задержки в микропроцессорной системе управления

Пример 4. Требуется разработать программу для специализированного микропроцессорного устройства, решающего частную задачу о сигнализации сра-

Таблица 2.8

Метка	Мнемокод	Операнд	Комментарий
TIME:	MVI	B, 10	Загрузить в B число 10
FORW:	DCR	B	Уменьшать на 1 содержимое B
	JNZ	FORW	Повторять процесс, если результат не равен нулю
	NOP		} Пустая операция
	NOP		
	NOP		
	NOP		
	RET		} Возврат в основную программу

батывания защиты какого-либо агрегата на электровозе или тяговой подстанции. Этот факт отображается загоранием светодиода на пульте управления *V2* (рис. 2.11) В нормальном состоянии светится светодиод *V1*, но после кратковременного срабатывания защиты светодиод *V1* должен на 0,25 с погаснуть, а светодиод *V2* загореться. После этого в течение 0,5 с устройство не должно реагировать на срабатывание защиты. Эта выдержка необходима для записи факта срабатывания защиты на самопишущем устройстве.

Для реализации задачи необходимо использовать в микропроцессорном устройстве два порта — один порт ввода *P2* и один порт вывода *P4*. К порту ввода по линии, связанной с младшим разрядом, подключается блокировочный контакт аппарата защиты *S1*, а к порту вывода по линиям, связанным с его двумя младшими разрядами, подключены два светодиода *V1* и *V2*. Дешифраторы *D2* и *D3* формируют сигналы $\overline{BM1}$, $\overline{BM2}$, служащие для подключения соответствующих портов. При этом сигналы $\overline{BM1}$ и $\overline{BM2}$ появляются только тогда, когда

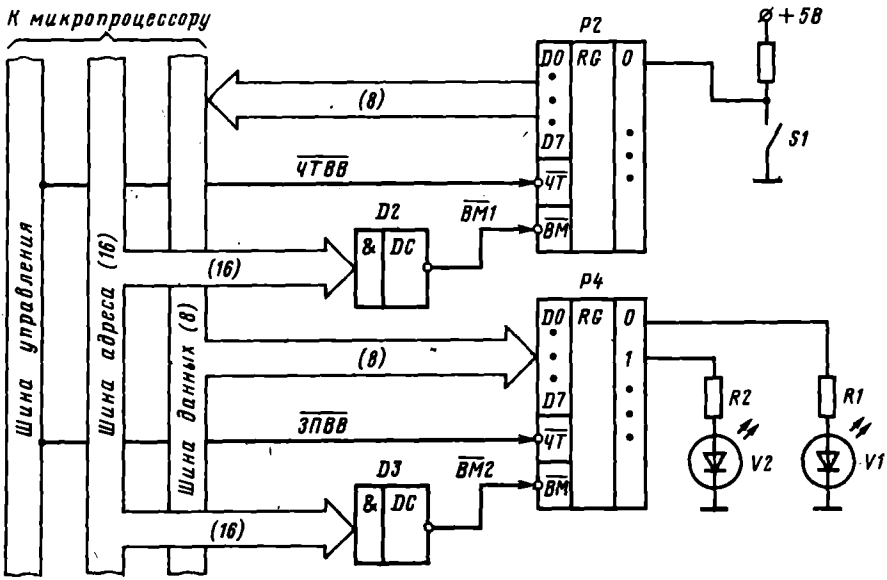


Рис. 2.11. Схема внешних соединений микропроцессорной системы для решения задачи о сигнализации срабатывания защиты агрегата

на младших восьми разрядах шины адресов возникают коды соответственно 01 и 02. В таком случае говорят, что порт *P2* включен, как устройство с номером 1, а порт *P4*, как устройство с номером 2. Далее порт *P2* будем называть портом 1, а *P4* — портом 2.

Перед тем как писать программу в системе команд микропроцессора, необходимо подробно изобразить схему алгоритма работы устройства. До этого полезно составить словесное описание алгоритма.

Срабатывание защиты будет сопровождаться подачей логической единицы в порт ввода *P2*. Поэтому программа должна анализировать этот факт. Если он не произошел, то микропроцессор должен повторить анализ. Если факт срабатывания защиты будет установлен, то в порт *P4* посылается байт, в младших разрядах которого будет комбинация 10. При этой комбинации светодиод *V2* горит, а *V1* погашен. После этого делается выдержка времени 0,25 с, а затем после полусекундной выдержки переходят вновь к началу программы. Выдержки времени выполняются программным способом.

Схема описанного алгоритма показана на рис. 2.12. Справа от схемы алгоритма приведены необходимые пояснения и используемый мнемокод операции. В табл. 2.9 приведена распечатка (листинг) программы для микропроцессора, где в первых двух колонках указаны адреса кодов команд и двоичных (объектных) кодов команд, которые были получены использованием таблицы перевода мнемоники команд в коды в шестнадцатеричной форме. Если при составлении программы операнды задаются неявно, то в начале или в конце текста программы определяется, чему эти операнды равны на самом деле. Для этого в поле 3 записывают символическое имя операнда, в поле 4 — сокращенное английское слово «равно» (EQU), а в поле 5 — действительное значение операнда. В поле 2 записываются коды и соответствующие операнды команд, при этом символические имена заменяются на их действительные значения.

При записи шестнадцатиразрядных (двухбайтовых) операндов и адресов в поле 5 слева записывают два старших, а справа — два младших разряда числа. При записи кодов операндов в поле 2 порядок записи обратный, что объясняется порядком байтов шестнадцатиразрядных чисел в памяти микропроцессора.

Первое значение адреса в поле 1 (адрес первой команды программы) выбирается программистом в зависимости от свободного места в памяти микропроцессорной системы. Поле заполняется так, чтобы значения адресов соответствовали кодам команд. При расчете очередного адреса в поле 1 в зависимости от длины предшествующей команды к предыдущему значению прибавляются 1, 2 или 3 в зависимости от того, какие предыдущие команды используются: одно-, двух- или трехбайтовые. Необходимо учитывать, что номера адресов указываются в шестнадцатеричной форме (т. е. $8+3=B$, $B+2=D$, $F+3=12$ и т. д.). Коды второго и третьего байтов в командах передачи управления и вызова подпрограммы в поле 2 вносятся в таблицу в последнюю очередь после заполнения всех строк поля 1. Коды из поля 2 вводятся в память микропроцессорной системы по адресам, указанным в поле 1, и при пуске системы с начального адреса она начинает выполнять программу.

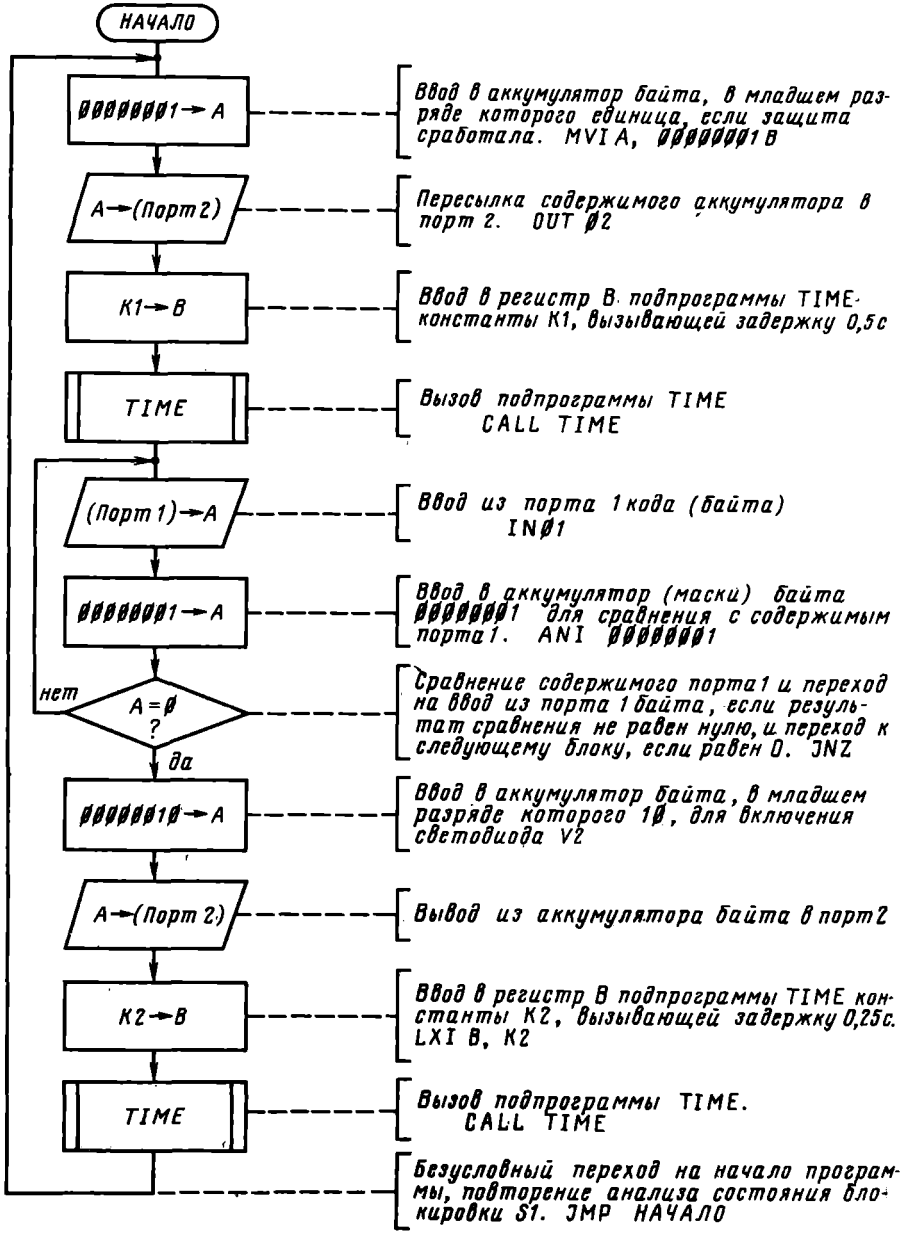


Рис. 2.12. Алгоритм программы для решения задачи о сигнализации срабатывания защиты агрегата

Таблица 2.9

Адрес	Код	Метка	Мнемокод	Операнд	Комментарий
1	2	3	4	5	6
0000	012800		LXI	SP, СТЕК	; настройка указателя стека
0003	3E01	НАЧАЛО:	MVI	A, 01H	; 00000000→A
0005	D302		OUT	02H	; включить *V*1 и выключить *V*2
0007	0170CB		LXI	B, K1	; задать величину задержки 0,5 с
000A	CD1F00		CALL	TIME	; вызов подпрограммы задержки
000D	DB01	ВВОД:	IN	01H	; ввод байта из порта 1
000F	E601		ANI	01H	; маскирование неиспользованных разрядов
00011	C20D00		JNZ	ВВОД	; повторить ввод, если *S*1 не замкнут
0014	3E02		MVI	A, 02H	; 0000010→A
0016	D302		OUT	02H	; включить *V*2, выключить *V*1
0018	01B865		LXI	B, K2	; задать задержку 0,25 с
001B	CD2100		CALL	TIME	; вызов подпрограммы задержки
001E	C30300		JMP	НАЧАЛО	; возврат на «начало»
0021	OB	TIME:	DCX	B	; подпрограмма задержки TIME; уменьшить на 1 содержимое BC
0022	7B		MOV	A, B	; переслать в A содержимое B
0023	B1		ORA	C	; B и C равны нулю?
0024	C22100		JNZ	TIME	; если нет, то повторить цикл
0027	C9		RET		; возврат в основную программу
		K1	EQU	52080D	; присвоение числовых значений
		K2	EQU	26040D	символическим операндам
		СТЕК	EQU	0028H	; присвоение символического значения номера ячейки начала стека

Необходимо пояснить расчет времени задержки и определение числа, которое помещается в регистровую пару BC для задания временно́й задержки 0,5 с.

Выполнение любой команды микропроцессором занимает строго определенное время. Поэтому необходимо знать длительность выполнения каждой команды, которая определяется числом тактов, необходимых для выполнения этой команды, и длительностью одного такта, которая зависит от длительности импульсов тактового генератора микропроцессора.

Так, например, общее время однократного выполнения подпрограммы TIME составляет 9,6 мкс.

Следовательно, для задания временно́й задержки 0,5 с подпрограмма должна быть выполнена $0{,}5/(9{,}6\cdot 10^{-6})=52\,080$ раз, и этот результат присваивается операнду K2. При переводе программы в машинные коды это число переводится в шестнадцатери́чную форму (CB70) и используется в команде загрузки регистровой пары BC.

2.9. Микрокалькуляторы

Появление в настоящее время новых типов ЭВМ: персональных ЭВМ, домашних компьютеров (БК-0010, «Микроша» и др.), программируемых микрокалькуляторов («Электроника БЗ-34», «Электроника МК-56», «Электроника МК-52») открыло перед специалистами любых профессий большие возможности непосредственного проведения вычислительных экспериментов для поиска оптимальных путей решения производственных задач [11, 12, 13, 14]

Микрокалькуляторы имеют много существенно общих черт в структуре построения, конструкции и методах использования при решении задач с вычислительными средствами универсального применения ЭВМ и мини-ЭВМ. Наибольший интерес для инженерных расчетов представляют программируемые микрокалькуляторы, способные автоматически по заранее написанной программе выполнять решение необходимой задачи.

Отечественные микрокалькуляторы типа «Электроника БЗ-34» с емкостью памяти, равной 98 шагам прграммы, имеют 14 регистров памяти для хранения исходных данных и промежуточных результатов. Недостатком их является отсутствие возможности сохранять программу при выключении питания, что вызывает потери времени при многократном использовании рабочих программ.

Микрокалькулятор «Электроника МК-52» имеет объем программной памяти, увеличенный до 105 шагов, 15 регистров памяти. Он имеет возможность сохранять программы в энергозависимой памяти объемом 512 шагов программы и хранить их до 5000 ч. Число циклов перезаписи информации равно 104 [14]

Программное обеспечение дополнено возможностями генерирования псевдослучайных чисел с помощью датчика случайных чисел, определения максимального значения одного из двух чисел, хранящихся в операционных регистрах, выполнения логических операций над числами.

Важнейшей характеристикой МК являются используемые правила ввода данных и последовательность выполнения операций или их логическая структура. Различают МК с алгебраической логикой со скобками и с инверсной польской записью. В МК с алгебраической логикой операции выполняются в том порядке, в котором они вводятся. Организация вычислений более сложной структуры требует применения открывающихся и закрывающихся скобок, что значительно усложняет структуру МК, требует дополнительных запоминающих устройств.

В отечественных МК широко распространена логическая организация, называемая инверсной польской логикой, которая позволяет получить однозначную запись арифметических выражений без использования скобочной записи. Термин «инверсная» характеризует принятый способ записи операций: сначала записываются операнды, а затем символ операций. Операции производятся в прямом порядке записи их символов над предшествующими операндами. Результат операции запоминается в регистрах X и Y как новый операнд для последующей операции. Такое построение позволяет уменьшить объем дополнительной памяти МК. Команды вводятся вручную или из программной памяти в порядке их выполнения.

Это составляет заметное преимущество инверсной польской логики и объясняет широкое ее распространение. Отличительной чертой МК с инверсной польской записью является наличие клавиши «↑», используемой для разделения последовательно вводимых операндов и ввода операнда в регистр Y.

Все составные части МК объединены в соответствии с общей структурой вычислительной машины (рис. 2.13): устройство ввода-вывода, процессор, запоминающее устройство, устройство управления.

Устройство ввода представлено клавиатурой с многофункциональными клавишами, устройством вывода является индикатор (дисплей). Команда, введенная с клавиатуры, попадает в запоминающее устройство (*ОЗУ* программ), которое состоит из набора ячеек, имеющих номера, в каждой из которой может быть записано одно кодовое слово. Адрес текущей ячейки запоминается в программном указателе. При каждом вводе команды адрес в программном счетчике увеличивается на единицу. Адресный стек состоит из пяти ячеек и используется для запоминания адреса команды, на которую нужно передать управление после окончания работы какой-либо подпрограммы.

Регистры памяти служат для записи и одновременного хранения числовой информации.

Постоянное запоминающее устройство содержит программы, которые и организуют процесс вычислений. Эти программы нельзя прочесть, к ним можно только обращаться. Они обеспечивают выполнение всех арифметических операций, определяют значения всех функций, обозначенных на клавиатуре. Все операции по программам выполняет арифметико-логическое устройство, работающее вместе с операционным стеком, состоящим из пяти регистров X, Y, Z, T, X1, (регистр предыдущего результата). Совместную

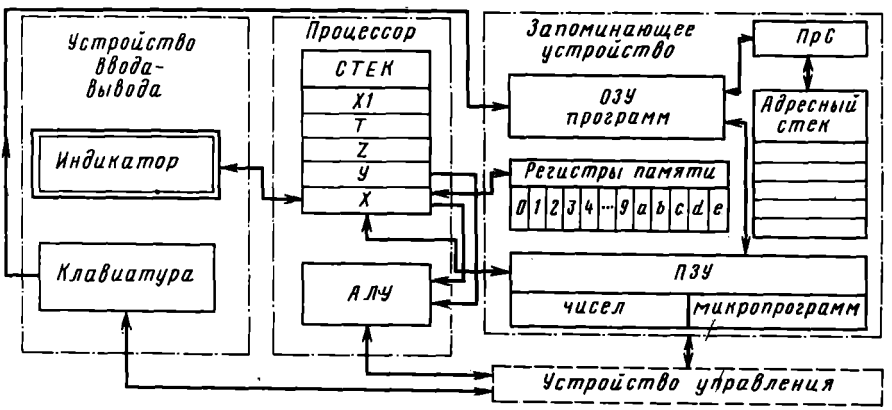


Рис. 2.13. Структурная схема программируемого микрокалькулятора

согласованную работу всех частей МК обеспечивает устройство управления.

Полный цикл работы МК при выполнении программы начинается с нажатия клавиши «В/О» очистки программного счетчика *ПрС* (см. рис. 2.13). Нажатие клавиши «С/П» запускает программу. *Устройство управления* считывает команду, адрес которой записан в программном счетчике, при этом содержимое счетчика увеличивается на единицу. После анализа команды в регистре кода операции и определения типа операции команда пересылается из *ОЗУ* в *АЛУ* По сигналам из устройства управления процессор вырабатывает результат операции.

Затем устройство управления опрашивает программный счетчик и определяет следующую команду. После этого весь цикл повторяется.

При выполнении разветвленной программы считывание из программной памяти кода команды безусловного перехода БП и его идентификация в регистре кода операции приводят к засылке в программный счетчик адреса перехода, считываемого следующим. В этом случае выполнение программы будет выполняться с оператора, код которого хранится по адресу перехода, который в данный момент находится в программном счетчике.

При считывании из программной памяти команды ППN (команды безусловного перехода к подпрограмме) считываемый за ней адрес перехода также засылается в программный счетчик, но в этом случае *адресный стек* смещается «вверх» и предыдущее содержимое программного счетчика (адрес перехода) засылается во второй регистр *адресного стека*. После этого выполнение программы также продолжается с шага подпрограммы, адрес которого заслан в программный счетчик.

После выполнения подпрограммы и считывания кода оператора возврата «В/О» из программной памяти *адресный стек* смещается «вниз». К новому содержимому программного счетчика (ранее хранимому во втором регистре стека) добавляется единица и выполнение программы продолжается по этому адресу с шага, следующего за соответствующим оператором обращения к подпрограмме.

Для конструктивной реализации оператора условного перехода вида ХАZ (рис. 2.14, *а*, шаг *13*) процессор МК дополняют регистром состояний, соединенным с регистром Х операционного стека. В процессе выполнения программы в регистр состояний непрерывно заносится информация о знаке и равенстве нулю содержимого регистра Х. При считывании из программной памяти кода команды условного перехода устройство управления проверяет содержимое регистра состояния. Если проверяемое условие в команде условного перехода выполнено, то считываемый следующим сигнал, кодирующий указатель перехода, блокируется и выполнение программы продолжается с оператора, записанного после оператора

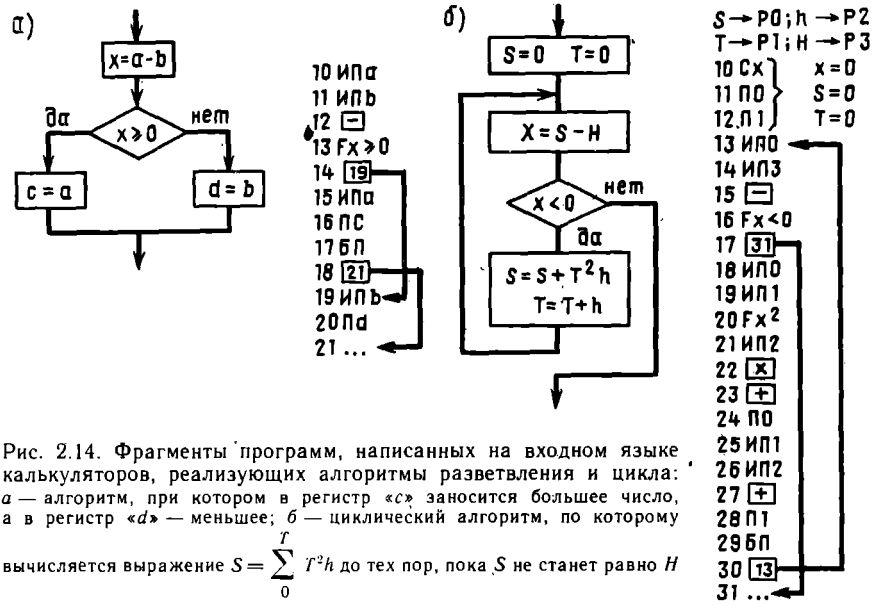


Рис. 2.14. Фрагменты программ, написанных на входном языке калькуляторов, реализующих алгоритмы разветвления и цикла: *а* — алгоритм, при котором в регистр «*c*» заносится большее число, а в регистр «*d*» — меньшее; *б* — циклический алгоритм, по которому вычисляется выражение $S = \sum_{0}^{T} T^2 h$ до тех пор, пока S не станет равно H

условного перехода (см. рис. 2.14, *а, б* соответственно шаги *15* и *18*). Если условие не выполняется, то считываемый после команды условного перехода код указателя перехода заносится в программный счетчик и выполнение программы продолжается с оператора, код которого хранится по адресу, записанному в указателе перехода (см. рис. 2.14, *а, б* соответственно шаги *14* и *31*).

В программируемых МК допускается косвенная адресация, предусматривающая возможность засылки в программный счетчик не только адресов переходов из регистра кода операции, но и содержимого любого регистра памяти.

Косвенная адресация обеспечивает возможность автоматического выполнения операций над адресами переходов и номерами регистров памяти и расширяет возможности программирования МК.

Более подробные сведения о системе команд МК приведены в технических описаниях к ним, а также в специальной литературе [11, 15]. Общая схема программного режима МК имеет следующий вид: F ПРГ ПРОГРАММА F АВТ ВВОД ИСХОДНЫХ ДАННЫХ В/О С/П.

Сразу после включения питания МК находится в режиме автоматических вычислений. Для перевода ее в режим программирования следует набрать команду F ПРГ При этом на индикаторе справа появляются цифры ∅ ∅, т. е. счетчик команд устанавливается на нулевой адрес.

В процессе ввода программы на индикаторе высвечиваются коды трех последних команд и адрес очередной ячейки ПП, куда можно занести следующую команду после ввода программы и ее

проверки для организации автоматических вычислений, набрав команду FABT. Затем следует ввести необходимые исходные данные. При первом вводе программы целесообразно ввести такие данные, которые дают известный конечный результат, т. е. выполнить расчет контрольного примера.

Для индикации результата вычислений какого-то этапа расчета следует записывать в программу команду С/П (счет—пуск). После записи или анализа результата расчета дальнейшая работа программы осуществляется нажатием клавиши С/П.

В качестве примера рассмотрим программу[1] для МК-56, выполняющую тяговый расчет при использовании прямой адресации.

Как известно, в основе тягового расчета лежит решение уравнения движения поезда

$$\frac{dv}{dt} = \xi f_y$$

Представив это уравнение в конечных разностях и измеряя скорость движения поезда v в километрах в час, а время в секундах, выразим из него приращение скорости ΔV на интервале времени Δt:

$$\Delta v_к = f_y \Delta t/30.$$

Текущие значения скоростей $v_{к+1}$, пути $wS_{к+1}$, расхода энергии на тягу $W_{к+1}$, времени хода $T_к$ могут быть найдены на основании следующих соотношений:

$$v_{к+1} = v_к + \Delta v_к;\ S_{к+1} = S_к + \Delta S_к;$$
$$W_{к+1} = W_к + \Delta W_к;\ T_{к+1} = T_к + \Delta t,$$

где $\Delta W_к = VI_э \Delta t/3600$; $\Delta S_к = v_{ср} \Delta t/3{,}6$ (здесь $v_{ср} = v_{к+1} - \Delta v_к/2$).

Удельная ускоряющая сила

$$f_y = f_к - w_{0x} - i - b,$$

где $f_к$ — удельная сила тяги; $f_к = F_к/(m_л + m_с)$; $m_л$ — масса сответственно локомотива и состава; $m_с$ — сила тяги локомотива при скорости движения; w_{0x} — удельное основное сопротивление движению в режиме тяги или выбега; i — значение уклона пути; b — удельная тормозная сила.

Зависимости $f_к(v)$, $w_x(v)$ и $b(v)$ должны быть предварительно получены по известным формулам и представлены в виде таблиц с помощью программ, составленных для калькулятора.

Перед составлением программы тягового расчета распределим регистры памяти под константы и переменные и запишем в виде следующей таблицы:

Регистры памяти	0	1	2	3	4	5	6	7	8	a	b	c
Константы и переменные	v	$f_к$	w_{0x}	i	$b(v)$	S	$I_э$	W	T	U	Δv	Δt

[1] Программа составлена канд. техн. наук Г. И. Гатальским.

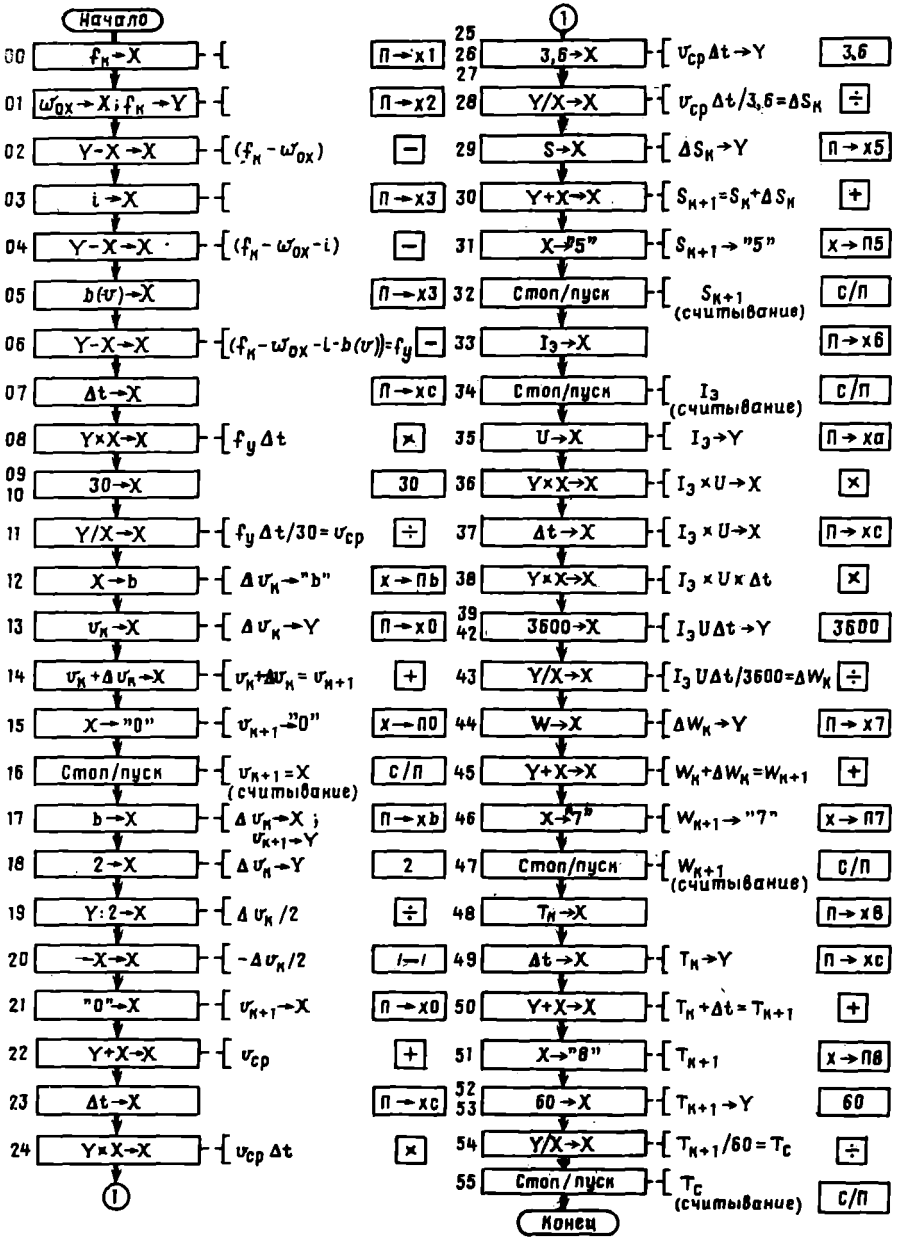


Рис. 2.15. Схема алгоритма тягового расчета и соответствующие команды в обозначениях клавиш калькулятора МК-56 (цифры слева указывают номера шагов программы)

Схема алгоритма и программа тягового расчета для МК-56 приведены на рис. 2.15.

Краткая инструкция по работе с программой:

после ввода программы перейти в режим «автоматическая работа», нажимая клавиши F АВТ;

набрать команду БП 00 F ПРГ и проверить по кодам введенную программу, нажимая клавишу Ш. При наличии ошибки записать требуемую команду;

перейти в режим «автоматическая работа» F АВТ и ввести в регистры памяти значения $f_к$, w_{0x}, i, Δt для скорости $v=0$;

нажимая клавишу В/О, С/П, получаем значение скорости v_{4+1};

далее нажимая клавишу С/П последовательно получаем значения $S_{к+1}$, $I_э$, $W_{к+1}$, T и заносим их в таблицу;

выполнив шаг тягового расчета, заносим в соответствующие регистры памяти новые значения $f_к$, w_{0x}, i, Δt.

При движении поезда по участку происходит чередование режимов тяги, выбега и торможения. Для режима тяги в соответствующие регистры заносим значения $f_к$, w_{0x}, $I_э$ при данном значении скорости. Для режима выбега заносим в соответствующие регистры $f_к=0$, w_{0x} и $I_э=0$. Для режима торможения заносим $f_к=0$, $I_э=0$, $b(v)$.

Интервал расчета по времени Δt назначаем таким, чтобы приращение скорости не превышало 5 км/ч. Изменения профиля пути производим при значениях S, отличающихся от изменения профиля на ± 50 м.

В результате выполнения тягового расчета получаем таблицу значений v, S, $I_э$, W, $T_с$, на основании которой строим кривые движения поезда.

Контрольные вопросы

1. Что такое микропроцессор и чем он отличается от процессора?
2. Что лежит в основе построения микроЭВМ и какие устройства необходимы для ее функционирования?
3. Что такое шина и порты?
4. Что такое аппаратные и программные средства ЭВМ?
5. Что такое языки высокого уровня и зачем они нужны?
6. В чем разница между машинным языком и языком Ассемблера? Какой из них и где наиболее удобен?
7. Почему программируют в двоичном коде и зачем нужен шестнадцатеричный код?
8. Как устроена память ЭВМ, чем ограничивается ее размер, как он определяется?
9. В чем заключаются принципы программного управления ЭВМ и хранимой в памяти программы?
10. Что такое схема алгоритма и как ее использовать для написания программы?
11. Какие типовые структуры алгоритмов встречаются при разработке сложных схем алгоритмов обработки информации?
12. Что такое естественная и нормальная формы представления чисел?
13. Что такое адресное поле?
14. Что такое система команд микропроцессора и на какие группы команд она разбивается? Привести примеры.
15. Приведите примеры типовых программ при программировании микропроцессора. Объясните их необходимость.

Глава 3

МИКРОЭВМ И МИКРОПРОЦЕССОРНЫЕ СИСТЕМЫ В ЭЛЕКТРИЧЕСКОЙ ТЯГЕ

3.1. Элементная база микропроцессорных систем

На определенном этапе развития полупроводниковой техники появилась возможность совместить в одном кристалле несколько полупроводниковых элементов (диодов и транзисторов), соединив их между собой определенным образом. Наряду с полупроводниковыми в составе такого кристалла обычно могут быть получены и некоторые другие элементы электронной техники, например, резисторы, конденсаторы и некоторые другие. Естественно, эти элементы рассчитаны на работу при небольших электрических напряжениях и токах и выполняют функции информационной электроники. Полученная таким образом в монолитном многослойном кристалле полупроводниковая структура называется *интегральной схемой* (ИС).

Количество отдельных элементов в ИС называют *степенью интеграции*. Этот показатель может составлять несколько тысяч, а в ряде случаев и несколько десятков тысяч. Электрические соединения между элементами в ИС также реализуются на этапе ее изготовления без дополнительного монтажа (проводов).

Речь идет в основном о цифровых интегральных схемах, которые реализуются на базе ИС малой и средней степени интеграции. При этом в одном корпусе получают типовые узлы цифровых схем: счетчик, сумматор, регистр, блок сравнения кодов, многоразрядные логические схемы.

Переход на *интегральные схемы с большой степенью интеграции* (БИС) позволяет получить принципиально новое решение: в одном кристалле реализуется базовый узел малой ЭВМ, например, процессор или запоминающее устройство.

Общеизвестным примером БИС является микрокалькулятор общего применения, в котором БИС дополнена клавиатурой и цифровым индикатором. Простейшие микрокалькуляторы не имеют запоминающих устройств внешнего доступа, но возможности современных БИС позволяют создать устройство с программируемой работой. Такая БИС имеет оперативное запоминающее устройство, используемое для записи промежуточных результатов, и специальное запоминающее устройство для ввода и хранения расчетных программ.

Микропроцессорные простые и развитые комплекты БИС (табл. 3.1) выпускаются серийно и широко используются в практике автоматизации. С точки зрения пользователя интерес представляют

Таблица 3.1

Серия БИС (разрядность)	Базовая технология (тип)	Число БИС	
		общее	в базовом комплекте
К536(8)	*p*-МДП	12	2
К580(8)	*n*—МДП	3	1
К581(16)	*n*—МДП	4	2
К584(4п)	$И^2Л$	3	2
К586(16)	*n*—МДП	4	1
К1800(4п)	ЭСЛ	8	2
К1801(16)	*n*—МДП	2	1
К587(4п)	ТТЛДШ	4	2
К588(16)	КМДП	3	2
К589(2п)	ТТЛДШ	8	2
К1883(8п)	*n*—МДП	4	2
К1802(8п)	ТТЛДШ	11	2
К1804(4п)	ТТЛДШ	4	4
К1810(16)	*n*-МДП	3	5

в первую очередь функциональные возможности этих комплектов, т. е. набор входящих в них блоков (обычно от 3 до 10), разрядность и быстродействие. Серии, типы и параметры БИС простых микропроцессорных комплектов приведены в табл. 3.2, а развитых микропроцессорных комплектов — в табл. 3.3.

Важным показателем комплекта является технологическая база, которая определяет быстродействие, помехоустойчивость, цену, показатели надежности, напряжение электропитания, а также возможности сопряжения блоков от разных комплектов между собой. Наиболее отработанными и простыми считаются элементы типа ТТЛ, т. е. выполненные на базе двухтранзисторной логики и реализующие операции вида $y=x_1x_2$. Для них характерно напряжение питания 5 В и токи нагрузки элементов до 16 мА. С целью улучшения помехоустойчивости и быстродействия в типовые элементы типа ТТЛ могут вводиться диоды Шоттки. При этом получаются ИС типа ТТЛДШ (К589, К1802, К1804).

Наиболее быстродействующими являются элементы типа ЭСЛ. Для них характерно использование активного режима работы транзисторов в состояниях 0 и 1, включение на выходах элемента эмиттерных повторителей. Все элементы типа ЭСЛ имеют инверсные выходы, что облегчает построение сложных счетно-решающих узлов. По ЭСЛ-технологии выполнены БИС типа К1800. Недостаток БИС, выполненных по этой технологии, состоит в необходимости иметь два источника питания.

Таблица 3.2

Тип	Разрядность (емкость), бит	Частота, МГц	Напряжение питания, В	Выполняемая функция
КР580ИК80	8	2,5	+5;+12; −5	Процессор параллельный
КР580ВВ51	8	2,0	+5	Интерфейс последовательный
КР580ВВ55	8	2,0	+5	Интерфейс параллельный
КР580ВИ53	8;16	2,0	+5	Таймер
КР580ВТ57	8;16	2,0	+5	Контроллер доступа к памяти
КР580ВН59	8	2,0	+5	Контроллер прерываний
КР580ВГ75	8; 16	3,0	+5	Контроллер ЭЛТ
КР581РУ4	16К	—	+5	ОЗУ
КР581ВЕ1	16	2,5	+5;+12	Процессор
КР581ВА1	8	0,4	−12;+5	Приемопередатчик асинхронный
КР582ИК1	4п	0,6	1,5	Процессор
КР582ИК2	4п	0,6	1,5	Процессор
К583ИК3	8п	1,0	1,5	Процессор
К583КП2	4п	—	1,5	Приемопередатчик без памяти
К583КП3	5	—	1,5	Приемопередатчик с памятью
К583ХЛ1	8	—	1,5	Коммутатор
КР584ИК1А,Б,В	4п	0,5	5	Процессор
КР588ВС2	16	1,0	5	АЛБ
КР588ВУ2	150	1,2	5	Память специальная
КР588ВГ1	—	1,0	5	Контроллер
КР588ИР1	8	—	5	Регистр
КР588ВА1	8	—	5	Приемопередатчик

Таблица 3.3

Тип	Разрядность (емкость), бит	Частота, МГц	Напряжение питания, В	Выполняемая функция
К1800ВС1	4п	36	−5; +2; −2	АЛБ
К1800ВБ2	—	36	−5; +2; −2	Синхронизация
К1800ВТ3	4п	36	−5; +2; −2	Управление памятью
К1800ВР8	16п	36	−5; +2	Сдвига́тель
К1801ВЕ1	16	8	5	МикроЭВМ
К1801ВМ1	16	5	5	Процессор
К1801ВП1	—	—	5	Матрица вентилей
КР1802ВС1	8п	8	5	АЛБ
КР1802ВР1	(16×4)п	10	5	Регистры
КР1802ВР1	16п	8	5	Расширитель
КР1802ВВ1	(4×4)п	10	5	Обмен информацией
КР1802ВР2	(8×8)п	8	5	Умножение
КР1802ВВ2	—	10	5	Интерфейс-адаптер
КР1804ВС1	4п	8	5	Секция микропроцессора
КР1804ВУ1,2	—	8	5	Блок управления
КР1804ВР1	—	8	5	Ускоренный перенос
КР1804ВУ3	32×8	8	5	Выбор адреса
КР1804ИР1	4п	8	5	Регистр Д-типа
830-К1883ИА0	8п	5	5	АЛБ
832-К1883РТ1	—	5	5	Память
832-К1883ВР2	16п	5	5	Расширитель арифметический
834-К1883ВА4	—	5	5	Адаптер

Наиболее распространенными являются БИС, изготовленные по технологии МДП. Возможны варианты этой технологии: *n*-МДП и *p*-МДП.

Последний вариант практически не используют, так как *n*-МДП обеспечивает более высокое быстродействие. В БИС, от которых не требуется повышенное быстродействие, перспективна КМДП-технология. По ней выполнены БИС типов К587 и КР588.

Таким образом, элементную базу микропроцессорных систем составляют БИС, каждая из которых реализует функцию структурного узла ЭВМ (процессор, оперативная память, постоянная память, узлы сопряжения с линиями связи и внешними устройствами и др.).

3.2. Структура микропроцессоров

По мере развития БИС стало возможным реализовать в одном кристалле функции отдельных блоков малой ЭВМ, а в дальнейшем даже перейти на простейшие однокристальные микроЭВМ. *Сверхбольшие интегральные схемы* (СБИС) позволяют реализовать в одном кристалле универсальный вычислительный модуль, т. е.

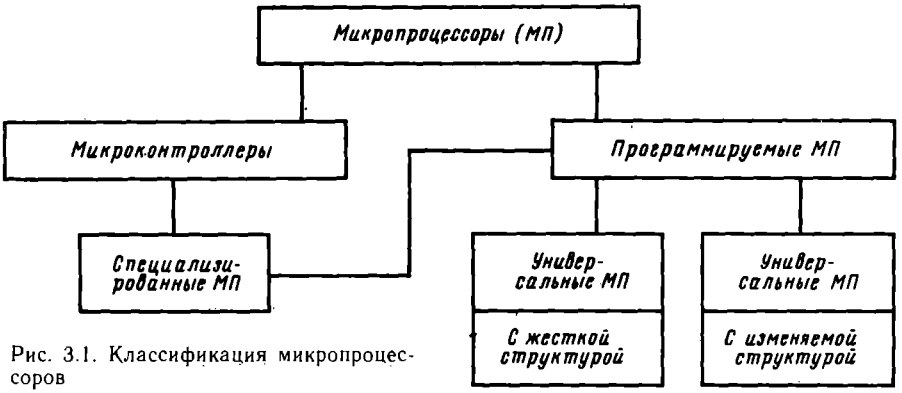


Рис. 3.1. Классификация микропроцессоров

центральный процессор ЭВМ и оперативную память достаточно большой емкости [15, 16, 17]

Такой модуль при достаточно высоком уровне надежности (интенсивность отказов $\lambda < 10^{-6}$ ч$^{-1}$) допускает работу по программе, составленной как на машинном языке (или в командах Ассемблера), так и на языках высокого уровня (BASIC, PASCAL и др.). Под термином *микропроцессор* в инженерной практике понимают именно указанный модуль, который не требует технического обслуживания и допускает подключение типовых внешних устройств, включая устройства связи с управляемым объектом.

Использование микропроцессоров и микроЭВМ целесообразно в первую очередь для непосредственного управления достаточно

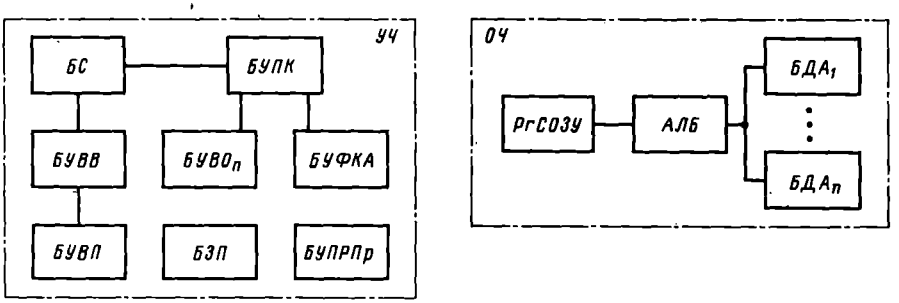


Рис. 3.2. Обобщенная логическая структура типового микропроцессора:
УЧ — управляющая часть; *ОЧ* — операционная часть; *БС* — блок синхронизации; *БУПК* — блок управления последовательностью команд; *БУВВ* — блок управления вводом-выводом; *БУВП* — блок управления виртуальной памятью; *БУВОп* — блок управления выполнением операций; *БУФКА* — блок управления формированием кодов адресов; *БЗП* — блок защиты памяти; *БУПРПр* — блок управления прерыванием работы процессора; *РгСОЗУ* — регистровое сверхоперативное запоминающее устройство; *АЛБ* — арифметический логический блок; *БДА*$_1$—*БДА*$_n$ — блоки дополнительной арифметики

сложными объектами, где дополнительные затраты на микропроцессорные аппаратные средства и устройства их сопряжения с объектом окупаются повышением качества управления объектом и, в частности, возможностью полной автоматизации управления путем составления и реализации соответствующих управляющих программ.

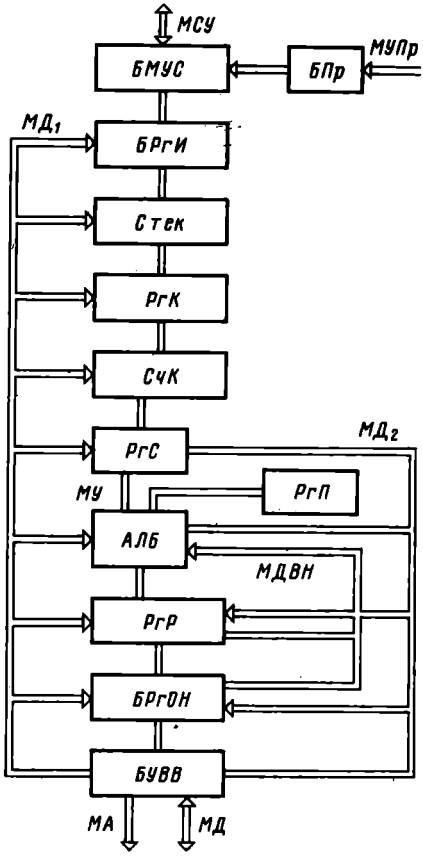


Рис. 3.3. Развитый однокристальный микропроцессор:
БМУС — блок местного управления и синхронизации; *БПр* — блок прерываний; *БРгИ* — блок индексных регистров; *РгК* — регистр команд; *СчК* — счетчик команд; *РгС* — регистр когда состояния; *РгП* — регистр триггеров признаков; *АЛБ* — арифметический логический блок; *РгР* — регистр результатов; *БРгОН* — универсальные регистры общего назначения; *БУВВ* — блок управления вводом-выводом; *МУ* и *МД* — магистрали управления и данных

Для микропроцессора характерна алгоритмически полная система команд, которая позволяет реализовать любой алгоритм. Наряду с аппаратной реализацией простейших команд (микроопераций) широко используется программный способ реализации сложных команд в виде микропрограмм, что позволяет расширить содержание команд до десятков и даже сотен операций.

Основное отличие микропроцессора от микроЭВМ состоит в том, что микроЭВМ содержит набор внешних устройств, позволяющих реализовать ввод-вывод данных. Это позволяет использовать микроЭВМ в режиме вычислительной машины при вводе данных с носителя информации и соответственно при выводе результатов на носитель.

Микропроцессор обычно содержит минимальный набор функциональных узлов для обработки информации без внешних устройств.

Простейшие микропроцессоры не содержат полного набора операций для реализации вычислительных алгоритмов.

Однокристальное исполнение ведет к простейшим системам программирования на основе микроопераций при наличии унифицированных входных и выходных каналов для данных и команд. Лишь большие микропроцессоры могут быть дополнены внешними устройствами и системами сопряже-

ния до состава оборудования нормальной микроЭВМ.

Далее под термином *микропроцессор* будем понимать универсальный процессор, применяемый для решения широкого круга задач управления в соответствии с составленной программой. Кроме универсальных, известны и специализированные микропроцессоры (рис. 3.1), которые рассчитаны на строго определенный объект управления.

Логическая структура типового микропроцессора включает управляющую часть *УЧ* (рис. 3.2) и операционную часть *ОЧ*, причем *УЧ* генерирует сигналы управления в соответствии с программой, а *ОЧ* в соответствии с этими командами реализует обработку числовой информации.

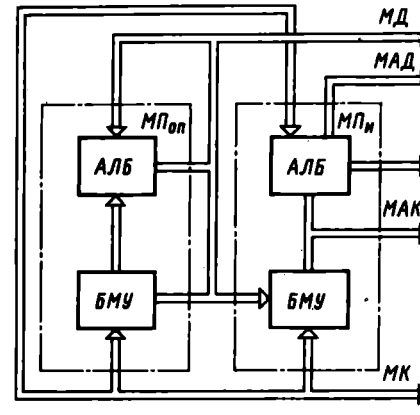


Рис. 3.4. Распределенный микропроцессор:
МД, МАД, МК, МАК — магистрали данных, адресов данных, команд, адресов команд; $МП_{оп}$, $МП_{и}$ — микропроцессоры операционный и индексный

Рассмотрим более детально логическую структуру типового микропроцессора (рис. 3.3). Характерные для ЭВМ блоки нанизаны на общую внутреннюю информационную магистраль, показанную двойной линией. Она содержит магистрали вводимых данных $МД_1$, выводимых данных $МД_2$, внутренних данных *МДВН* и сигналов управления *МСУ*

Основными блоками микропроцессора являются арифметико-логический *АЛБ*, блоки управления вводом-выводом *БУВВ* и местного управления *БМУС*. Остальные узлы соответствуют аналогичным узлам типовой ЭВМ и служат для последовательной реализации команд программы, реализации переходов и индексации.

Возможны и более сложные варианты логических структур, например, с распределенными функциями, т. е. с несколькими параллельными каналами. Могут быть предусмотрены отдельные информационные магистрали для данных *МД* (рис. 3.4), т. е. для чисел, и для команд *МК*.

Возможно архитектурное расширение структуры и за счет выделения специализированных микропроцессоров, называемых информационными контроллерами *ИК* (рис. 3.5), которые осуществляют связь центрального микропроцессора с устройствами ввода *УВв* и вывода *УВыв*, связь блоков *БСД* и *БСИМ* с датчиками *Д* и исполнительными элементами *ИМ* объекта управления *ОУ* Такая структура фактически соответствует структуре управляющей микроЭВМ, работающей в реальном масштабе времени.

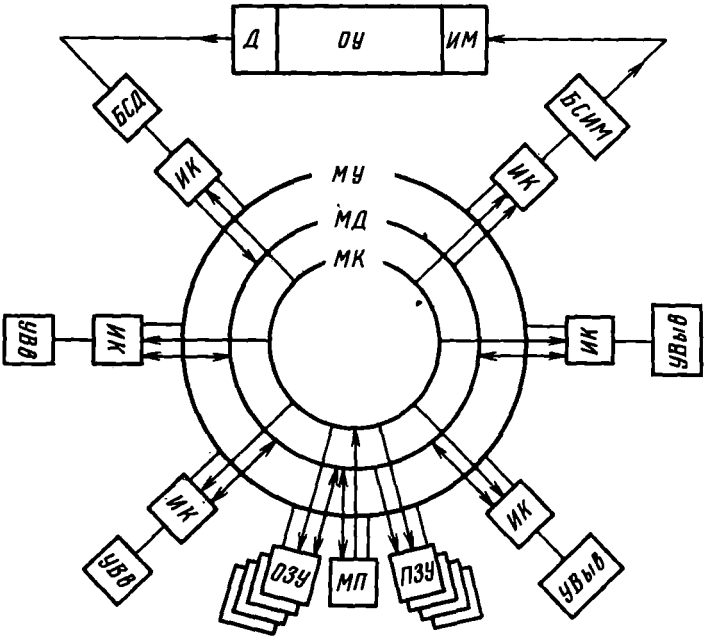


Рис. 3.5. Управляющая микропроцессорная система:
МУ, МД, МК — магистрали управления, данных, команд; *МП* — микропроцессор; *ИК* — информационный контроллер; *УВв, УВыв* — устройства ввода-вывода; *ОЗУ* — оперативное запоминающее устройство; *ОУ* — объект управления; *Д* — датчики; *ИМ* — исполнительные механизмы; *БСД, БСИМ* — блоки сопряжения с датчиками и исполнительными механизмами

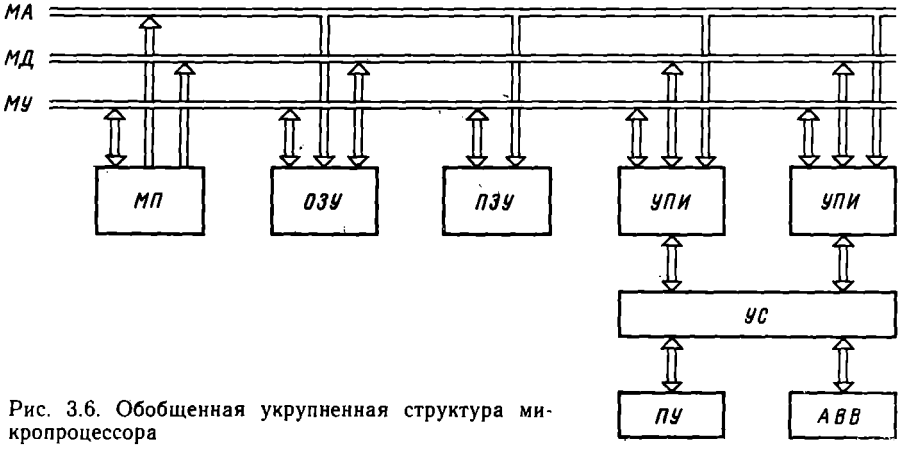


Рис. 3.6. Обобщенная укрупненная структура микропроцессора

Пользователь получает обычно готовый *МП* и может выбрать необходимое количество блоков *ОЗУ* и *ПЗУ*, а также интерфейсы, обеспечивающие сопряжение с объектом и внешними устройствами. Эти задачи целесообразно рассмотреть на конкретном микропроцессоре, который является базовым для решения задач тягового электроснабжения и электрической тяги. В качестве такого принят микропроцессорный комплект К580.

3.3. Микропроцессорный комплект К580. Структура и программирование

В комплекте К580 процессор вместе с узлами управления реализован в виде отдельной БИС с фиксированной разрядностью, составляющей 8 бит. Система команд в виде микропрограмм встроена в основную и частично в отдельную БИС памяти. Такой комплект типичен для микропроцессоров массового выпуска и в то же время по своим возможностям, стоимости, показателям надежности он наиболее подходит для применения на тяговых подстанциях и на электроподвижном составе.

Комплект К580 включает микропроцессорную БИС типа КР580ИК80, которая представляет собой однокристальный микропроцессор (8 бит) с однонаправленной шестнадцатиразрядной магистралью *МА* и двунаправленной восьмиразрядной магистралью *МД* (рис. 3.7) Магистраль управления *МУ* позволяет передавать шесть входных и шесть выходных сигналов. Арифметические и логические операции выполняются с восьмиразрядными числами, а также с числами двойной длины (16 разрядов).

Функциональные назначения внешних выводов комплекта следующие: A_0—A_{15} (рис. 3.8) — магистраль адресов *МА*, обеспечивающая доступ к любой из 2^{16} ячеек оперативной или внешней памяти; $Д_0$—$Д_7$ — магистраль данных *МД*, по которой осуществляется двунаправленный обмен данными с памятью; *С* — выход, по которому формируется сигнал в начале каждого цикла; *П* — выход, по которому указывается готовность процессора к приему данных; *ОЖД* — вывод, с помощью которого микропроцессор находится в состоянии ожидания к вводу информации или приему внешней команды; $\overline{Зп}$ — вывод, по которому данные выдаются процессором в магистраль *МД* и могут быть приняты внешним

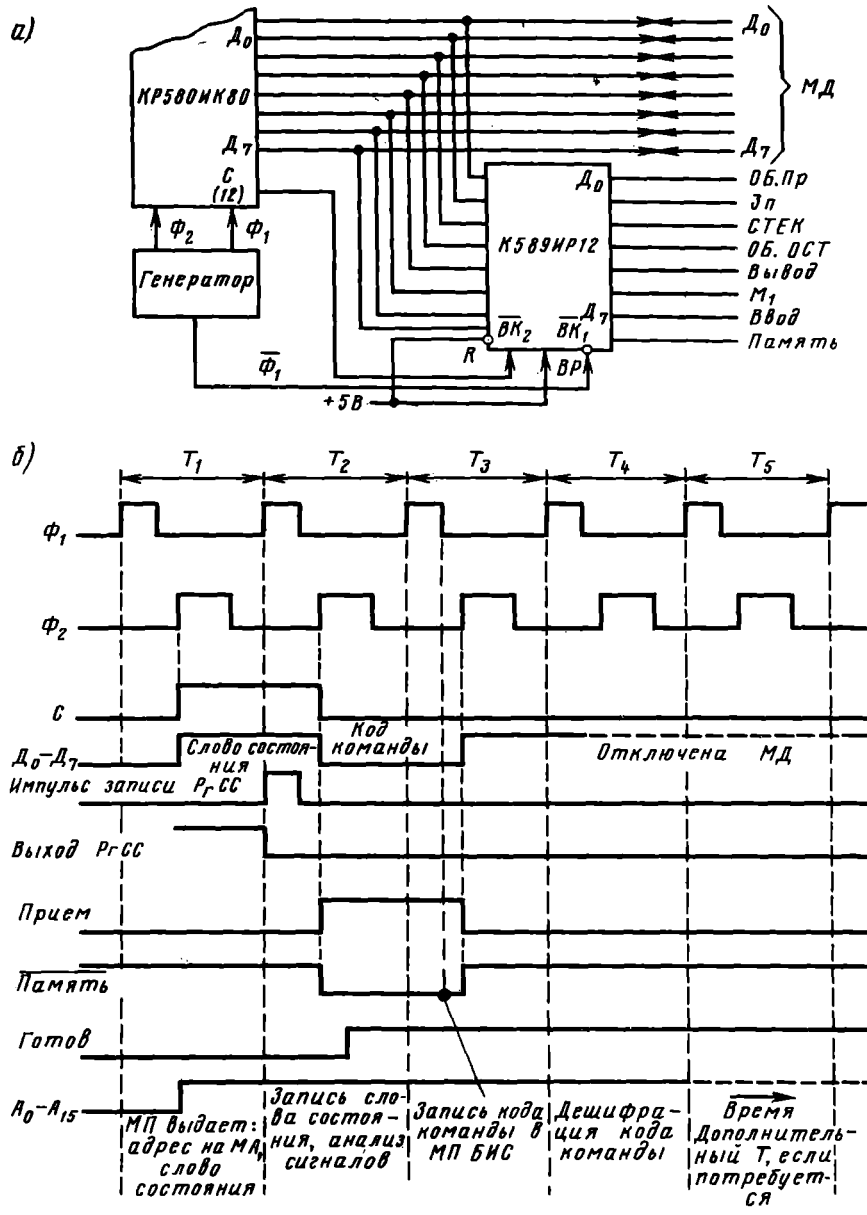


Рис. 3.7 Структурная схема микропроцессора КР580(*а*) и диаграмма передачи слова состояния (*б*)

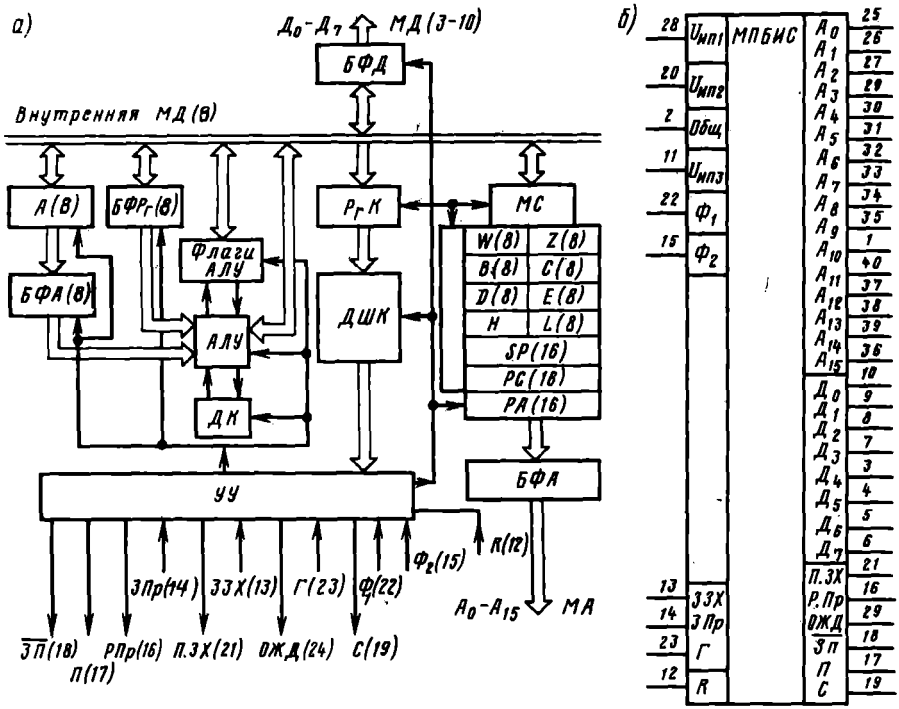


Рис. 3.8. Логическая схема микропроцессора КР580 (*а*) и обозначения его внешних соединений (*б*)

устройством; *П.ЗХ* — вывод, по которому подтверждается захват, т. е. дается ответ на сигнал *З.З.Х* (запрос захвата), когда *МД* и *МА* находятся в состоянии высокого сопротивления; *Р.П*$_р$ — указывает на разрешение прерывания, когда возможен прием запроса прерывания извне; *Г* — сигнализирует о готовности внешнего устройства к обмену информацией с микропроцессором (при *Г*=0 микропроцессор переходит в состояние ожидания *ОЖД*); *З.Пр* — вход для подачи сигнала запроса прерывания от внешних устройств; *R* — вход для обнуления микропроцессора; $Ф_1$ и $Ф_2$ — входы для подачи тактовых сигналов.

Микропроцессор КР580 содержит следующие функциональные блоки: шестнадцати- и восьмиразрядные регистры *W*, *Z*, *B*, *C*, *D*, *E*, *H*, *L*; программный счетчик *PC*, указатель стека *SP*. При этом восьмиразрядные регистры общего назначения *B*, *C*, *D*, *E*, *H*, *L* могут быть использованы как по отдельности, так и в парах. Счетчик *PC* содержит текущий адрес памяти, к которому обращается рабочая программа. Содержимое *PC* меняется после исполнения очередной команды. Указатель *SP* содержит адрес памяти, начиная с которого

ее можно применять для хранения и восстановления содержания программно доступных регистров. Регистры *W* и *Z* не относятся к программно доступным.

Арифметико-логический блок включает в себя восьмиразрядное *АЛУ*, схему десятичной коррекции *ДК*, аккумулятор *А* с буфером *БФА* и регистром *БФРг*. *АЛУ* реализует операции сложения и вычитания, логические операции и сдвиг кода. При этом один операнд всегда берется из *БФА*, а другой — из *БФРг*.

Регистр и дешифратор команд *РгК* и *ДШК* используются при выполнении очередной команды. При этом следует различать следующие нормы времени:

машинный такт *Т*, который соответствует периоду синхросигналов Φ_1 и Φ_2 (0,5—2 мкс);

машинный цикл *М* извлечения 1 байта информации из памяти или цикл времени выполнения элементарной команды, определяемой одним машинным словом, причем $M=(3\div5)\ T$;

время выполнения команды составляет $(1\div5)$ М.

Предусмотрено 10 типов машинных циклов: извлечение кода команды M_1; чтение данных из памяти; запись данных в память; извлечение из стека; запись в стек; ввод данных из внешнего устройства; запись данных во внешнее устройство; цикл обслуживания прерывания; останов; обслуживание прерывания в останове.

Первым при исполнении команды является цикл M_1. Очередной цикл указывают с помощью восьмиразрядного слова состояния, выдаваемого из *МД*.

Код любой команды представляет восьмиразрядное слово. Всего имеется 246 различных команд: команды переноса данных, арифметические, логические, команды передачи управления и работы со стеком.

Память процессора состоит из восьмиразрядных ячеек, каждая из которых имеет шестнадцатиразрядный адрес. Емкость памяти равна 65536 байтов. Каждая команда имеет 1—3 байта.

3.4. Сопряжение микропроцессора с объектом

Основными вариантами использования микропроцессора являются следующие:

в составе вычислительной микроЭВМ (рис. 3.9, *а*), имеющей устройство ввода *УВв* с перфорированного носителя или с клавиатуры и устройство вывода *УВыв* на печать или цифровую индикацию;

в составе оперативной справочно-информационной системы (рис. 3.9, *б*), когда микропроцессор *МП* получает информацию непосредственно с датчиков *Д*, установленных на объекте управления *ОУ*, а результаты вычислений выдает на устройство вывода *УВыв*, выполненное в форме экрана дисплея или цифрового индикатора;

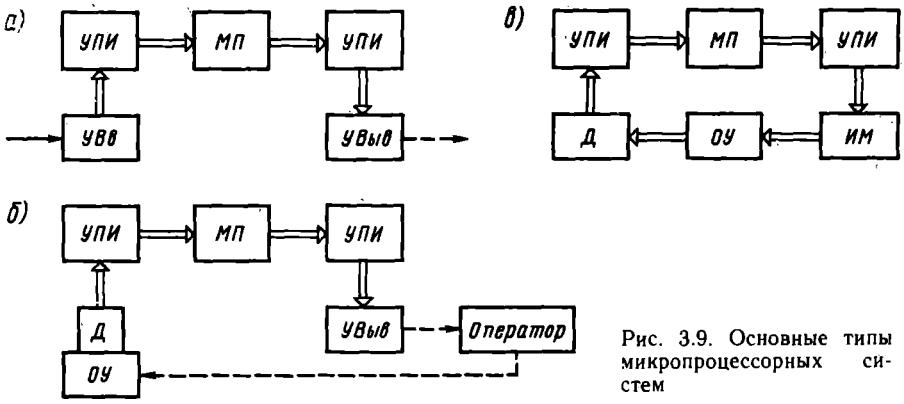


Рис. 3.9. Основные типы микропроцессорных систем

в составе оперативной управляющей системы (рис. 3.9, *в*), когда процессор *МП* принимает информацию с датчиков *Д*, а результаты вычислений непосредственно используются для управления объектом *ОУ* при помощи предусмотренных для этой цели исполнительных механизмов *ИМ*.

С точки зрения управления устройствами электрической тяги и электроснабжения интерес представляют два последних режима, которые характеризуются использованием *МП* в реальном масштабе времени (т. е. время решения задачи в *МП* увязано со скоростью протекания реального процесса в *ОУ*). Ниже перечислены характерные типы задач, решаемых в справочно-информационных системах [8].

Централизованный контроль оборудования тягового подвижного состава предполагает сбор диагностической информации о состоянии оборудования поезда по сигналам диагностических датчиков. МП осуществляет периодический опрос состояний датчиков и сравнивает их показания x_1, x_2, x_3, ..., x_i с предельно допустимыми значениями диагностируемых параметров x_1^*, x_2^*, x_3^*, ..., x_i^* (рис. 3.10). В качестве диагностируемых параметров используют такие параметры, которые однозначно характеризуют исправное или неисправное состояние узла оборудования. Например, для системы рессорного подвешивания электровоза диагностическими параметрами могут быть амплитуда и частота колебаний, декремент затухания колебаний, спектральная характеристика. Некоторые из этих величин могут быть непосредственно замерены датчиками, а некоторые приходится вычислять в МП по показаниям датчиков.

Для контроля электрооборудования могут использоваться различные электрические параметры (к.п.д., коэффициент мощности, форма электрического сигнала), а также температура нагрева электрических машин и аппаратов. Разработаны и опробованы достаточно сложные системы диагностирования тяговых электродвигателей, контакторного оборудования, аккумуляторных батарей.

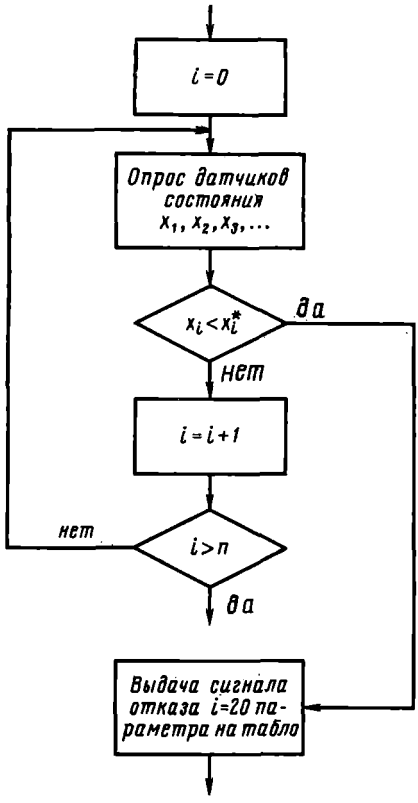


Рис. 3.10. Схема алгоритма централизованного контроля состояний датчиков

Диспетчерский контроль в системе тягового энергоснабжения широко используется в практике эксплуатации электрических железных дорог. Его сущность заключается в представлении информации о состоянии электрооборудования тяговых подстанций на табло, содержащем мнемоническую схему участка. В такой системе информационно-справочная ЭВМ обеспечивает обработку информации, выбор режимов работы (отключение части агрегатов, задание ступеней регулирования напряжения и т. д.). Расчетные данные выдаются на экран дисплея в виде рекомендаций для энергодиспетчера. Микропроцессор в такой системе может быть использован для регистрации действий энергодиспетчера и для регистрации защитных отключений, а также для непрерывного учета нагрузок оборудования.

Автоведение поезда представляет собой классическую задачу оптимального управления, которая детально рассмотрена ниже. Обычно эта задача представляет собой расчет такого режима движения поезда, при котором обеспечивается его прибытие на конечный пункт в заданное время при минимальном расходе энергии. При этом МП непосредственно воздействует на цепи управления тягой и торможением поезда.

Автоматическое управление объектами тягового электроснабжения также предполагает непосредственное воздействие МП на цепи управления всех силовых выключателей и разъединителей, которые должны быть оборудованы соответствующими дистанционными приводами. Такой МП реализует функции защиты со сложными алгоритмами контроля уставок, устройств автоматического повторного включения и выбора оптимального числа работающих агрегатов.

С учетом разнообразия применения управляющих и информационных систем на базе МП целесообразно ввести понятие об универсальной управляющей микроЭВМ, содержащей типовой МП с

дополнительными устройствами, обеспечивающими: ввод данных объекта управления; воздействие на объект управления; ввод данных от оператора (с перфорированного носителя или с клавиатуры); вывод данных на регистрирующее устройство или на экран дисплея.

Управляющая система должна, как правило, иметь режим прерывания, когда процесс вычислений может быть прерван оператором для ввода нового задания или запроса, а также сигналами датчиков объекта. Функции связи МП с объектом и оператором реализуются с помощью дополнительных устройств, которые целесообразно рассмотреть на примере комплекта КР580.

Интерфейс КР580ВВ55 обеспечивает ввод-вывод информации по каналам *A, B* и *C* (рис. 3.11). Входы A_0, A_1 обеспечивают выбор одного из каналов *A, B, C*, а при помощи входа «сброс» производят начальную установку схемы. Интерфейс может быть использован для связи МП с любым внешним устройством (клавиатура, дисплей, внешняя память, датчики и исполнительные органы на объекте управления).

Блок приоритетного прерывания КР580ВН59 является многофункциональным программируемым устройством, формирующим запрос на прерывание МП и выдающим на МД команду CALL$<A_1>$ $<A_2>$ Возможны четыре режима прерываний, из которых наиболее простой характеризуется неизменными приоритетами.

Микросхема КР580ВТ57 обеспечивает прямой обмен данными между памятью и любым внешним устройством, минуя МП. С по-

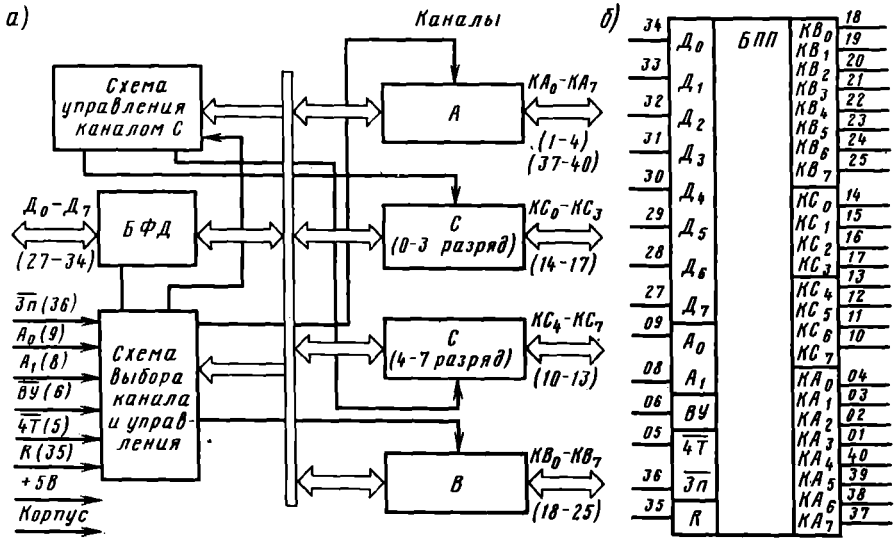


Рис. 3.11. Структурная схема сопряжений микропроцессора КР580 с объектом управления (*а*) и обозначения внешних соединений (*б*)

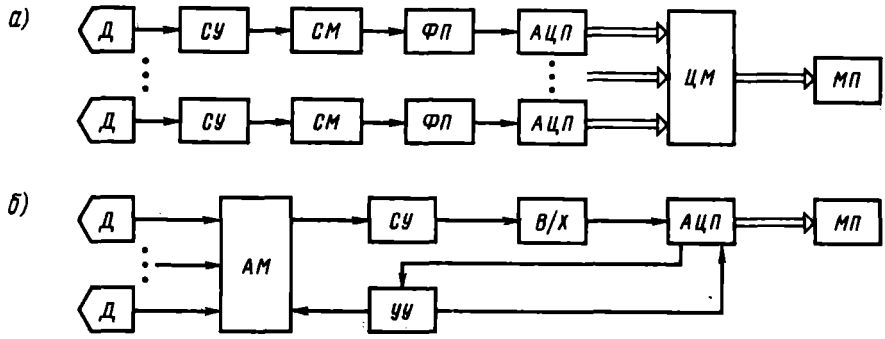


Рис. 3.12. Структурная схема сопряжения микропроцессора *МП* с аналоговыми датчиками *Д* при использовании индивидуальных АЦП в каждом канале (*а*) и общего АЦП (*б*)

мощью этой микросхемы можно непосредственно вводить данные с датчиков объекта в ячейки памяти, не прерывая работу МП по выполнению основной расчетной программы.

Таймер КР580ВИ53 вырабатывает по трем каналам временные интервалы согласно программе. По каждому каналу может быть задан интервал времени определенной продолжительности либо импульсная последовательность с заданным числом периодов.

Универсальный приемопередатчик КР580ВВ51 обеспечивает двунаправленный обмен данными МП с любым внешним устройством. В синхронном режиме стандартная скорость обмена составляет 56 тыс.бит/с, а в асинхронном 9,6 тыс.бит/с при длине слова 5—8 разрядов. При передаче всегда осуществляется преобразование кодов, потому что МП типа КР580 рассчитан на работу с параллельными восьмиразрядными кодами, а внешние устройства обычно принимают или выдают последовательные коды с другой длиной слова.

Конкретные примеры структур параллельного и последовательного устройств сопряжения микропроцессора *МП* с объектом управления даны на рис. 3.12 применительно к общему случаю аналоговых датчиков *Д*, установленных на объекте. Преобразование информации с датчиков осуществляется селективными усилителями *СУ*, селективными модуляторами *СМ*, фильтрами-преобразователями *ФП*, а также аналого-цифровыми преобразователями *АЦП*.

На рис. 3.12 вверху показан пример многоканальной системы с цифровым мультиплексором *ЦМ* на входе микропроцессора *МП*, а на рис. 3.12 внизу — пример одноканальной системы, когда

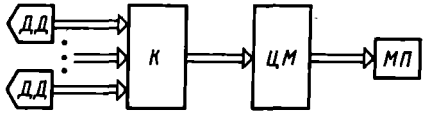


Рис. 3.13. Сопряжение микропроцессора *МП* с дискретными датчиками *ДД*

сигналы всех датчиков проходят через общий канал с усилителем *СУ*, масштабным блоком *В/Х* и *АЦП*. Коммутация датчиков в схеме (рис. 3.12, *б*) осуществляется аналоговым мультиплексором (коммутатором) *АМ* при помощи устройства управления *УУ*.

Ввод двоичных сигналов с дискретных датчиков осуществляется по более простой схеме через устройство кодирования *К* (рис. 3.13) и мультиплексор *ЦМ*. Системы, показанные на рис. 3.12 и 3.13, обычно применяются совместно при работе на общий *МП*.

3.5. Решение задач справочно-информационного характера

При решении подобных задач микропроцессор (МП) не выполняет непосредственно функций управления, а лишь осуществляет сбор, обработку и хранение информации, поступающей с датчиков. Работа МП происходит в режиме реального времени. Если МП обнаруживает сверхнормативное отклонение любого из контролируемых параметров, то он сигнализирует об этом оператору или (в бортовых системах) машинисту поезда.

В качестве примера рассмотрим использование МП типа К580 в качестве машины централизованного контроля оборудования электровоза. При этом в *МП* (рис. 3.14, *а*) через блок ввода подключены выходы датчиков (Д). В качестве примера указаны характерные типы датчиков: температуры *Д* (θ), вибраций *Д(а)*, скорости *Д(υ)*, частоты вращения *D(n)*, электрических величин тока, напряжения, частоты *Д(I)*, *Д(V)*, *Д(f)*. При этом сигналы аналоговых датчиков Д вводят через аналого-цифровой преобразо-

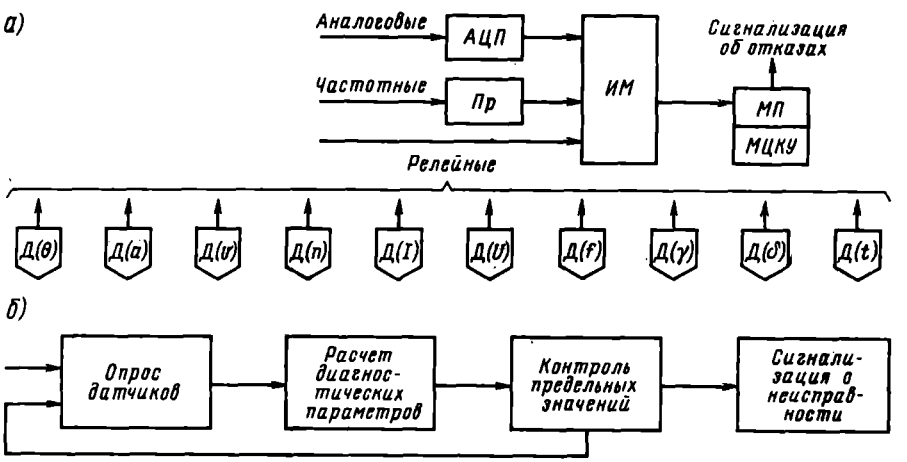


Рис. 3.14. Микропроцессорная система технического диагностирования электроподвижного состава

ватель *АЦП*, сигналы частотных датчиков — через преобразователь «частота-код» *Пр*, а сигналы релейных, т. е. дискретных, датчиков — непосредственно.

При регулировании тяговых электроприводов с тиристорными преобразователями необходимо замерять более сложные параметры — угол коммутации γ, угол запаса δ, схемное время восстановления $t_{сх}$, а также электрические и временны́е параметры импульсов сложной формы. Эти задачи измерения решаются при помощи специальных датчиков, построение которых представляет собой отдельную достаточно сложную техническую задачу. Часть этих датчиков может быть выполнена по принципу непосредственного измерения соответствующей физической величины, например, напряжения, тока или температуры. Некоторые величины приходится вычислять по показаниям непосредственных датчиков. Например, ускорение поезда обычно вычисляют как разницу скоростей движения, замеренных в фиксированные моменты времени.

В ряде случаев приходится использовать сложные вычислительные алгоритмы. Например, температуру нагрева вращающихся частей тягового двигателя (обмотки якоря, коллектора) непосредственно замерить, например термопарой, очень сложно. Поэтому в практике ее вычисляют при помощи тепловой модели, в которой исходными параметрами являются нагрузка двигателя́ и скорость воздушного потока в функции времени. Такой датчик использован, например, в системе регулирования мотор-вентиляторов на электровозе.

Обычно любой датчик дает на выходе (посылает в ЭВМ) измеряемую величину в виде электрического сигнала (напряжение или ток) аналогового типа. Выход такого датчика соединяют со входом ЭВМ посредством АЦП.

Другой тип датчиков — это датчики релейного типа, генерирующие выходной сигнал двоичного типа. Их выход может находиться в одном из двух состояний: 0 или 1. Используются также импульсные датчики частотного типа, у которых частота импульсов пропорциональна измеряемой величине (скорости, напряжению и др.). Считывание показания частотного датчика микропроцессором осуществляется непосредственно (без АЦП) путем счета числа импульсов за фиксированный промежуток времени, отмеряемый таймером.

Датчики обычно не входят в состав микропроцессорных комплексов, но микропроцессоры всегда рассчитаны на подключение к ним датчиков и имеют для этого интерфейсы и АЦП. Далее будем полагать́, что комплекс датчиков уже выбран и нужно лишь составить алгоритм решения задачи.

Для задач любого типа характерны следующие этапы: опрос состояний датчиков, выполнение расчета, т. е. обработка показаний датчиков, выдача решения. Для рассматриваемой задачи диагностического контроля эти этапы могут быть расписаны более подробно

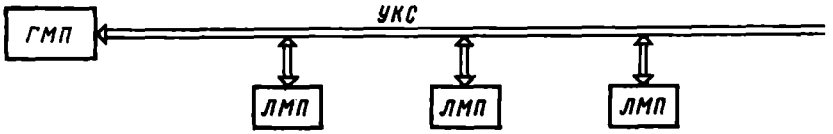


Рис. 3.15. Структура двухступенчатых микропроцессорных систем подвижного состава

на блок-схеме (рис. 3.14, *б*). Их программирование осуществляется с помощью типового набора команд МП.

Определенная трудность решения этой задачи для многосекционного электровоза или электропоезда связана с передачей информации от датчиков по межсекционным или междувагонным связям. Если для каждого датчика предусматривать свою линию связи, то число их будет значительным (несколько сотен); кроме того, датчики обычно не допускают подключения к ним длинных линий.

Специфика подвижного состава состоит в том, что в большинстве случаев приходится предусматривать двухступенчатую структуру микропроцессорных систем. В такой структуре имеется главный микропроцессор *ГМП* (рис. 3.15), соединенный с локальными процессорами *ЛМП*, которые собирают информацию с датчиков и передают ее в *ГМП* по уплотненным каналам связи *УКС*. Таким образом, информация от всех датчиков, работающих на *ЛМП*, передается по одной паре проводов последовательными кодами. *ЛМП* выполняет здесь функции концентратора информации, а также производит предварительную обработку информации.

В ряде случаев возможен многопроцессорный комплекс однородного типа, когда все идентичные секции электровоза оборудованы одинаковыми *МП* (рис. 3.16). В такой системе один из *МП*, находящийся в головной секции, переключается в режим *ГМП*, а остальные работают в режиме *ЛМП*. Переключение может выполняться переключателем электропневматического тормоза ЭПТ, имеющимся на всех видах электроподвижного состава и обозначенным на рисунке *Г/Л*.

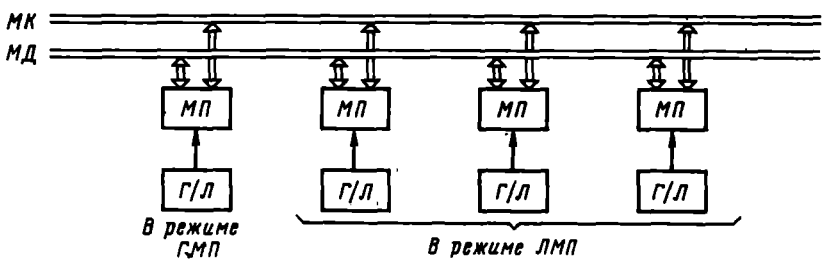


Рис. 3.16. Структура одноступенчатых микропроцессорных систем подвижного состава

3.6. Управление электроподвижным составом с помощью микроЭВМ

При помощи типовых микропроцессорных комплексов могут быть реализованы алгоритмы управления электроподвижным составом. Задачи управления, решаемые при помощи микропроцессоров, могут быть разделены на следующие четыре группы [18].

Непосредственное управление тиристорным преобразователем в системе питания тягового двигателя. При этом микропроцессор в каждом такте работы преобразователя задает моменты генерации управляющих импульсов для включения тиристоров преобразователя. Расчет фазы импульсов должен быть выполнен в каждом периоде до момента его генерации, т. е. для такого расчета отводится время менее одного периода (в ряде случаев менее полупериода) работы преобразователя. Например, для непосредственного управления выпрямителем (или инвертором) электровоза переменного тока полный цикл расчета фазы импульса управления не должен превышать 0,01 с. Задачи этого типа специфичны из-за ограничений не только по времени, но и по реализуемым алгоритмам, которые определяются типом преобразователя (выпрямитель, зависимый или автономный инвертор, импульсный преобразователь и т. д.).

Регулирование тока тяговых двигателей. Различают стабилизацию тока, когда $I = I_з = \text{const}$ (рис. 3.17, *а*), и программное регулирование, когда $I = I_з(v)$. Уставку $I_з$ задают в функции скорости v, соответствующей ограничению по сцеплению. В обоих

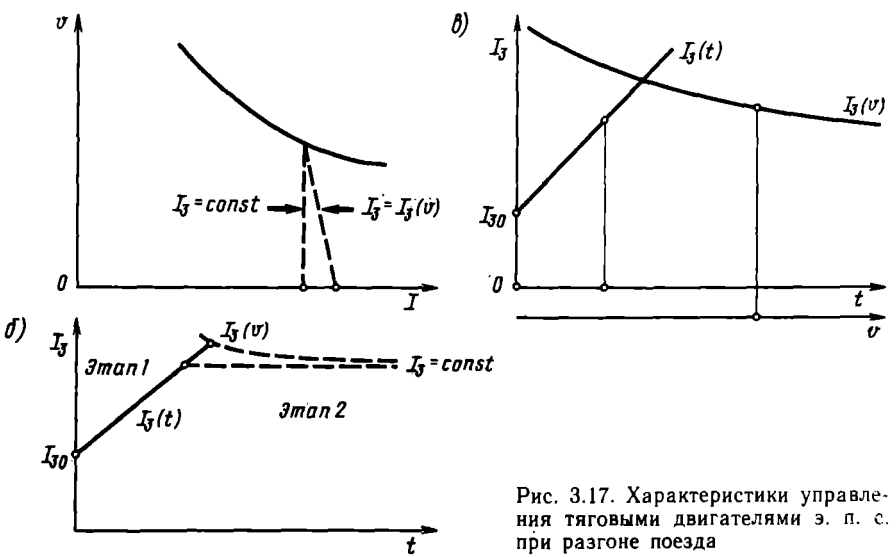


Рис. 3.17. Характеристики управления тяговыми двигателями э. п. с. при разгоне поезда

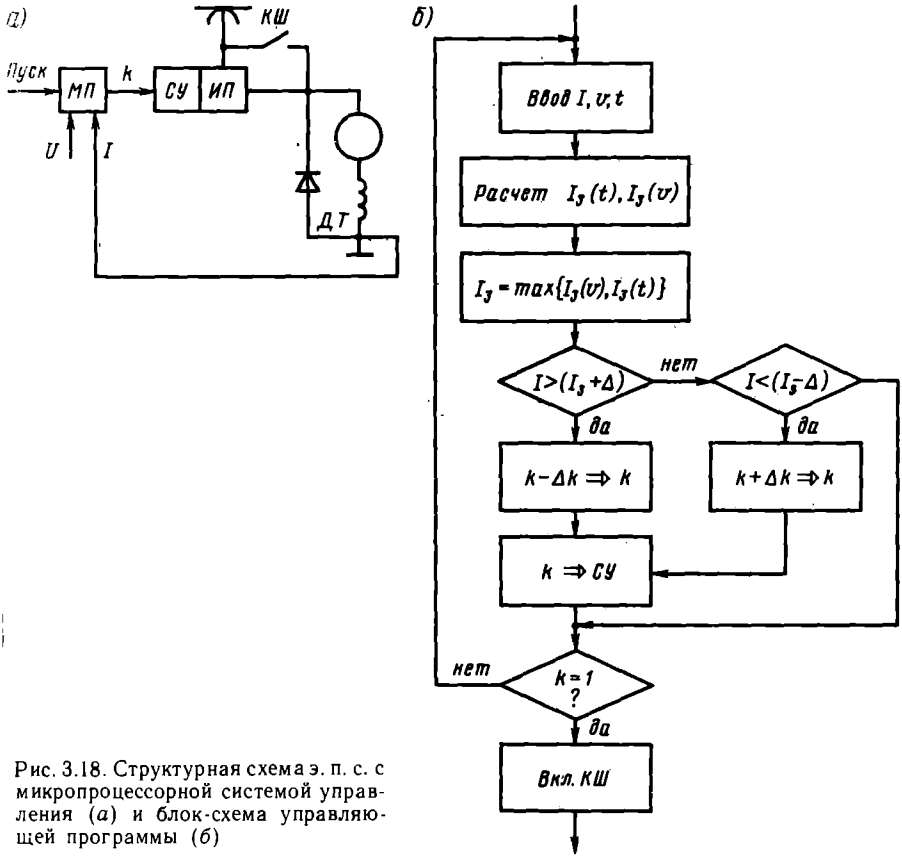


Рис. 3.18. Структурная схема э. п. с. с микропроцессорной системой управления (*а*) и блок-схема управляющей программы (*б*)

случаях при $I_з = \text{const}$ и $I_з = I_з(v)$ на начальном этапе (т. е. сразу после включения тяговых двигателей) реализуют постепенное увеличение тока по линейному или экспоненциальному закону в функции времени (рис. 3.17, *б*). Таким образом, в общем случае МП должен реализовать законы программного регулирования (рис. 3.17, *в*)

$$\left.\begin{aligned} I_з &= I_{з0} + at \\ I_з &= I_з(v) \end{aligned}\right\}$$

Из них следует, что система задания тока выбирает наибольшее значение $I_з$ из двух и использует его в качестве уставки в системе регулирования.

Рассмотренные функции определения $I_з$ реализуются в блоке *МП* (рис. 3.18). Блок *СУ* реализует управление импульсным преобразователем *ИП* путем увеличения или уменьшения его коэффициента заполнения *k*.

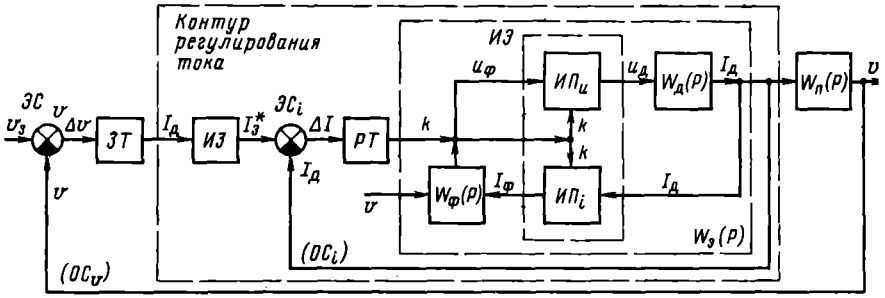


Рис. 3.19. Двухконтурная система стабилизации скорости электропоезда

Блок *ИП* фиксирует конец регулирования, когда $k = 1$, и реализует выход из программы, включая перед этим контактор *КШ*, шунтирующий импульсный преобразователь *ИП*.

Стабилизация скорости поезда. Ее осуществляют путем выбора такого значения силы тяги, которое уравновешивало бы силу

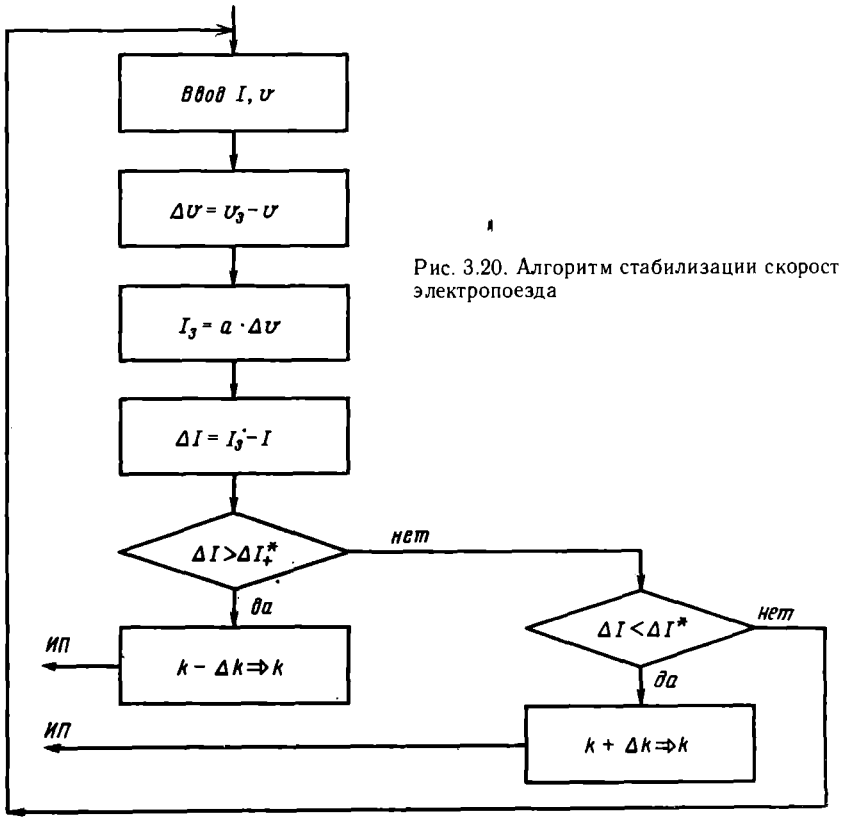


Рис. 3.20. Алгоритм стабилизации скорости электропоезда

сопротивления движению. С точки зрения теории автоматического регулирования такая система (рис. 3.19) относится к классу систем регулирования по отклонению, причем регулируемой величиной является скорость v. В функции рассогласования по скорости определяют заданный ток $I_з$ и далее стабилизируют его путем изменения k. В контуре регулирования тока имеются релейный элемент *РТ* и интегрирующий элемент *ИЭ*. Программная реализация этой системы приведена на рис. 3.20.

Автоведение поезда. Оно предполагает реализацию такого режима, когда при соблюдении заданного (графикового) времени прибытия на следующую станцию обеспечивается минимальный расход энергии из всех возможных режимов движения с заданным временем прибытия [19]. Автоведение целесообразно рассмотреть на простейшем примере движения поезда метро по перегону *АБ* (рис. 3.21, *а*). Оптимальный режим всегда включает фазу разгона *ФР* с максимальным ускорением, фазу выбега *ФВ* и фазу целевого торможения *ФЦТ* с остановкой у платформы *Б*. Режим движения может при этом регулироваться только положением точки *О* отключения тяговых двигателей. Для этого предварительно рассчитывают параметры графикового режима, т. е. положение точки *О* (координата $v_{о-г}$ или $S_{о-г}$) в случае следования поезда точно по графику. Это достигается отправлением поезда со станции *А* в момент времени $t_{А\text{-}г}$ и прибытием на станцию *Б* в момент времени $t_{Б\text{-}г}$. При наличии отклонения фактического времени отправления $t_{А\text{-}ф}$ от графикового $t_{А\text{-}г}$ нужно для соблюдения условия $t_{Б\text{-}ф}=t_{Б\text{-}г}$ изменить положение точки *О*. При $t_{А\text{-}ф}>t_{А\text{-}г}$ увеличивают $v_{о\text{-}ф}$ или $S_{о\text{-}ф}$, а при $t_{А\text{-}ф}<t_{А\text{-}г}$ соответственно уменьшают их. Практически это выполняют при помощи корректировочной кривой $\Delta S=\varphi(\Delta t)$ (рис. 3.21, *б*) на примере корректировки по пути s. Управляющая программа, обеспечивающая выполнение заданного режима, приведена на рис. 3.22.

На заключительной фазе движения *ФЦТ* реализуют целевое торможение, т. е. обеспечивают соответствие скорости поезда заданной кривой целевого торможения, генерируемой в функции пути s. Другими словами, заданную скорость определяют как $v_з(s)$ и осуществляют закон регулирования $v=v_з(s)$ путем воздействия на систему электрического торможения поезда методами, рассмотренными выше.

Применение микропроцессоров для целей регулирования связано с учетом двух особенностей: дискретность представления всех величин в цифровой форме и задержка времени на расчет управляющего воздействия, т. е. на работу программы. Дискретность приводит к ступенчатому изменению регулируемых величин (коэффициента заполнения k, напряжения на двигателях $V_д$ и тока I). Максимальные ступени изменения тока I имеют место при малых скоростях движения, т. е. при трогании поезда. Расчетным и опытным путем установлено, что минимальная разрядность кодов

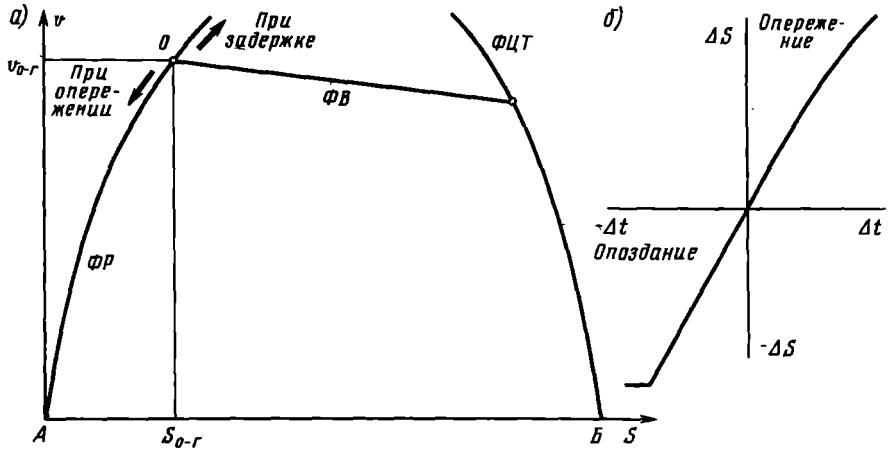


Рис. 3.21. Типичная диаграмма движения электропоезда метрополитена (*а*) и корректировочная кривая (*б*) для определения момента перехода на выбег

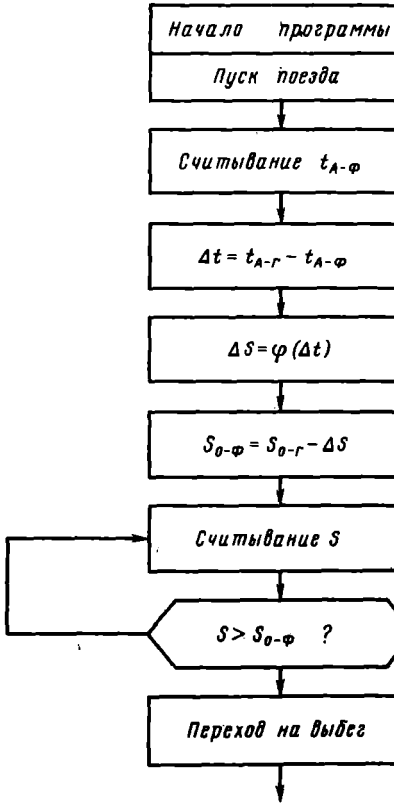


Рис. 3.22. Алгоритм перехода на выбег

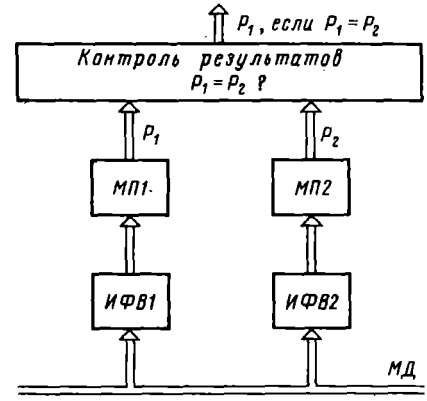


Рис. 3.23. Отказоустойчивая микропроцессорная система

в микропроцессоре должна быть равна 8, что хорошо согласуется с возможностями комплекса КР580. При этом дискретность изменения тока и силы тяги не превышает 2—3% их номинальных значений.

Быстродействие микропроцессора определяется затратами времени на каждый тип операции и сложностью программы. Обычно быстродействие может быть лимитирующим для задач первого и второго типов. В этом случае нужен специальный расчет по методике, рассмотренной в [10].

Еще одна проблема связана с показателями надежности. Обычно при использовании МП для управления поездом ставится задача исключения опасных отказов. Это означает, что при неправильном расчете управляющего воздействия оно не должно быть передано исполнительным элементам. Для этого обычно применяют сдвоенные или строенные комплексы со сравнением результатов, полученных в параллельных каналах. К такому типу относятся комплекс SIMIS фирмы «Сименс» (ФРГ) и система автоведения ATOMIC (Япония) для скоростных электропоездов.

В сдвоенной системе блокируется выдача результата при несовпадении P_1 и P_2 (рис. 3.23). В строенной системе возможна выдача результата при совпадениях по двум каналам. Практически при использовании МП на электровозах и электропоездах предусматривают также запасной вариант ручного неавтоматического управления.

В системах подобного рода, которые обычно называют отказоустойчивыми, приходится дублировать не только вычислительные и управляющие блоки, но также и измерительно-функциональные входы *ИФВ1, ИФВ2*. Возможно также и дублирование контрольных устройств, установленных на выходе микропроцессоров *МП1* и *МП2*.

Контрольные вопросы

1. Опишите структуру типового микропроцессора.
2. Как подключают микропроцессор к объекту управления?
3. Назовите последовательность выполнения программы в типовом микропроцессоре.
4. Какие задачи по автоматизации устройств тягового электроснабжения могут быть решены при помощи микропроцессора?
5. Какие задачи по автоматизации управления движением поездов могут быть решены при помощи микропроцессора? Какой эффект может быть при этом получен?
6. Где помещается программа, исполняемая микропроцессором?
7. Как контролируют правильность работы микропроцессора при управлении ответственными объектами железнодорожного транспорта?
8. Для каких целей объединяют несколько микропроцессоров в единую систему? Приведите примеры структур таких систем.
9. Каковы функции интерфейса и адаптера?
10. Какую роль играет разрядность слова микропроцессора?

Глава 4

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ БЕЙСИК

4.1. Общие сведения о языке Бейсик

Бейсик является алгоритмическим диалоговым языком. Название происходит от английского слова basic — основной. Однако есть и другое объяснение названия языка. Возможно, оно является аббревиатурой английских слов Beginner's All purpose Symbolic Instruction Code, означающих «многоцелевой язык символических инструкций для начинающих».

Бейсик, будучи разработанным в 1965 г., первоначально был ориентирован на решение небольших вычислительных задач в диалоговом режиме и не представлял трудности в освоении даже начинающим программистам.

В настоящее время — это самый распространенный язык программирования для персональных ЭВМ. Он достаточно прост, универсален и применяется для решения широкого круга задач. При относительно небольшом объеме исходной информации он может быть применен (и применяется) для решения инженерных задач в области проектирования, эксплуатации и управления устройствами электроснабжения и электроподвижного состава электрифицированных железных дорог. Он входит также в число языков, используемых для программирования микропроцессорных систем.

К сожалению, Бейсик не стандартизован и в настоящее время существует изрядное множество диалектов этого языка. В связи с этим затруднительно дать изложение Бейсика для любого персонального компьютера. Поскольку в вузах СССР наиболее распространены ЭВМ «Искра-226», будем в основном ориентироваться на особенности Бейсика именно этой ЭВМ.

Следует также иметь в виду, что Бейсик, как и другие алгоритмические языки, непрерывно развивается и совершенствуется, появляются новые версии языка. Поэтому, приступая к программированию задачи на ЭВМ, следует ознакомиться с конкретной версией языка, используемой на данной ЭВМ и в данный момент.

Отметим, что освоение языка Бейсик закладывает хорошую основу для изучения других более мощных алгоритмических языков, которые широко распространены сегодня (например, Фортран и ПЛ/1), а также сравнительно новых, таких, как АДА и СИ.

Бейсик, являясь сравнительно молодым языком, в значительной мере своим происхождением обязан идеям Фортрана. Программы с языка Бейсик почти один к одному могут быть переведены на языки Фортран и ПЛ/1. Этим объясняется возможность быстрого

освоения новых языков и перевода старых программ на новый язык с наименьшими потерями времени.

Знание хотя бы нескольких алгоритмических языков весьма полезно, так как позволяет обоснованно выбрать самый подходящий язык и ЭВМ для программирования и эффективного решения конкретной технической задачи.

Бейсик является алгоритмическим языком, т. е. языком, предназначенным для записи алгоритмов, причем в удобной, привычной для человека форме. Ни одна ЭВМ не может выполнять программу, записанную на алгоритмическом языке. ЭВМ понятен лишь свой машинный язык. Поэтому необходимо осуществлять перевод (трансляцию) исходной программы на Бейсике в программу, состоящую из машинных команд. Эту операцию выполняют специальные программы, называемые транслляторами Существуют два типа трансляторов — интерпретаторы и компиляторы.

Интерпретаторы каждое предложение алгоритмического языка переводят в команды ЭВМ по мере чтения программы и тут же выполняют, после чего переходят к чтению следующего предложения. При таком способе трансляции в памяти ЭВМ в процессе выполнения программы хранятся и сам транслятор, и программа на исходном (алгоритмическом) языке, причем при повторном выполнении предложения программы происходит и повторный перевод. Недостатками такого способа трансляции являются неэффективное использование памяти ЭВМ и замедление работы программы. Однако в этом случае легко исправляются ошибки, допущенные в программе.

При переводе программ с языка Бейсик на язык ЭВМ, как правило, используются интерпретаторы. Использование интерпретатора освобождает пользователя от необходимости дополнительного ознакомления с совокупностью программ, обеспечивающих вычислительный процесс и называемых операционными системами.

Компиляторы переводят программу целиком. Присутствие компилятора в памяти ЭВМ при выполнении программы излишне. Не происходит и многократный перевод одних и тех же предложений. В этом преимущество компиляторов. Исправление допущенной ошибки в оттранслированной программе практически недоступно пользователю. Поэтому при обнаружении ошибки необходимо внести соответствующие коррективы в исходную программу и вновь повторить трансляцию.

Компиляторы используются при работе на больших ЭВМ с объемными программами и сложными программными комплексами, измеряемыми тысячами, десятками и сотнями тысяч команд. В этом случае используются и более мощные языки Фортран, ПЛ/1 и др. Выполнение программ на больших ЭВМ неизмеримо сложнее, поскольку требует от пользователя дополнительно знаний специального языка, называемого языком управления заданиями (ЯУЗ), и особенностей используемых операционных систем.

4.2. Описание языка

Алфавит языка Бейсик представляет собой множество символов, включающих в себя следующие подмножества:

латинский алфавит из 26 прописных букв;

русский алфавит из 31 прописной буквы;

десятичные цифры от Ø до 9 (для отличия нуля от буквы О его перечеркивают);

шестнадцатеричные цифры Ø, 1, ..., 9, A, B, ..., F;

знаки препинания: (точка), , (запятая), (точка с запятой), (апостроф), « (кавычки), (двоеточие);

знаки арифметических операций: + (плюс), — (минус), × (умножение), / (деление), ↑ (возведение в степень);

знаки операций отношения: < (меньше), = (равно), > (больше) и производные от них ⩽ (меньше или равно), <> (не равно), ⩾ (больше или равно), которые набираются на клавиатуре терминалов нажатием двух соответствующих клавиш;

прочие знаки: ! (восклицательный знак), ? (вопросительный знак), # (решетка), ⊙ (знак денежной единицы, на ЭВМ может иметь другое изображение), % (процент), ( ) круглые скобки.

Под *данными* понимают информацию, подлежащую обработке при помощи программ. В Бейсике используются следующие виды данных: константы, переменные и массивы. В более мощных алгоритмических языках существуют виды данных более высокого уровня (например, структуры в языке ПЛ/1 и Фортран ОЕ).

Под *константами* понимаются величины, значения которых не изменяются в процессе работы программы. В Бейсике различают константы трех типов: целые, вещественные и символьные. Целые константы представляют собой последовательность десятичных цифр, которой предшествует знак. После последней цифры записывается символ целочисленности (%); например, 5%, 1989%, -356%.

Вещественные константы могут быть представлены в двух формах: в форме с фиксированной точкой и в форме с плавающей точкой. Во втором случае порядок числа отделяется от мантиссы символом Е. При этом порядок может содержать максимально две десятичные цифры и знак, а мантисса должна обязательно начинаться со значащей цифры (нормализованная мантисса). Ниже приведен пример записи чисел в различной форме:

Математическая запись числа	Запись на языке Бейсик	
	В форме с фиксированной точкой	В форме с плавающей точкой
3,2	3,2	Ø.32E1
0	Ø	Ø
1,753000000	1.753ØØØØØØ	Ø.1753E+1Ø
—3298,5	—3298.5	—Ø.32985E4
25,3548	25.3548	Ø.253548E2
0,00000758	Ø.ØØØØØ758	Ø.758E —5

Из данного примера следует, что запись констант в форме с фиксированной точкой отличается от общепринятой в русской математической литературе лишь использованием десятичной точки вместо десятичной запятой. Наличие самой точки не обязательно. Если точке предшествует нуль, его можно опускать (например, писать не Ø.ØØØØØØ75, а .ØØØØØØ75). Применение в данном случае запятой приводит к ошибке. В форме с плавающей точкой удобно записывать очень большие или очень маленькие константы. Числа, не превышающие 6—7 цифр, лучше записывать в форме с фиксированной точкой. Деление констант на целые и вещественные позволяет программисту управлять точностью вычислений и распределением памяти. Целые и вещественные числа представляются в памяти по-разному. Целые занимают меньше места и вычисления с их участием выполняются быстрее.

Диапазон изменения целых констант $0 \leqslant |N| \leqslant 7999$.

Диапазон изменения вещественных констант $10^{-99} \leqslant |N| \leqslant (10 - -10^{12})\,10^{+99}$

В памяти ЭВМ хранится число π, при вычислениях оно записывается в виде # PI.

Следующим типом констант, с которыми может работать Бейсик, являются символьные константы. Символьная константа представляется цепочкой символов алфавита языка, заключенной в кавычки, например, «АВС», «ДА», «НЕТ», «ИВАНОВ И. И., ГР.ЭЛ-321».

Под *переменными* понимаются величины, изменяющиеся в процессе работы программы и принимающие различные допустимые значения. Для обозначения переменных во всех алгоритмических языках используют имена (идентификаторы). В Бейсике различают переменные трех типов аналогично типам констант: целые, вещественные и символьные. Они в свою очередь подразделяются на простые переменные и элементы массивов.

Имя (идентификатор) простой переменной составляется на Бейсике из одной буквы латинского алфавита или из буквы и цифры. Начинаться оно должно всегда с буквы, например, V, I, RØ, L, I1, V1.

Признаком целой переменной в обозначении является символ %, например, A1%, N%.

Идентификатор простой символьной переменной составляется по тем же правилам, но за последним символом имени следует знак денежной единицы ⊙, например, A⊙, X6⊙.

Идентификатор простой целой переменной может совпадать с идентификатором простой действительной или символьной переменной, но интерпретатор будет рассматривать их как разные имена. В общей сложности на Бейсике можно составить 3·286 имен простых переменных. Однако следует помнить, что общее число различных переменных в одной программе не должно превышать 208. Диапазон изменений простой целой и действительной переменной таков же, как и диапазон изменения соответствующих констант.

Стандартная длина символьной переменной составляет 16 символов. Максимальная длина символьной переменной — 253. Способы задания длины символьной переменной будут рассмотрены далее.

Большие удобства в программировании возникают при объединении некоторых данных в структуру под одним именем. Это облегчает работу программиста и экономит память ЭВМ. Такие объединения называются *массивами*. Каждый элемент массива однозначно определяется своим порядковым номером. Данные объединяются в массивы по каким-либо общим для всех элементов признакам, например: массив значений токов электровозов, находящихся на рассматриваемом участке электрифицированной линии; массив значений напряжения на шинах тяговой подстанции, полученный в результате 100 замеров; массив коэффициентов при неизвестных в системе линейных алгебраических уравнений и т. д. Первые два массива (массив токов и массив напряжений) являются одномерными (или векторами). Массив коэффициентов при неизвестных является двухмерным, он содержит строки и столбцы. Двухмерные массивы еще называют матрицами. Двухмерный массив — массив максимальной размерности, допустимой в Бейсике. В Фортране, например, разрешены семимерные массивы.

Итак, исходя из предыдущих объяснений можно дать следующее определение массива. Массив — это совокупность объединенных по физическому смыслу данных, имеющих одно общее имя (идентификатор), причем в этой совокупности каждый элемент однозначно определяется значениями индексов.

Правила составления имен массивов на Бейсике не отличаются от правил составления имен простых переменных. В одной программе допускается применение одних и тех же обозначений для простых переменных и массивов, но одномерные и двухмерные массивы не должны иметь одинаковых идентификаторов.

Порядковый номер элементов массива отсчитывают от единицы, а не от нуля (как это делается в ряде других диалектов Бейсика), например:

V — массив напряжений тяговой подстанции;

I — массив токов электровозов;

A — массив коэффициентов при неизвестных в системе линейных алгебраических уравнений;

тогда V(5) — пятый элемент в массиве V;

I(30) — тридцатый элемент в массиве I;

A(4,5) — коэффициент, расположенный в 4-й строке и в 5-м столбце, т. е. при 5-м неизвестном в 4-м уравнении.

Так же, как константы и простые переменные, массивы и элементы массивов в Бейсике подразделяются на три типа: целые, вещественные и символьные. Идентификаторы целых и символьных массивов сопровождаются специальными символами — % и ⊙.

Структура языка Бейсик основана на строках, т. е. основной единицей информации в языке является строка. Программная строка начинается с номера, за которым следует один или несколько операторов языка.

Оператор — это допустимое в данном языке предписание, задающее определенные действия в программе по обработке информации. Оператор состоит из имени оператора и параметров.

Операторы в строке отделяются друг от друга символом двоеточия (:).

Длина строки не должна превышать 240 символов, т. е. три строки экрана (по 80 символов в каждой).

Операции над данными составляют суть работы любой программы. Действуя по заданному алгоритму, программа преобразовывает входные данные в конечный результат.

Данные (константы, переменные и массивы), объединенные знаками операций, образуют выражение. Если в выражение входят числовые данные, то такое выражение называется *арифметическим*. Ниже даны примеры записи арифметических выражений:

Математическая запись	На языке Бейсик
$2\cos(x-\frac{\pi}{6})$	2*COS(X-#PI/6)
I_f^2R	I(J)↑2*R
$e^{\lvert x-y\rvert}+\lvert x-y\rvert^{x+y}$	EXP(ABS(X-Y))+ +ABS(X-Y)↑(X+Y)

Вычисление арифметических выражений происходит в соответствии с определенной субординацией операций. Сначала выполняются операции, заключенные в круглые скобки, и вычисляются функции. Затем производится возведение в степень, далее равноценные операции — умножение и деление и за ними сложение и вычитание. Равноценные операции (умножение и деление или сложение и вычитание) выполняются в естественном порядке слева направо. Если же имеется несколько операций возведения в степень (при отсутствии скобок), то они выполняются справа налево, например:

Математическая запись	На языке Бейсик
$y^{\sin^2 x}$	Y↑SIN(X)↑2
$(y^{\sin x})^2$	(Y↑SIN(X))↑2

В примерах арифметических выражений были использованы функции SIN(X), COS(X), EXP(X) и ABS(X). Перечисленные функции и ряд других, которыми располагает Бейсик, называются встроенными. Они встроены в программу — транслятор.

Перечень элементарных математических функций Бейсика приведен в табл. 4.1.

Примеры записи и результаты вычисления некоторых встроенных функций:

Запись	Результаты вычислений
SIN(#PI/6)	.5
COS(.75↑2+.1)	.7884569
ARCCOS(SIN(#PI/3))	.523598
ABS(SIN(3.49))	.342020
ABS(—7.539)	7.539
INT(7.3*2)	14
INT(—7.3*2)	—15
SGN(255.5)	1
SGN(∅)	∅
EXP(—3568.7)	—1
EXP(1)	2.7182818

Особо следует остановиться на функции RND (см. табл. 4.1). Эту функцию называют генератором псевдослучайных чисел. Выдаваемые им числа подчиняются закону равномерной плотности в промежутке от ∅ до 1. Если в качестве аргумента функции RND задать ∅, то последовательность псевдослучайных чисел будет повторяться при каждом повторном запуске программы. Повторения последовательности не будет происходить, если в качестве аргумента задать 1.

Арифметические выражения на Бейсике (так же как и на других алгоритмических языках) записываются в одну строчку, а не в две в отличие от математических формул, в которых используются операции деления и возведения в степень. Для этого используются лишние по сравнению с формулой скобки. Например, формула

$$\frac{1+\operatorname{sh}^2(x+y)}{\left|x-\dfrac{2y}{1+x^2y^2}\right|}\,x^{|y|}$$

на Бейсике должна быть записана в следующем виде:

```
(1+((EXP(X+Y)-EXP(-(X+Y)))/2↑2)/(ABS(X-2*Y/(1+
+X↑2*Y↑2))*X↑ABS(Y).
```

В приведенной математической формуле используется функция — гиперболический синус sh x, отсутствующая в описанной версии Бейсика в составе встроенных функций. При программировании гиперболических функций следует пользоваться известными в математике формулами, выражающими их через экспоненциальные функции, которые в свою очередь имеются в составе встроенных функций:

$$\operatorname{sh} x=\frac{e^x-e^{-x}}{2};\quad \operatorname{ch} x=\frac{e^x+e^{-x}}{2};$$

$$\operatorname{th} x=\frac{e^x-e^{-x}}{e^x+e^{-x}}.$$

Правила записи встроенных функций таковы, что за именем функции обязательно следует открывающая круглая скобка, далее аргумент и закрывающая скобка. Поэтому приведенные ниже примеры записи функций недопустимы и приводят к ошибке:

Математическая запись	Запись на Бейсике ошибочная	верная
$\sin^2 x$	SIN↑2(x)	SIN(X)↑2
$(e^x)^2$	EXP↑2(X)	EXP(X)↑2 или EXP(2∗X)

Следует также иметь в виду, что запись SIN(X+Y)↑2 или (SIN(X+Y))↑2 соответствует математической формуле $\sin^2(x+y)$, а формуле $\sin(x+y)^2$ соответствует следующая запись на Бейсике: SIN((x+y)↑2).

Для вычисления корней, отличных от степени 2, следует выражать их в виде степени. Например, выражение $\sqrt[3]{|y-2|}$ запишется в Бейсике в виде ABS(Y−2)↑(1/3).

При выполнении операции возведения в степень следует следить за тем, чтобы основание степени было положительным. Отрицательное основание можно возводить только в целую степень.

Таблица 4.1

Выполняемая операция	Имя функции	Математическое определение	Примечание
Вычисление синуса	SIN(X)	$y=\sin x$	Аргумент может быть выражен как в градусах, так и в радианах
Вычисление косинуса	COS(X)	$y=\cos x$	
Вычисление тангенса	TAN(X)	$y=\mathrm{tg}\,x$	
Вычисление арксинуса	ARCSIN(X)	$y=\arcsin x$	Результат вычисления может быть получен как в градусах, так и в радианах
Вычисление арккосинуса	ARCCOS(X)	$y=\arccos x$	
Вычисление арктангенса	ARCTAN(X)	$y=\mathrm{arctg}\,x$	
Задать случайное число между 0 и 1	RND(X)	—	Аргумент следует задавать в виде 0 или 1
Вычисление абсолютного значения	ABS(X)	$y=\lvert x\rvert$	
Выделение целой части числа	INT(X)	$y=$ (знак x)n, n — наибольшее целое $n\leqslant\lvert x\rvert$	
Присвоение знака	SGN(X)	$y=\begin{cases}1, & \text{при } x>0\\ 0, & \text{при } x=0\\ -1, & \text{при } x<0\end{cases}$	
Вычисление натурального логарифма	LOG(X)	$y=\ln x$	
Вычисление показательной функции	EXP(X)	$y=e^x$	
Вычисление квадратного корня	SQR(X)	$y=\sqrt{x}$	

Таблица 4.2

Имя функции	Выполняемая операция	Примечание
STR	Выборка любых символов в символьной переменной	В аргументе указывается имя переменной, номер начального символа и число символов
LEN	Определение числа символов в символьной переменной	Конечные пробелы игнорируются
NUM	Определение количества символов числа в символьной переменной	Учитываются предшествующие числу и следующие за ним пробелы и знак
POS	Определение положения первого символа в символьной переменной, удовлетворяющего заданному условию. Фиксируется порядковый номер символа	Если такой символ не обнаружен, функция POS принимает значение, равное нулю
VAL	Преобразование двоичного (шестнадцатеричного) значения первого символа символьной переменной или константы в десятичное число	Функция VAL обратна оператору BIN

Операции над символьными данными производятся при помощи встроенных строковых функций. К таким операциям относятся, например, выборка любых символов из символьной переменной (извлечение части строки), подсчет числа символов в символьной переменной, преобразование данных одного типа в другой. Перечень встроенных строковых функций приведен в табл. 4.2.

Если символьная переменная A⊙ стандартной длины имеет значение МИКРОПРОЦЕССОР, то результатом выполнения функции STR(A⊙, 6, 9) будет ПРОЦЕССОР. Число символов в функции STR указывать не обязательно. Если оно опущено, то символы извлекаются от первого заданного символа до конца цепочки.

Результатом выполнения функции LEN(A⊙) будет число 14, а функции NUM(A⊙) — число 2. Результатом выполнения функции POS(A⊙ = «П») будет число 6, так как символ П является шестым символом символьной переменной A⊙.

Результатом выполнения функции VAL (A⊙) будет число 77 Первым символом переменной A⊙ является символ М, которому соответствует шестнадцатеричный код ED или двоичный код 1110 1101, что соответствует числу 237 в десятичной системе счисления.

В Бейсике допустимы и другие операции над символьными данными, такие, например, как логические операции (дизъюнкция, конъюнкция, исключающее ИЛИ), операции сдвига и ряд других. Эти

операции выполняются на операторном уровне и будут рассмотрены ниже.

Условное выражение — это еще один тип выражения, допустимого в языке Бейсик. В общем случае оно представляет собой два арифметических выражения, связанных знаками операций отношения ($<$, $\leqslant$, $=$, $\neq$, $>$, $\geqslant$). В качестве составляющих (операндов) условного выражения могут выступать также и символьные данные (переменные и константы). Проверка на совпадение и несовпадение символьных переменных и констант не требует пояснения. Выполнение условий «больше» и «меньше» определяется в соответствии со степенью старшинства символов, которая устанавливается порядком их следования ($A<B<C$.....).

Пробел	0	A	Q	Ю	П	
!	1	B	R	А	Я	
"	2	C	S	Б	Р	
#	3	D	T	Ц	С	
¤	4	E	U	Д	Т	
%	5	F	V	Е	У	
,	6	G	W	Ф	Ж	
	7	H	X	Г	В	
(	8	I	Y	Х	Ь	
)	9	J	Z	И	Ы	
*		K		Й	З	
+		L		К	Ш	
,	<	M		Л	Э	
–	=	N		М	Щ	
	>	O		Н	Ч	
/	?	P		О		

Степень уменьшается (↑)

Степень уменьшается (←)

Оператор языка — основная минимальная логически законченная конструкция языка. Любой оператор Бейсика начинается со служебного слова, определяющего действие над операндами, которые следуют за именем оператора. К основным операторам Бейсика отнесен набор операторов, позволяющих писать простейшие неразветвленные линейные программы:

LET (пусть) — оператор присваивания;
DATA (данные) — блок данных;
READ (читать) — оператор чтения из блока данных;
INPUT (ввод) — оператор ввода данных с клавиатуры терминала в процессе выполнения программы;

RESTORE (восстанавливать) — оператор организации считывания из блока данных;
PRINT (печатать) — оператор вывода;
REM (сокращение от REMARK—комментарий) — невыполняемый оператор, служит для пояснения текста программы;
END (конец) — оператор окончания работы программы.

Конструкция оператора LET*:

$$\text{нс LET} a = b$$

где нс — номер строки; a — простая числовая или символьная переменная (элемент массива); b — арифметическое выражение (если a — цифровая переменная) или константа (если a — символьная переменная).

Примеры записи оператора присваивания (служебное слово LET в ряде версий Бейсика допускается не писать):

а) мощность однофазного потребителя

$$P = UI[\cos\varphi - \cos(2\omega t - \varphi)];$$

запись на Бейсике

```
1∅P=U∗I∗(cos(V)−cos(2∗W∗T−V))
```

б) формула распределения потенциала в рельсовой цепи

$$\varphi_x = RIe^{-\frac{\alpha l}{2}} \operatorname{sh} \alpha\left(\frac{l}{2} - x\right),$$

где $R = \sqrt{v_р v_п}$, $\alpha = \sqrt{\frac{v_р}{v_п}}$;

запись на Бейсике

```
1∅M=SQR(R1∗R2)
2∅K=SQR(R1/R2)
3∅F=M∗I∗EXP(−K∗L/2)∗(EXP(K∗(L/2−X))−EXP(−K∗(L/
2−X)))/2
```

в) примеры с использованием символьных переменных

```
1∅A⊙="ДА"
4∅A⊙=HEX (41424321)
6∅B⊙=STR (A⊙, 2)
```

Оператор присваивания в Бейсике допускают присвоение значения правой части одновременно нескольким переменным, например,

```
X(5), Y, Z=2∗#PI∗F,
```

*При описании операторов в квадратных скобках записываются необязательные элементы конструкции оператора.

в результате чего 5-му элементу массива X, переменным Z и Y будет присвоено значение арифметического выражения $2\pi f$.

Конструкция оператора DATA:

нс DATA c_1, c_2, ..., c_n

где c_1, c_2, ..., c_n — цифровые или символьные константы.

Пример записи оператора DATA:

1∅ DATA 5.3, 17.5, #P1, .52E-8, «Фидер № 2», 1∅∅
2∅ DATA HEX (4142).

В ряде версий Бейсика в качестве констант c_1, c_2, ..., c_n допускается записывать арифметические выражения.

Конструкция оператора READ:

нс READ a_1, a_2, a_n

где a_1, a_2, ..., a_n — список ввода, переменные любого типа.

Оператор READ предназначен для выборки констант, хранящихся в операторе DATA. Между константами в операторе DATA и переменными в операторе READ устанавливается соответствие по порядку следования, т. е. каждой переменной из списка оператора READ присваивается очередная константа из блока данных до тех пор, пока всем переменным не будут присвоены значения. Если переменных в списке окажется больше, чем констант в блоке данных, то будет осуществлен переход к следующему блоку данных, а в случае его отсутствия будет выдано сообщение об ошибке. Если же оператор READ содержит меньше переменных, чем значений в операторе DATA, то следующий оператор READ начнет считывание с первого неиспользованного значения в операторе DATA. Блок данных можно рассматривать как устройство последовательного доступа.

Элементы списка переменных оператора READ и список констант оператора DATA должны быть согласованы и по типу, т. е. цифровой переменной должна быть поставлена в соответствие цифровая константа, а символьной переменной — символьная константа, например:

1∅ DATA 27.5, 3∅∅, «ПОДСТАНЦИЯ А», HEX (4B42)
2∅ READ U, I, A⊙, B⊙(1).

Действия операторов № 1∅ и № 2∅ эквивалентны действию следующей строки:

1∅ U=27.5: I=3∅∅: A⊙=«ПОДСТАНЦИЯ А»: B⊙(1)=HEX(4B42).

Конструкция оператора RESTORE:

нс RESTORE [b],

где b — арифметическое выражение (необязательный параметр $1 \leqslant b \leqslant 255$).

Оператор RESTORE позволяет повторно использовать данные блока данных. При этом целая часть арифметического выражения определяет номер константы, с которой начнется считывание данных. При отсутствии b считывание начнется с первого значения первого оператора DATA.

Примеры записи оператора:
10 RESTORE
110 RESTORE 10
60 RESTORE 2*X+1

Конструкция оператора PRINT

$$\text{нс ПRINT}\,[\,[u]\;\;[e_1\{;,\}e_2\{;,\}\cdots\cdots\{;,\}e^n]\,]$$

где u — код устройства вывода (05 — экран, 0C — печатающее устройство); e_1, e_2, ..., e_n — список вывода, содержащий переменные и константы любого типа, арифметические выражения, функцию TAB, строковые встроенные функции, разделенные запятой или точкой с запятой.

Оператор PRINT предназначен для индикации и распечатки информации.

Вывод констант по оператору PRINT осуществляется в том виде, в каком они представлены в теле оператора (символьная константа без кавычек).

Функция TAB в операторе PRINT используется для вывода информации на определенное знакоместо строки. Номер позиции печати от начала строки задается значением аргумента функции TAB, который в общем случае является арифметическим выражением. По своей конструкции функция TAB не отличается от встроенной функции. Результат вычисления арифметического выражения берется по модулю и выделяется целая часть, которая не должна превышать 255.

С помощью разделителей (запятой или точки с запятой) списка оператора PRINT можно управлять печатью. При наличии запятой информация выводится в зонном формате (длина зоны 16 символов). Наличие точки с запятой между элементами списка свидетельствует о том, что информация будет выводиться в уплотненном формате (с разрывом в один символ между элементами списка). Список вывода может быть пустым, тогда при выполнении оператора пропускается строка печати.

Конструкция оператора INPUT:

$$\text{нс INPUT}\;\;\overset{c}{\underset{\odot a}{}}\;\;a_1, a_2, \quad a_n$$

где c — символьная константа и a — вещественная переменная.

При выполнении оператора INPUT ЭВМ приостанавливает работу программы (происходит прерывание) и высвечивает на экране вопросительный знак (?). Пользователь должен ввести через запятые значения данных, соответствующих переменным в операторе INPUT; и нажать клавишу «перевод строки» (возврат каретки). После этого ЭВМ продолжит выполнение программы. Оператор INPUT является средством, позволяющим программировать диалог человека с ЭВМ.

Символьная константа в операторе INPUT может быть задана описанными ранее способами (в кавычках, при помощи HEX-функ-

ции, а также без кавычек) и используется для пояснений пользователю перед вводом данных.

Конструкция оператора REM имеет вид нс REMc (с — цепочка символов). Единственным ограничением для цепочки символов является запрет на символ двоеточие (:), поскольку он является разделителем операторов в строке.

При помощи оператора REM осуществляется документирование текста программы на Бейсике в соответствии с действующими требованиями [21] В комментариях рекомендуется указывать символическое имя программы, функциональное назначение, описание входных и выходных данных. Текст программы сопровождается пояснительными комментариями, описывающими логику программы. Для алгоритмических языков высокого уровня рекомендуется на 8—10 строк текста вставлять одну строку комментария.

Конструкция оператора END имеет вид нс END.

После выполнения оператора END выполнение программы прекращается, и на экране высвечивается количество свободной памяти в байтах.

Программа на Бейсике представляет логически завершенную совокупность (последовательность) строк. В отличие от Фортрана и ряда других алгоритмических языков высокого уровня, на которых программа строится по модульному принципу, программа на Бейсике представляет собой единое целое. Строки Бейсик-программы нумеруются. Для удобства последующей модификации программы нумерация обычно выполняется через 10 номеров. Номера строк выполняют несколько функций. Во-первых, только нумерованные строки включаются в программу (ненумерованные подлежат немедленному выполнению); во-вторых, номера определяют последовательность выполнения строк; в-третьих, номера выполняют роль меток строк, на которые может быть передано управление. Диапазон допустимых номеров изменяется от 1 до 9999.

В качестве примера составим программу для расчета индуктивного сопротивления 1 км контактного провода в контуре «несущий трос-контактный провод» по формуле

$$x_{кт} = 2\omega \cdot 10^{-4}\left(0{,}25 + l_{п}\frac{d_{кт}}{R_{к}}\right),$$

где $R_к$ — радиус контактного провода, равный $\sqrt{\frac{S_к}{\pi}}10^{-3}$, м; $s_к$ — сечение контактного провода, мм^2; $d_{кт}$ — эквивалентное расстояние между несущим тросом и контактным проводом, м; $\omega = 2\pi f$ — частота тока.

Физическим переменным, входящим в формулу, дадим имена (идентификаторы), допустимые на Бейсике, и представим их в таблице, которую будем называть таблицей идентификаций.

Отметим, что идентификаторы составляются таким образом, чтобы они как можно больше напоминали исходные физические переменные (это упрощает чтение и понимание текста программы):

Физические переменные	$x_{кт}$	ω	$d_{кт}$	$R_к$	$s_к$	f
Идентификаторы	X	W	D	R	S	F

Текст программы будет иметь следующий вид:

```
10 REM РАСЧЕТ ИНДУКТИВНОГО СОПРОТИВЛЕНИЯ КОНТАКТНО-
     ГО ПРОВОДА В КОНТУРЕ НЕСУЩИЙ ТРОС-КОНТАКТНЫЙ
     ПРОВОД
20 PRINT «ИВАНОВ И. И., ГР. ЭЛ-321, ВАР. № 10»
30 DATA 50, 1.3, 1E3
40 READ F, D, S
50 W=2*#PI*F
60 R=SQR (S/#PI)*1E-3
70 X=2*W*1E-4*(0.25+LOG (D/R))
80 PRINT ТАБ (5); «СОПРОТИВЛЕНИЕ КОНТАКТНОГО ПРОВОДА
     ХКТ=»; X; «Ом/км»
90 END
```

Для ввода данных в приведенной программе может быть использован оператор INPUT, тогда вместо операторов 30 и 40 следует записать один 30 INPUT «ВВЕДИТЕ ЧАСТОТУ, РАССТОЯНИЕ МЕЖДУ НЕСУЩИМ ТРОСОМ И КОНТАКТНЫМ ПРОВОДОМ И СЕЧЕНИЕ КОНТАКТНОГО ПРОВОДА», F, D, S.

В этом случае программа получается универсальной, поскольку становится независимой от значений исходных данных и пригодной для многовариантных расчетов.

Запуск программы осуществляется при помощи оператора RUN.

Конструкция оператора RUN имеет вид [нс]RUN[a], где a — номер строки, с которой начинается работа программы.

Чаще всего оператор RUN применяется без нс и параметра a. В этом случае выполнение программы начинается со строки с наименьшим порядковым номером, всем цифровым переменным присваивается значение нуля, а символьным — значение пробела. При использовании конструкции RUN a переменные сохраняют значения, полученные при последнем выполнении программы.

По окончании работы приведенной программы на экране появляются результаты в виде: ИВАНОВ И. И., ГР. ЭЛ-321, ВАР № 10
СОПРОТИВЛЕНИЕ КОНТАКТНОГО ПРОВОДА ХКТ=.3575692.

При использовании для ввода данных оператора INPUT на 30 операторе произойдет прерывание работы программы и на экране появится надпись:
ВВЕДИТЕ ЧАСТОТУ, РАССТОЯНИЕ МЕЖДУ НЕСУЩИМ ТРОСОМ И КОНТАКТНЫМ ПРОВОДОМ И СЕЧЕНИЕ КОНТАКТНОГО ПРОВОДА?

После ввода данных в виде 50, 1.3, 1E3 и нажатия клавиши «перевод строки/возврат каретки» работа программы будет продолжена и получен приведенный выше результат.

4.3. Разветвленные и циклические процессы. Структурное программирование

Программы, приведенные в предыдущем параграфе, а следовательно, и реализованные в них алгоритмы могут быть названы *линейными*. В таких программах заранее строго определена последовательность выполнения операторов. В отличие от линейных программ и процессов различают *разветвленные* и *циклические* процессы, в которых вычисления могут развиваться в различных направлениях (ответвляться как ветви у дерева) в зависимости от выполнения (или невыполнения) условий, проверяемых в различных точках алгоритма. В этих точках реализуются операции, аналогичные мыслительной деятельности человека. В них принимается решение о дальнейших действиях в зависимости от условий, сложившихся на данный момент времени.

Циклические процессы — это те же разветвляющиеся процессы, в которых одна из ветвей многократно замыкается на пройденную часть алгоритма для повторных вычислений до момента выполнения определенного условия.

Все алгоритмические языки высокого уровня (в том числе и Бейсик) располагают средствами для организации разветвленных и циклических программ, реализующих сложнейшие вычислительные, имитационные, графические и прочие алгоритмы.

Все большее применение находит реализация идей структурного программирования, получившего развитие в трудах ряда авторов [3, 15, 22, 23]

По своему существу *структурное программирование* является воплощением принципов системного подхода при проектировании, разработке и эксплуатации программного обеспечения ЭВМ. Суть системного подхода состоит в разбиении общей сложной задачи на ряд подзадач, решении подзадач и принятии на основе этого решения.

Сложную проблему, подлежащую решению на ЭВМ, следует при программировании расчленить на ряд более простых подпроблем, каждую из которых в свою очередь можно представить еще более простыми и т. д., до тех пор, пока решение конкретной подпроблемы не станет простой задачей. Такая возможность обосновывается теоремой о структурировании, являющейся фундаментом структурного программирования. Она может быть сформулирована следующим образом: сколь бы ни сложна была задача, схема алгоритма соответствующей программы всегда может быть представлена с использованием весьма ограниченного количества ($\leqslant 3$) элементарных управляющих структур (например, «следование», «если-то-иначе», «цикл-пока») и операций присвоения и проверки над дополнительным программным счетчиком.

Комбинации трех управляющих базовых структур могут обеспечивать множество простых программ.

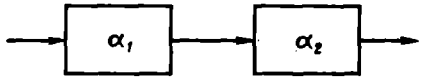


Рис. 4.1. Структура «следование»

К *основным управляющим базовым структурам* можно отнести следующие: «следование» (рис. 4.1), «если-то-иначе» (рис. 4.2), «цикл-пока» (рис. 4.3).

Структура «следование» отражает линейный процесс. Эту структуру следует понимать широко. Функциональный узел обработки (α_1 или α_2) может содержать какое-либо одно действие, один оператор, совокупность действий (операторов) или целую логически завершенную программу (или подпрограмму). Более того, из функционального узла может быть осуществлен выход в другую программу (или подпрограмму), но с непременным условием возврата в этот же узел обработки.

Примером такой структуры является программа расчета эквивалентного сопротивления контактной сети однопутного участка переменного тока $z_{1кс}$, в которой организованы выходы из основной программы в подпрограммы арифметики комплексных чисел.

Расчет может быть произведен по формуле [24]

$$z_{1кс} = \frac{z_{кр1}\, z_{тк} + j z_{кт}\, x^{т}_{кр1}}{z_{тк} + z_{кт}},$$

где $z_{кр1}$ — комплексное сопротивление контактного провода в контуре контактный провод—рельс, Ом/км; $z_{тк}$ — комплексное сопротивление несущего троса в контуре несущий трос—контактный провод, Ом/км; $z_{кт}$ — комплексное сопротивление контактного провода в контуре несущий трос—контактный провод, Ом/км; $x^{т}_{кр1}$ — сопротивление взаимоиндукции между контуром контактный провод-рельс и несущим тросом, Ом/км.

Перечисленные величины могут быть определены из выражений:

$$z_{кр1} = r_к + j m l_л (1{,}28 d_{кр}/R_к);$$
$$z_{тк} = r_т + j m l_л (1{,}28 d_{кт}/R_т);$$
$$z_{кт} = r_к + j m l_л (1{,}28 d_{кт}/R_к);$$
$$x^{т}_{кр1} = m l_л (d_{тр}/d_{кт}),$$

где $r_к$, $r_т$ — активное сопротивление контактного провода и несущего троса, Ом/км; $d_{кр}$ — расстояние между контактным проводом и рельсом, м; $R_к$,

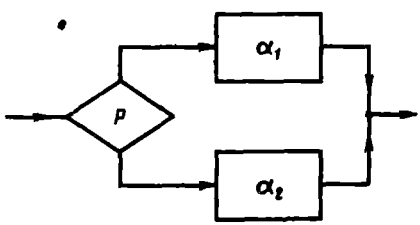


Рис. 4.2. Структура «если-то-иначе»

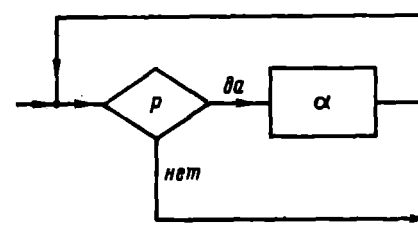


Рис. 4.3. Структура «цикл-пока»

$R_т$ — радиус контактного провода и несущего троса, м; $m=2\omega\times$ $\times 10^{-4}=0{,}0628$ Ом/км.

В этих выражениях встречаются три одинаковые конструкции: $ml_п(1{,}28d_{кр}/R_к)$, $ml_п(1{,}28d_{кт}/R_т)$ и $ml_п(1{,}28d_{кт}/R_к)$, отличающиеся переменными, входящими в аргумент функции.

Для расчета такой конструкции может быть запрограммирована специальная функция (функция пользователя), к которой можно обращаться многократно при различных значениях аргумента. Такой прием позволяет сокращать текст программы. При этом используется оператор Бейсика DEFFN (от слов definition — определение и function — функция).

Конструкция этого оператора имеет вид не DEFFN$a(x)=b$. Здесь FNa — имя функции, причем a задается пользователем и может быть буквой латинского алфавита либо цифрой; x — формальный аргумент, в качестве которого допускается только простая цифровая переменная; b — арифметическое выражение.

Обращение к функции пользователя можно совершать из любой точки программы при помощи функции FN$a(y)$, где (y) — фактический аргумент, который при обращении к функции FNa замещает формальный. Результат вычислений возвращается функцией. В качестве фактического аргумента в общем случае может быть арифметическое выражение.

В нашем случае функция пользователя может быть записана в следующем виде:

1∅ DEF FNL(x) = ∅.∅628∗LOG (1.28∗X).

Обращение к ней выполняется из разных точек программы при помощи функций:

FNL(D1/R1); FNL(D2/R2); FNL(D2/R1)

В приведенном примере D_1 и D_2 — идентификаторы переменных $d_{кр}$ и $d_{кт}$, а R1 и R2 — идентификаторы соответственно $R_к$ и $R_т$.

При вычислении $z_{1кс}$ необходимо производить действия над комплексными числами (умножение, сложение и деление). В Бейсике отсутствует арифметика комплексных чисел.

Чтобы не повторять программирование операций с комплексными числами при расчете по формуле, следует сделать это однажды и оформить в виде подпрограмм. В дальнейшем можно обращаться к ним по мере необходимости многократно из разных точек программы, как это имело место в случае с функцией пользователя (оператором-функцией). В отличие от оператора-функции подпрограмма может быть представлена совокупностью строк.

В тексте программы приняты следующие обозначения: сечение контактного провода — S1; сечение несущего троса — S2; $d_{кр}$ — D1; $d_{кт}$ — D2; $d_{тр}$ — D3; активное сопротивление контактного провода — K; активное сопротивление несущего троса — T.

Подпрограмму или подпрограммы размещают в памяти ЭВМ, начиная с номеров, заведомо бо́льших последнего номера строки основной (вызывающей) программы.

Подпрограмма на Бейсике в отличие от подпрограммы на Фортране не является самостоятельной программной единицей, а вместе с основной программой составляет единое целое. Следовательно, имена переменных и массивов, а также метки должны быть уникальны по отношению ко всей программе.

Логически завершенная часть программы может быть оформлена в виде подпрограммы при помощи операторов GOSUB и RETURN. Оператор GOSUB (GO — идти к, SUBroutine — подпрограмма) предназначен для входа в подпрограмму из основной программы. Он имеет конструкцию вс GOSUB $нс_1$, где $нс_1$ — номер начальной строки подпрограммы.

Последним выполняемым оператором подпрограммы обязательно должен быть оператор RETURN, который служит для возвращения к основной программе, т. е. к оператору, следующему за оператором GOSUB.

При помощи оператора GOSUB можно войти в подпрограмму в любом месте. Разрешены вложенные (одна в другую) подпрограммы.

Составим подпрограммы арифметики комплексных чисел, используя формулы:

$$z_1+z_2=(a_1+a_2)+j(b_1+b_2);$$
$$z_1-z_2=(a_1-a_2)+j(b_1-b_2);$$
$$z_1z_2=(a_1a_2-b_1b_2)+j(a_1b_1+a_1b_1);$$
$$\frac{z_1}{z_2}=\left(\frac{a_1a_2+b_1b_2}{a_2^2+b_2^2}\right)+j\left(\frac{a_2b_1-a_1b_2}{a_2^2+b_2^2}\right)$$

Подпрограмма сложения:

```
5Ø Ø A3=A1+A2:B3=B1+B2:RETURN
```

Подпрограмма вычитания:

```
52Ø A3=A1-A2: B3=B1-B2: RETURN
```

Подпрограмма умножения:

```
54Ø A3=A1*A2-B1*B2: B3=A1*B2+A2*B1: RETURN
```

Подпрограмма деления:

```
56Ø C=A2↑2+B2↑2
57Ø A3=(A1*A2+B1*B2)/C
58Ø B3=(A2*B1-A1*B2)/C: RETURN
```

После ознакомления с оператором-функцией и подпрограммой решим полностью задачу расчета $z_{1кс}$.

Текст программы:

```
1ØREM ПРОГРАММА РАСЧЕТА Z1КС
2ØDEF FNL(X)=Ø.Ø628*LOG(1,28*X)
3ØDATA1ØØ, 12Ø, 6, 1.3, 7.2, Ø.177, Ø.158
4ØREAD S1, S2, D1, D2, D3, K, T
5ØREM РАСЧЕТ ИНДУКТИВНЫХ СОПРОТИВЛЕНИЙ
6ØR1=SQR (S1/#PI)*1E-3:R2=SQR(S2/#PI)*1E-3
7ØX1=FNL(D1/R1):X2=FNL(D2/R2):X3=FNL(D2/R1)
8ØX4=Ø.Ø628*LOG(D3/D2)
```

```
9ØREM РАСЧЕТ ПО ФОРМУЛЕ
1ØØ A1=K: B1=X1: A2=T: B2=X2: GOSUB53Ø
11ØS1=A3: S2=B3
12ØA1=K: B1=X3: A2=Ø: B2=X4: GOSUB 53Ø
13ØA1=S1: B1=S2: A2=A3: B2=B3: GOSUB 49Ø
14ØS1=A3: S2=B3
15ØA1=T: B1=X2: A2=K: B2=X1: GOSUB 49Ø
16ØA1=S1: B1=S2: A2=A3: B2=B3: GOSUB 55Ø
17ØPRINT «АКТИВНОЕ СОПРОТИВЛЕНИЕ КОНТАКТНОЙ СЕТИ=»;
    A3; «ОМ/КМ»
18ØPRINT «ИНДУКТИВНОЕ СОПРОТИВЛЕНИЕ КОНТАКТНОЙ СЕ-
    ТИ=»; B3; «ОМ/КМ»
19ØEND
48ØREM АРИФМЕТИКА КОМПЛЕКСНЫХ ЧИСЕЛ
49ØREM ПОДПРОГРАММА СЛОЖЕНИЯ
5ØØA3=A1+A2: B3=B1+B2: RETURN
51ØREM ПОДПРОГРАММА ВЫЧИТАНИЯ
52ØA3=A1-A2: B3=B1-B2: RETURN
53ØREM ПОДПРОГРАММА УМНОЖЕНИЯ
54ØA3=A1*A2-B1*B2: B3=A1*B2+A2*B1: RETURN
55ØREM ПОДПРОГРАММА ДЕЛЕНИЯ
56ØC=A2↑2+B2↑2
57ØA3=(A1*A2+B1*B2)/C
58ØB3=(A2*B1-A1*B2)/C: RETURN
```

Структура «если-то-иначе» осуществляет разветвление в программе в зависимости от выполнения условия ДА или невыполнения условия НЕТ в узле проверки *Р*. Внутри структуры «если-то-иначе» можно снова употребить ту же структуру (рис. 4.4). Однако не следует увлекаться большой глубиной вложений, поскольку при этом будет теряться наглядность (читаемость) структуры.

Структура «цикл-пока» (повторение), представленная на рис. 4.3, позволяет организовать цикл. Отметим, что тело (содержание) цикла располагается за узлом проверки и, следовательно, будет выполняться, пока остается истинным условие, заданное в узле проверки. Разновидностью рассматриваемой структуры является структура «цикл-до» (рис. 4.5). Она отличается тем, что проверка на окончание цикла осуществляется после выполнения тела цикла. В этом слу-

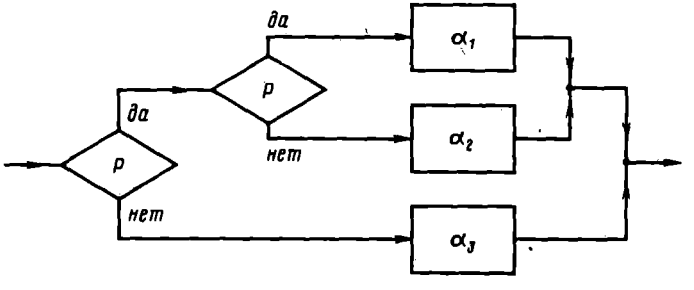


Рис. 4.4. Вложенная структура «если-то-иначе»

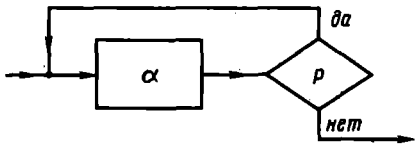


Рис. 4.5. Структура «цикл-до»

чае независимо от условия, заданного в узле проверки, тело цикла выполняется хотя бы один раз.

В ряде задач может оказаться полезной структура, носящая название «выбор». Структура «выбор» (рис. 4.6) — это обобщение рассмотренной выше структуры «если-то-иначе». В ней реализуется выбор одной из нескольких альтернатив.

Отметим одно более общее свойство всех рассмотренных выше управляющих структур. Каждая из них имеет один-единственный вход и один-единственный выход. Таким образом читатель блок-схемы или программы может быть уверенным, что управление от входа структуры обязательно попадает к ее выходу по одной из избранных альтернатив.

В теории структурного программирования показывается, что разработка структурированных программ, использование практических принципов и методов структурного программирования позволяют создавать правильные программы (правильность программы определяется как соответствие между программой и ее заданной функцией), доказывать их правильность путем логических рассуждений и получать безошибочный результат при первом же испытании.

Реализация управляющих базовых структур на языке Бейсик (в отличие от таких языков, как Паскаль, ПЛ/1 и в особенности АДА) характеризуется некоторой потерей наглядности.

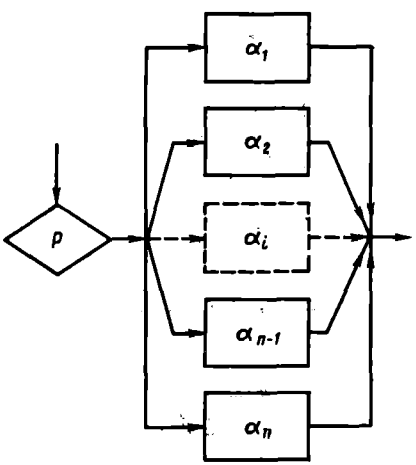


Рис. 4.6. Структура «выбор»

Структуры «если-то-иначе», «цикл-пока» и «цикл-до» реализуются при помощи операторов IF (если) и GOTO (идти к).

Конструкция оператора IF имеет вид:

$$\text{нс}\,\text{IF}\,e_1 \otimes e_2\,\text{THEN}\,\text{нс}_1$$

где e_1, e_2 — арифметические выражения; $\otimes$ — знак операции отношения; $e_1 \otimes e_2$ — условное выражение (или просто условие); THEN — служебное слово (тогда); нс_1 — номер строки, которой передается управление в случае выполнения условия $e_1 \otimes e_2$.

Если условие, записанное в операторе IF, выполняется (истинно), то происходит переход к выполнению строки нс_1, в против-

ном случае (условие не выполняется, т. е. ложно) выполняется следующий оператор.

Конструкция оператора GOTO имеет вид нс GOTO нс$_1$.

Если оператор IF называют оператором условной передачи управления, то оператор GOTO — оператором безусловной передачи. Оба оператора предназначены для изменения естественного порядка выполнения программы (по возрастанию номеров строк).

Приведем примеры реализации структур «если-то-иначе» и «цикл-до» на языке Бейсик.

Пример 1. Пусть требуется рассчитать значения коэффициента сцепления $\psi_к$ колеса с рельсом для электровоза переменного тока ВЛ80К в диапазоне скоростей от 0 до 100 км/ч с шагом ΔV, равным 5 км/ч, по формуле:

$$\psi_к = \begin{cases} 0{,}228 + \dfrac{7}{53+3v}, & \text{если } v \leqslant 40\text{км/ч}; \\ 0{,}09 + \dfrac{95}{413+3v}, & \text{если } 40 < v \leqslant 100\text{км/ч}. \end{cases}$$

Алгоритм решения задачи приведен на рис. 4.7. В схеме алгоритма использована композиция управляющих структур «если-то-иначе» и «цикл-до». Первая (блоки *3, 4* и *5*) используется для определения граничной скорости (40 км/ч) и выбора соответствующей расчетной формулы, вторая (блоки *3—8*) — для организации циклических вычислений по выбранной формуле.

При этом структура «если-то-иначе» (развилка) входит наряду с блоками *6* и *7* в тело структуры «цикл-до».

В данном и в более общем случае для организации циклических вычислений необходимо:

присвоить управляющей переменной цикла начальное значение (в данном примере управляющей переменной является скорость V, начальное значение которой равно 0);

выполнить тело цикла, т. е. реализовать блоки *3, 4* (или *5*) и *6*;

изменить значение управляющей переменной на величину приращения ΔV (блок *7*),

сравнить текущее значение управляющей переменной с ее конечным значением (блок *8*) и изменить естественный порядок выполнения задачи в случае выполнения условия.

В соответствии со схемой (см. рис. 4.7) напишем текст программы на Бейсике с использованием операторов управления IF и GOTO.

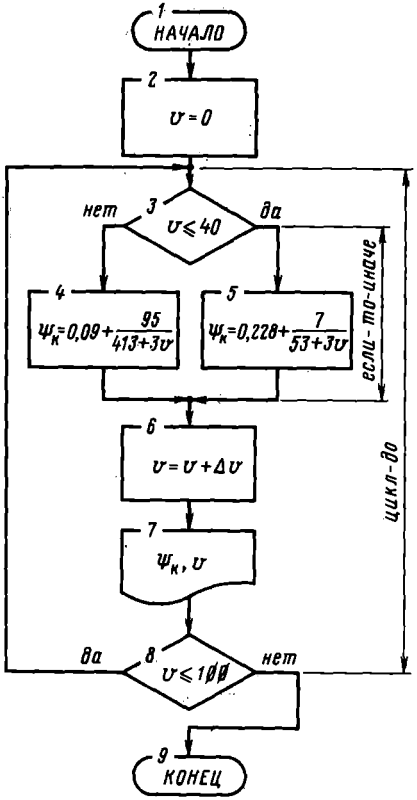


Рис. 4.7. Алгоритм расчета коэффициента сцепления колеса с рельсом

```
Блок 1   1Ø REM РАСЧЕТ КОЭФФИЦИЕНТА СЦЕПЛЕНИЯ КОЛЕСА С
         РЕЛЬСОМ ДЛЯ ЭЛЕКТРОВОЗА ПЕРЕМЕННОГО ТОКА ВЛ80К
         2ØV=Ø
         3ØIF V<=4ØTHEN6Ø
         4ØK=Ø.Ø9+95/(413+3*V)
         5ØGOTO 7Ø
         6ØK=Ø.228+7/(53+3*V)
         7ØPRINT «V=»; V, «КОЭФФИЦИЕНТ СЦЕПЛЕНИЯ=»; K
         8ØV=V+5
         9ØIF V<=1ØØTHEN 3Ø
         1ØØEND
```

Структуру «если-то-иначе» можно написать более наглядно, используя возможность записи нескольких операторов в одной строке программы:

```
3ØIF V≤4ØTHEN 4Ø: K=Ø.Ø9+95/(413+3*V): GOTO 5Ø
4ØK=Ø.228=7/(53+3V)
5ØPRINT «V=»; V, «КОЭФФИЦИЕНТ СЦЕПЛЕНИЯ-»; K
```

Если в предыдущем случае структура была реализована четырьмя строками (3Ø — 6Ø), то теперь только двумя. Альтернатива НЕТ записана следом за оператором IF в одной строке, альтернатива ДА в следующей строке. Узлом слияния обеих альтернатив служит строка с номером 5Ø.

Структура «цикл-до» может быть реализована на Бейсике и при помощи специального оператора, называемого оператором цикла FOR/NEXT.

Пример 2. Требуется рассчитать удельное сопротивление движению электровоза без тока w'_{0x} в диапазоне изменения скоростей от 0 до 100 км/ч с шагом 5 км/ч по формуле $w'_{0x}=2{,}4+0{,}011v+0{,}00035v^2$ Н/кН и построить графическую зависимость $w'_{0x}=f(v)$ при помощи функции TAB.

Для построения графика функции необходимо перед началом цикла рассчитать масштаб исходя из того, чтобы график расположился заданным образом в плоскости экрана, а аргумент функции TAB не превысил максимально допустимого значения (числа позиций в строке экрана или бумаги). Ось скорости пустим вдоль движения бумаги (одно перемещение каретки печатающего устройства будет соответствовать $\Delta v=5$ км/ч), тогда масштаб необходимо вычислить для w'_{0x}. Пусть число позиций в строке l равно 80. При построении графика отступим от левой и правой границы строки на 10 позиций. Тогда масштаб M будет:

$$M=\frac{l}{w'_{0x\,max}-w'_{0x\,min}}=\frac{l}{\Delta w'_{0x}}=\frac{60}{\Delta w'_{0x}}$$

Поскольку удельное сопротивление движению растет с увеличением скорости, $w'_{0x\min}$ и $w'_{0x\max}$ должны быть определены по следующим формулам:

$$w'_{0x\,min}=2{,}4+0{,}011\,V_{min}+0{,}00035\,V^2_{min};$$
$$w'_{0x\,max}=2{,}4+0{,}11\,v_{max}+0{,}00035\,v^2_{max}.$$

Текущие значения удельного сопротивления движению, измеренные в позициях строки s, определяются по формуле:

$$s=10{,}5+M(w'_{0x}-w'_{0x\,min})$$

Программа решения в соответствии со схемой алгоритма (рис. 4.8) будет иметь вид:

```
10 REM РАСЧЕТ УДЕЛЬНОГО СОПРОТИВЛЕНИЯ ДВИЖЕНИЮ ЭЛЕКТРОВОЗА
20 DEF FNW(X)=2.4+0.0011*0.000035*X↑2
30 V1=0: V2=100
40 W1=FNW(V1)
50 M=60(FNW(V2)-W1)
60 PRINT TAB (10) «ГРАФИК УДЕЛЬНОГО СОПРОТИВЛЕНИЯ ДВИЖЕНИЮ
                  ЭЛЕКТРОВОЗА»
70 PRINT «— — — — — — — — — — — — — — — — — — — — →Н/КН»
80 FOR V=V1 TO V2 STEP5
90 W=FNW(V)
100 PRINT«!»; «W=»; W; TAB(10.5+(W-W1)*M); «*»
110 NEXT V
120 PRINT «V»
130 PRINT «КМ/Ч»
140 END
```

Поскольку в программе требуется многократное обращение к расчетной формуле, в операторе 20 применен описанный ранее оператор определения функции DEFFN (функции пользователя). В операторах 40, 50, 90 идет обращение к функции по имени FNW с фактическим аргументом.

Цикл организован по управляющей переменной V (скорость), а в заголовке присутствует полное описание параметров, в том числе и шаг, равный 5 км/ч. В теле цикла производится вычисление текущего значения удельного сопротивления движению и на каждом этапе на экран (или на печать) с помощью оператора PRINT выводятся значение сопротивления (прижимается к вертикальной оси) и одна, соответствующая этому значению сопротивления, точка графика, изображаемая символом «*». Остальные операторы PRINT программы содержат только символьные константы и используются для оформления графика: печати заголовка, построения осей и указания размерностей величин, откладываемых по этим осям.

В заключение остановимся на реализации структуры ВЫБОР, которая может быть удобна в случае, если число разветвлений в данной точке алгоритма более двух. Пусть, например, требуется рассчитать эффективный ток одной из заданных обмоток тягового трансформатора подстанции переменного тока по известным значениям средних и эф-

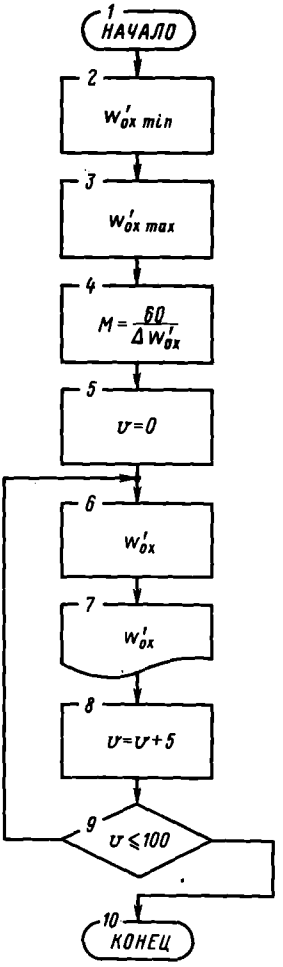


Рис. 4.8. Алгоритм расчета удельного сопротивления движению электровоза

фективных токов плеч и одинаковым коэффициентам мощности, используя одну из трех формул:

$$I_{\text{эIоб}} = \frac{1}{3}\sqrt{4I_{\text{эI}}^2 + I_{\text{эII}}^2 + 2I_{\text{I}}I_{\text{II}}};\quad I_{\text{эIIоб}} = \frac{1}{3}\sqrt{I_{\text{эI}}^2 + 4I_{\text{эII}}^2 + 2I_{\text{I}}I_{\text{II}}}$$

$$I_{\text{эIIIоб}} = \frac{1}{3}\sqrt{I_{\text{эI}}^2 + I_{\text{эII}}^2 - I_{\text{I}}I_{\text{II}}}$$

Будем использовать оператор ON, конструкция которого имеет вид:

нсONeGOTOнс$_1$, нс, нс$_к$

здесь ON — имя оператора; e — арифметическое выражение, которое определяет порядковый номер строки (1, 2, ..., к) в списке номеров строк в операторе ON; нс$_1$, нс$_2$, ..., нс$_к$ — список номеров строк.

Если значение e не целое, то оно округляется до целого. Если же оно не входит в диапазон номеров строк списка, то выполняется следующий по порядку оператор.

Составим программу расчета для нашей задачи, используя оператор ON и предварительно заполнив таблицу идентификаций:

I_{I}	$I_{\text{ии}}$	$I_{\text{эI}}$	$I_{\text{эII}}$	$I_{\text{эIоб}}$, $I_{\text{эIIоб}}$, $I_{\text{эIIIоб}}$
S1	S2	E1	E2	I

```
10 REM РАСЧЕТ ЭФФЕКТИВНОГО ТОКА ОБМОТКИ ТРАНСФОРМАТОРА
15 INPUT E1, E2, S1, S2
20 INPUT «ВВЕДИТЕ НОМЕР ОБМОТКИ ТРАНСФОРМАТОРА, ДЛЯ КОТОРОЙ БУДЕТ ВЫПОЛНЯТЬСЯ РАСЧЕТ», J
30 ON J GOTO 40, 60, 80
40 I=SQR(4*E1↑2+E2↑2+2*S1*S2)/3
50 GOTO 90
60 I=SQR(E1↑2+4*E2↑2+2*S1*S2)/3
70 GOTO 90
80 I=SQR(E1↑2+E2↑2-S1*S2)/3
90 PRINT «ЭФФЕКТИВНЫЙ ТОК ОБМОТКИ N»; J«=»; I; «A»
100 END
```

В конструкции оператора ON допускается применение оператора GOSUB на месте оператора GOTO.

4.4. Операции с массивами

Если в программе используются числовые или символьные массивы, а также символьные переменные нестандартной длины, они должны быть обязательно описаны при помощи оператора DIM (сокращение от DIMENSION — размер). Оператор DIM пред-

назначен для резервирования места в памяти ЭВМ для массивов и должен предшествовать использованию массивов в операциях.

В качестве размеров массивов могут использоваться только цифровые константы. Не следует смешивать указание размера с возможностью использования арифметических выражений в качестве индексов элементов массивов, участвующих в операциях.

Аналогичные оператору DIM функции выполняет и оператор COM (сокращение от COMMON — общий). Кроме резервирования места в памяти ЭВМ, он используется для объявления переменных и массивов общими для нескольких программ. Переменные и массивы, определенные оператором COM, являются общими для нескольких программ и их значения не стираются при вводе этих программ в машину. Оператор COM должен предшествовать использованию любых переменных и массивов и располагаться перед первым оператором DIM.

Пример программы с использованием одномерных массивов. Требуется рассчитать скоростную и тяговую характеристики тягового двигателя ТЛ-2К электровоза ВЛ10 при изменении диаметра колеса и передаточного числа редуктора. Исходные тяговая и скоростная характеристики тягового двигателя заданы для диаметра колеса $D=1250$ мм и передаточного отношения зубчатой передачи $\mu_1=3{,}826$:

v, км/ч	41,8	43,6	45,4	46,7	48,7	52,2	58,0	63,9	70,6	87,5
F, кН	93,6	80,0	66,5	59,5	49,7	39,5	26,1	20,4	14,0	8,1
I, А	800	700	600	500	480	400	300	250	200	150

Пересчет силы тяги и скорости при неизменном значении тока для значений диаметра колеса D_2 и передаточного отношения зубчатой передачи μ_2 производится по формулам:

$$F_2 = F_1 \frac{D_1}{D_2} \frac{\mu_2}{\mu_1}; \quad v_2 = v_1 \frac{D_2}{D_1} \frac{\mu_1}{\mu_2},$$

где F_1 и v_1 — тяговое усилие, развиваемое двигателем, и скорость электровоза при диаметре ведущего колеса D_1 и передаточном отношении μ_1.

Текст программы для расчета в соответствии с алгоритмом (рис. 4.9):

```
10 REM   РАСЧЕТ ТЯГОВОЙ И СКОРОСТНОЙ ХАРАКТЕРИСТИК ТЯГОВОГО
         ДВИГАТЕЛЯ ТЛ-2К ЭЛЕКТРОВОЗА ВЛ10
20 DATA  41.8, 93.6, 80.0, 43.6, 80.0, 70.0, 45.4, 66.5, 60.0, 46.7, 59.5, 55.0,
         48.7, 49.7, 48.0, 52.2, 39.5, 40.0, 58.0, 26.1, 30.0, 63.9, 20.4, 25.0, 70.6,
         14.0, 20.0
30 DATA  87.5, 8.1, 15.0
40 DIM F1 (10), V1(10),I(10), F2(10), V2(10)
50 INPUT «ВВЕДИТЕ ЗНАЧЕНИЯ ДИАМЕТРОВ КОЛЕС, ПЕРЕДАТОЧНЫХ
         ОТНОШЕНИЙ В ПОСЛЕДОВАТЕЛЬНОСТИ D1, D2, M1, M2», D1,
         D2, M1, M2
60 INPUT «ВВЕДИТЕ ЧИСЛО ЗНАЧЕНИЙ В ТАБЛИЦЕ», N
70 FOR J=1 TO N
80 READ V1 (J), F1 (J), I (J)
90 NEXT J
100 A=D1*M2/(D2*M1)
110 FOR J=1 TO N
120 F2 (J) =F1 (J) *A
```

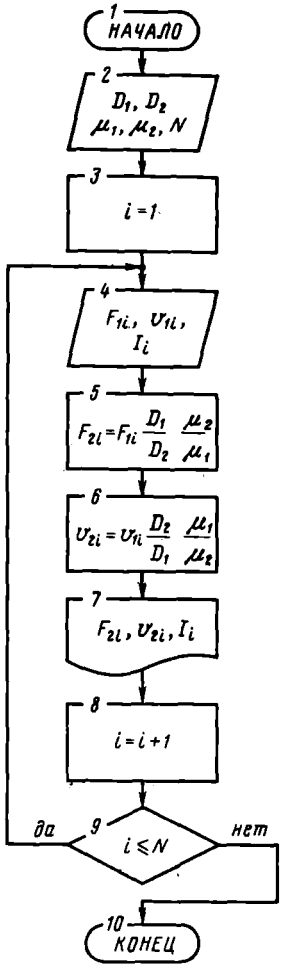
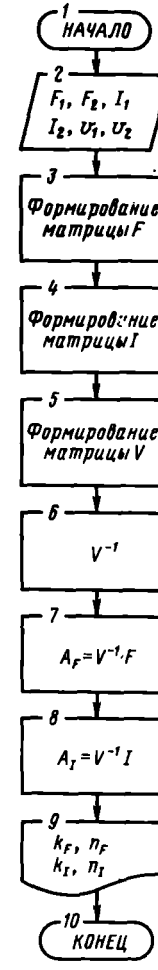


Рис. 4.9. Алгоритм расчета тяговой и скоростной характеристик

◁



▷

Рис. 4.10. Схема алгоритма аппроксимации тяговой и скоростной характеристик двигателя ТЛ-2К электровоза ВЛ10

```
13 V2 (J) = V1 (J) / A
14 PRINT «F2 (»;J;«) = »;F2(J),«V2(»;J;«) = »;V2(J),«I(»;J;«) = »; I(J)
15 NEXT J
16 END
```

Пример программы с использованием двухмерных массивов. Поставим задачу аппроксимации тяговой и скоростной характеристик (см. предыдущий пример) тягового двигателя ТЛ-2К электровоза ВЛ10 аналитическими выражениями вида:

$$F = \kappa_F v^{n_F};$$
$$I = \kappa_I v^{n_I}$$

Задача аппроксимации сводится к поиску коэффициентов κ_F, κ_I, n_F и n_I. Выберем по две точки на каждой характеристике F_1, F_2 и I_1, I_2, соответствующей скоростям v_1, v_2 с таким расчетом, чтобы погрешность аппроксимации была наименьшей в рабочей части характеристик.

Прологарифмируем аппроксимирующие выражения и составим систему уравнений:

$$\begin{cases} \ln F_1 = n_F \ln v_1 + \ln \kappa_F; \\ \ln F_2 = n_F \ln v_2 + \ln \kappa_F; \end{cases}$$

$$\begin{cases} \ln I_1 = n_I \ln v_2 + \ln \kappa_I; \\ \ln I_2 = n_I \ln v_2 + \ln n_I. \end{cases}$$

Метод решения приведенных систем уравнений рассмотрим на примере одной из них, относящейся к тяговой характеристике. Запишем систему уравнений в матричном виде:

$$\left\| \begin{matrix} \ln F_1 \\ \ln F_2 \end{matrix} \right\| = \left\| \begin{matrix} \ln v_1 & 1 \\ \ln v_2 & 1 \end{matrix} \right\| \times \left\| \begin{matrix} n_F \\ \ln k_F \end{matrix} \right\|$$

Обозначим столбцевую матрицу свободных членов именем F, квадратную матрицу коэффициентов при неизвестных — именем V и столбцовую матрицу неизвестных — A_F. Тогда система уравнений может быть записана в упрощенном виде $F = VA_F$.

Матрица V является квадратной неособенной, следовательно, имеет обратную матрицу. Тогда решение системы можно записать в виде $A_F = V^{-1}F$

Аналогично для скоростной характеристики $A_I = V^{-1}I$.

Искомые коэффициенты находятся из выражений:

$$n_F = A_F(1);\ n_I = A_I(1);$$

$$\kappa_F = e^{A_F(2)};\ \kappa_I = e^{A_I(2)}$$

В схеме алгоритма (рис. 4.10) встречаются операции с массивами (блоки *6*, *7*, *8*,), в частности, обращение матрицы и умножение обращенной матрицы на столбцовые матрицы (одномерные массивы). Работа с матрицами в языке Бейсик (в отличие от языка Фортран) решается на операторном уровне.

Прежде чем перейти к составлению программы по приведенной схеме, рассмотрим операторы языка Бейсик, представляющие возможности проведения различных операций над массивами и называемые матричными операторами (табл. 4.3).

Операторы MAT INPUT, MAT READ, MAT PRINT И MAT REDIM могут быть использованы не только для цифровых массивов, но и для символьных.

Остальные операторы, приведенные в табл. 4.3, используются только для цифровых операторов.

При программировании нашей задачи можно воспользоваться операторами MAT INV и MAT$*$. Конструкция оператора MAT INV имеет вид нс MATa = INV(b) [,c]

Таблица 4.3

Наименование оператора	Имя оператора	Конструкция оператора
Сложение матриц	MAT+	MAT A=B+C
Вычитание матриц	MAT−	MAT A=B−C
Умножение матриц	MAT*	MAT A=B*C
Присвоение элементам одной матрицы значений другой	MAT=	MAT A=B
Вычисление матрицы, обратной данной, и определителя	MAT INV	MAT A=INV(B), C
Умножение матрицы на число	MAT ( )*	MAT A=(K)*B
Транспонирование матрицы	MAT TRN	MAT A=TRN(B)
Обнуление матрицы	MAT ZER	MAT A=ZER
Приведение матрицы к единичной	MAT IND	MAT A=IND
Присвоение каждому элементу матрицы значения 1	MAT CON	MAT A=CON
Ввод элементов матрицы	MAT INPUT	MAT INPUT A, B
Присвоение элементам матрицы значений из оператора DATA	MAT READ	MAT READ A, B
Печать матриц	MAT PRINT	MAT PRINT A, B
Переопределение размерности матриц	MAT REDIM	MAT REDIM A(X,Y)

Здесь a — имя результирующей (обращенной) цифровой матрицы; b — имя исходной цифровой матрицы; c — необязательный параметр — имя цифровой переменной, в которой хранится определитель исходной матрицы; INV — служебное слово (от INVERSE — обратный).

Оператор выполняет вычисление матрицы, обратной квадратной матрице b, стоящей в правой части, и присваивает ее значение матрице a, стоящей в левой части. При этом размерность матрицы в левой части становится равной размерности матрицы, стоящей в правой части.

При желании можно получить значение определителя исходной матрицы b. Для этого необходимо в операторе указать имя цифровой переменной.

Конструкция оператора MAT* имеет вид не MAT$a=b*c$. Здесь a, b и c — имена цифровых массивов. Не допускается ис-

пользование одного и того же имени в правой и левой частях оператора.

Оператор выполняет умножение массивов b и c, указанных в правой части оператора, и присваивает значения произведения массиву a, указанному в левой части оператора. Недопустимо умножение двух одномерных массивов.

После знакомства с матричными операторами MAT INV и MAT $*$ составим программу аппроксимации тяговой и скоростной характеристик:

```
10 REM    АППРОКСИМАЦИЯ ТЯГОВОЙ И СКОРОСТНОЙ ХАРАК-
          ТЕРИСТИК ТЯГОВОГО ДВИГАТЕЛЯ ТЛ-2К
20 INPUT  «ВВЕДИТЕ ЗНАЧЕНИЯ F1, F2, I1, I1, V1, и V2», F1, F2,
          I1, I2, V1, V2
30 REM    ФОРМИРОВАНИЕ МАТРИЦ F, I и V
40 DIM    F(2), I(2), V(2,2), V1(2,2), A1(2), A2(2)
50 F(1)=LOG(F1):F(2)=LOG(F2)
60 I(1)=LOG(I1):I(2)=LOG(I2)
70 V(1,1)=LOG(V1):V(1,2)=1:V(2,1)=LOG(V2):V(2,2)=1
80 REM    РЕШЕНИЕ МАТРИЧНОГО УРАВНЕНИЯ
90 MATV1=INV(V)
100 MATA1=V1*F
110 MATA2=V1*I
120 REM   ОФОРМЛЕНИЕ РЕЗУЛЬТАТОВ НА ПЕЧАТЬ
130 PRINT TAB (10); «КОЭФФИЦИЕНТЫ ТЯГОВОЙ ХАРАКТЕРИС-
          ТИКИ»
140 PRINT TAB(10); «N=»;A1(1), «K=»;EXP(A1(2))
150 PRINT TAB(10), «КОЭФФИЦИЕНТЫ СКОРОСТНОЙ ХАРАКТЕ-
          РИСТИКИ»
160 PRINT TAB(10); «N=»;A2(1),«K=»;EXP(A2(2))
170 END
```

Использование матричных операторов значительно сокращает время выполнения действий над массивами по сравнению с программой, написанной на языке Бейсик без использования матричных операторов.

Для ввода в ЭВМ и вывода на печать тяговой и скоростной характеристик, заданных в табличном виде, можно использовать матричные операторы MAT READ и MAT PRINT.

Конструкция оператора MAT READ имеет вид нс MAT READ $a_1[x_1]\times$ $\times[,y_1]],a_2[x_2[,y_2]],\ldots\ldots$, Здесь a_1, a_2, a_i, a_n — список ввода $(a_i=a_i[x_i[,y_i]])$; x_i, y_i — размеры массива (необязательные параметры); a_i — имя массива.

Оператор предназначен для присвоения значений, содержащихся в операторах DATA, массивам a_1, a_2,..., a_n. Заполнение двухмерных массивов происходит построчно. В отличие от Бейсика эквивалентные по своим действиям операторы Фортрана производят заполнение по столбцам.

Если в операторах DATA недостаточно данных для оператора MAT READ, возникает ошибочная ситуация.

Конструкция оператора MAT PRINT имеет вид нс MAT PRINT $[a,]b_1[;]b_2[;]\ldots\ldots b_n[;]$ Здесь a — код устройства (0 5 — экран, ОС —

бумажная печать); b_1, b_2,..., b_i, b_n — список вывода (b_i = $=b_i[x_i[,y_i]]$); x_i, y_i — размеры массива (необязательные параметры); b_i — имя массива.

Если элементы списка вывода разделяются запятыми, печать будет производиться в зонном формате, если же точкой с запятой,— в уплотненном формате. Допустимо использование в списке вывода разделителей обоих типов. Каждая строка массива выводится с новой строки экрана (бумаги).

4.5. Операции над битами

При решении задач управления устройствами электрифицированной железной дороги возникает необходимость программирования алгоритмов, использующих алфавит, состоящий всего из двух символов. На практике чаще всего используют символы 0 и 1. Физической величине или состоянию какой-либо системы ставится в соответствие определенное сочетание символов 0 и 1 (двоичное число). Единицы в этом сочетании могут соответствовать, например, включенному состоянию элементов системы, а нули — отключенному. Перевод системы в новое состояние (управляющее воздействие) требует производства операций над отдельными символами (битами) двоичного числа.

Алгоритмический язык Бейсик располагает средствами, позволяющими производить различные операции как над отдельными битами двоичного числа, так и над их сочетаниями.

Для выполнения различных математических и логических операций над битовой структурой одного или нескольких символов символьной переменной в языке Бейсик имеется несколько операторов. Названия этих операторов и выполняемые ими функции представлены в табл. 4.4.

Рассмотрим группу операторов, выполняющих логические операции.

Если в качестве параметра *b* в операторах ADD, AND, OR, XOR и BOOL используется двузначное шестнадцатеричное число, то оно добавляется к каждому байту символьной переменной для выполнения указанных логических операций. Если же в качестве параметра *b* используется символьная переменная, то выполняются логические операции над двумя символьными переменными. В тех случаях, когда символьные переменные имеют разную длину, их выравнивают по правому краю (с младшего байта), а недостающие позиции более короткой переменной заполняют нулями.

Результат всегда (за исключением операции 5 в операторе BOOL) присваивается первой символьной переменной. Если результат длинее этой переменной, то запоминается допустимая длина младших байтов результата.

Необязательный параметр *C* в операторе ADD определяет наличие или отсутствие единицы переноса между складываемыми бай-

Таблица 4.4

Имя оператора	Выполняемая функция	Конструкция оператора
ADD	Сложение логических аргументов	нс ADD [C] (a, b), где c — параметр, обусловливающий сложение двух операндов с учетом переноса между байтами; a — символьная переменная; b — то же или двузначное шестнадцатеричное число
AND	Логическое умножение И	нс AND (a, b)
OR	Логическое сложение ИЛИ	нсOR (a, b)
XOR	Исключающее ИЛИ	нсXOR(a, b)
BOOL	16 возможных логических операций	нс BOOL h(a, b), где h — шестнадцатеричная цифра, задающая логическую операцию
ROTATE	Циклический сдвиг влево каждого байта символьной переменной	нс ROTATE(a, k), где k — число сдвигов ($1 \leqslant k \leqslant 7$)
INIT	Присвоение символьным переменным и массивам параметра, записанного в операторе в круглых скобках	нсINIT (d)a_1, a_2, ..., a_n, где d — символьная константа, символьная переменная, двузначное шестнадцатеричное число; a_1, a_2, ..., a_n — символьные переменные или массивы
BIN	Преобразование целой части значения арифметического выражения в двоичное число	нсBIN(a)=e, где a — символьная переменная; e — арифметическое выражение ($e \leqslant 256$)
CONVERT	Преобразование символьной переменной в цифровую и наоборот	1) нсCONVERTaTOx 2) нсCONVERTeTOa, (f), где x — цифровая переменная; f — формат
PACK	Упаковка значений цифровых переменных и массивов в символьные по заданному формату	нсPACK(f)aFROM e_1, e_2, где a — символьная переменная или массив; FROM — служебное слово (из; от); e_1, e_2, ..., e_n — арифметические выражения или цифровые массивы
UNPACK	Распаковка данных из символьной переменной или массива в цифровые	нсUNPACK (f)aTO (x_1, x_2, ..., x_n), где x_1, x_2, ..., x_n — цифровые переменные или массивы

тами двоичных чисел. Если параметр C включен в оператор, то перенос между складываемыми байтами учитывается.

Оператор BOOL является обобщенным логическим оператором, реализующим функции алгебры логики, образуемые на наборе из двух переменных. Результаты логических операций приведены в табл. 4.5.

Таблица 4.5

Коды	Операнды		Результат															
Двоичные	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Шестнадцатеричные			0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Как видно из табл. 4.5, операции 6, 8 и E реализуют соответственно «исключающее ИЛИ», логическое умножение и логическое сложение, т. е. те, которые могут быть реализованы и операторами XOR, AND и OR. Ряд других операций, приведенных в таблице, широко применяется и имеет специальные названия:

Шестнадцатеричный код	Название	Условное обозначение
0	Нулевая функция	0
1	Функция Вебба	.
7	Функция Штрихшефера	/
9	Равнозначность	~
B	Импликация	→
F	Единичная функция	1

Если в операторе INIT (оператор присвоения начального значения символьным переменным и массивам) в качестве параметра, записанного в круглых скобках, используется символьная переменная, то при определении значений символьных переменных и массивов в списке оператора используется ее первый символ.

Группа операторов BIN, CONVERT, PACK и UNPACK предназначена для преобразования цифровой информации в символьную и наоборот.

Таблица 4.6

Имя оператора языка Бейсик	Мнемоническое изображение команды микропроцессора КР580ИК80А	Условное обозначение операции	Примечание
ADD	ADD	+	Сложение логических аргументов
ADDC	ADC	+	Сложение с учетом переноса 1 между байтами
AND	ANA	∧	Конъюнкция
OR	ORA	∨	Дизъюнкция
XOR	XRA	⊕	Исключающее ИЛИ
ROTATE	RLC	Обозначение не предусмотрено	Циклический сдвиг влево

В результате преобразования целой части значения арифметического выражения при помощи оператора BIN получается двузначное шестнадцатеричное число, которое присваивается первому символу символьной переменной.

Оператор CONVERT имеет две формы. Первая предназначена для преобразования значения символьной информации в цифровую, вторая — для преобразования значения цифровой переменной в символьную по заданному формату.

Различают два формата:

формат для записи чисел в естественной форме (с фиксированной точкой), например, ±##·### (# — десятичная цифра);

формат для записи чисел в форме с плавающей точкой (экспоненциальная форма), например ±#·##E±XX (XX — показатель степени при основании 10).

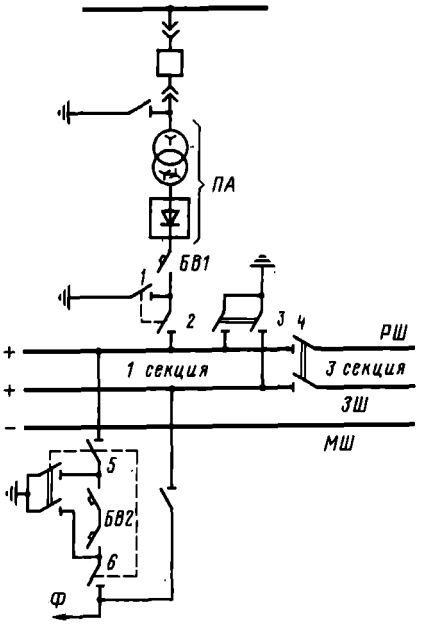


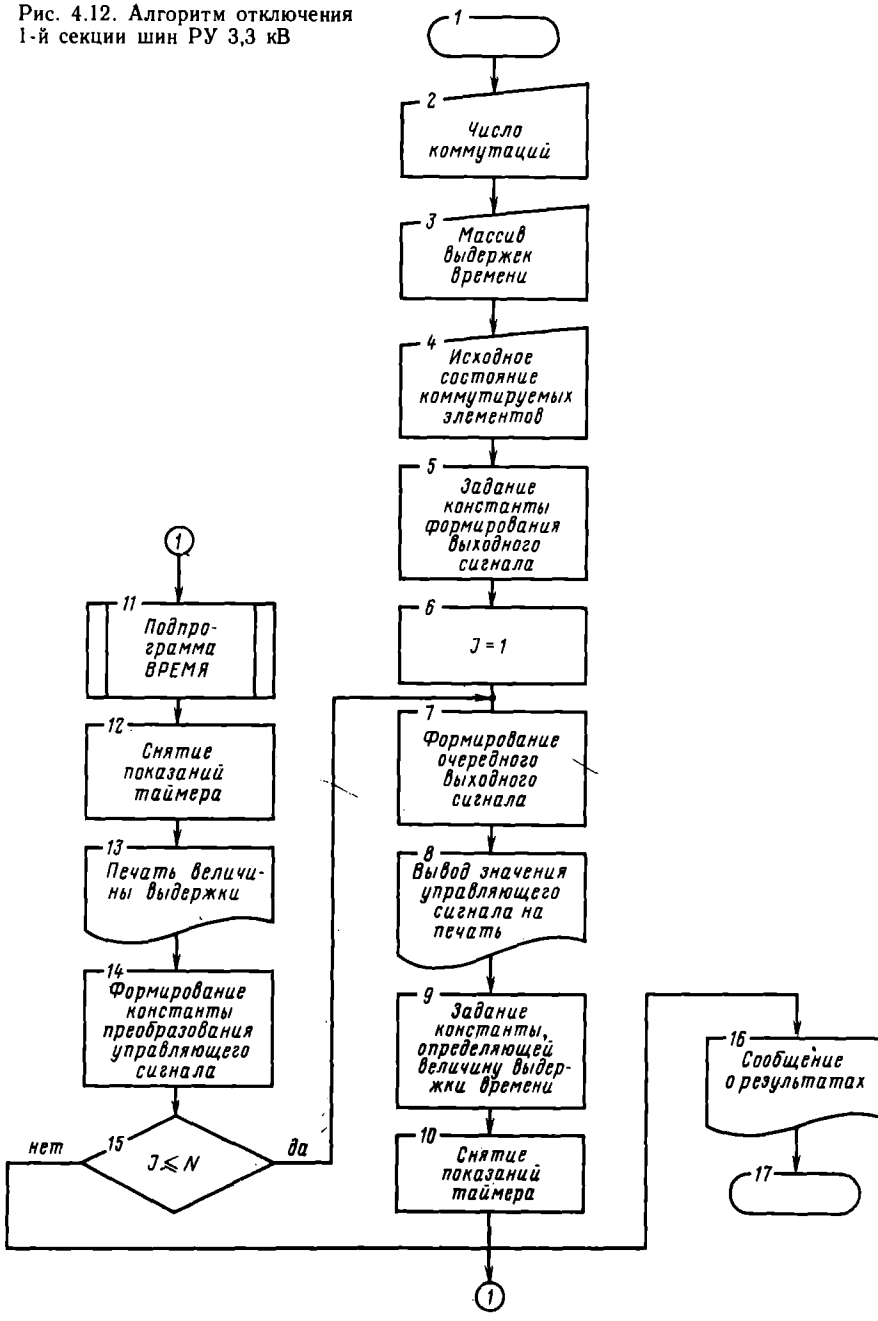
Рис. 4.11. Фрагмент принципиальной электрической схемы РУ 3,3 кВ

В отличие от оператора CONVERT операторы PACK и UNPACK осуществляют операции преобразования символьной информации в цифровую и наоборот не только со значениями переменных, но и массивов.

Включение в состав операторов языка Бейсик операторов типов ADD, AND, OR, XOR, ROTATE дает доступ к некоторым средствам уровня машинного языка микропроцессоров. Сравнение этих операторов с аналогичными по своим функциям командами микропроцессора КР580ИК80А приведено в табл. 4.6.

Сформулируем небольшую задачу, связанную с управлением работой одного из элементов тяговой подстанции постоянного тока. Пусть требуется разработать алгоритм и программу на Бейсике, осуществляющую выводы в ремонт 1-й секции РУ 3,3 кВ (рис. 4.11). РУ 3,3 кВ состоит из трех секций. Каждая секция имеет три шины — рабочую шину *РШ*, минусовую *МШ* и запасную *ЗШ*. К 1- и 2-й секциям подключают преобразовательные агрегаты *ПА* и питающие линии (фидера) контактной сети. Преобразовательный агрегат состоит из тягового трансформатора и полупроводникового преобразователя. Он присоединяется к шинам 3,3 кВ посредством быстродействующего выключателя *БВ1* и разъединителя *2* с заземляющим ножом *1*.

Рис. 4.12. Алгоритм отключения 1-й секции шин РУ 3,3 кВ



Коммутация 1- и 3-й секций осуществляется при помощи секционного разъединителя *4*. Секционирование рабочей и запасной шин 3,3 кВ позволяет поочередно выводить в ремонт 1- и 2-ю секции без полного отключения РУ 3,3 кВ.

При ремонте 1-й секции шин сначала отключают быстродействующие выключатели фидера *БВ2* и его разъединители *5* и *6*, подключенные к 1-й секции. Затем отключают *БВ1* и разъединитель *2* и включают заземляющий нож *1*. После этих операций отключают секционный разъединитель *4* и включают заземляющие ножи разъединителя *3*.

Описанная последовательность операций является словесным алгоритмом решения поставленной выше задачи, который представлен на рис. 4.12. В блоке *2* задается число коммутаций, в нашем случае оно равно 7. В блоке *3* задается шесть значений выдержек времени между коммутациями. В блоке *4* задается константа *K1*, соответствующая исходному состоянию коммутируемых элементов до отключения 1-й секции (0 — отключенное состояние, 1 — включенное): *БВ2* включен (1); разъединители *5* и *6* (см. рис. 4.11) включены (1); *БВ1* включен (1); разъединитель *1* включен (1); короткозамыкатель *2* отключен (0); секционный разъединитель *4* включен (1); заземляющие ножи *2* отключены (0)

Константа *K1*, представленная восьмиразрядным двоичным числом, имеет вид:

№ 7	№ 6	№ 5	№ 4	№ 3	№ 2	№ 1	№ 0
0	0	1	0	1	1	1	1

В блоке *5* задается константа *K2* для последовательного преобразования константы *K1*, соответствующей исходному состоянию элементов схемы. Она имеет в данном случае вид:

№ 7	№ 6	№ 5	№ 4	№ 3	№ 2	№ 1	№ 0
0	0	0	0	0	0	0	1

В блоке *6* открывается цикл по числу коммутаций. В блоке *7* реализуется логическая операция «исключающее ИЛИ» над константами *K1* и *K2*. В первый раз будет получен следующий результат:

```
0 0 1 0 1 1 1 1
0 0 0 0 0 0 0 1
---------------
0 0 1 0 1 1 1 0
```

т. е. новая константа *K1* отличается от своего предыдущего (исходного) состояния нулем в разряде № 0, что соответствует отключению *БВ2* и прежнему состоянию остальных элементов.

В блоке *8* значение полученного управляющего сигнала (константы *K1*) выводится на печать (при работе ЭВМ в режиме управления — на интерфейс или на исполнительный механизм). В

блоке *9* задается константа, определяющая выдержку времени между предыдущей и последующей коммутациями. В блоке *11* реализуется программа ВРЕМЯ, обеспечивающая временну́ю задержку.

В блоках *10, 12* и *13* производится снятие и распечатка показаний таймера с целью контроля работы программы ВРЕМЯ. В блоке *14* с помощью логической операции «циклический сдвиг влево» изменяется константа *K2* с целью формирования на следующем этапе очередного управляющего сигнала. В блоке *15* закрывается цикл по числу коммутаций. В блоках *16* и *17* завершается работа алгоритма.

Составим программу на языке Бейсик в соответствии со схемой алгоритма, приведенной на рис. 4.12.

```
10 REM    ВЫВЕДЕНИЕ В РЕМОНТ 1-Й СЕКЦИИ РУ 3,3 КВ БЕЗ
          ПОЛНОГО ЕГО ОТКЛЮЧЕНИЯ
20 DIM    A⊙I, B⊙I, C(8)
30 INPUT  «ВВЕДИТЕ ЧИСЛО КОММУТИРУЕМЫХ ЭЛЕМЕНТОВ»,N
40 PRINT  «ВВЕДИТЕ МАССИВ ВЫДЕРЖЕК ВРЕМЕНИ МЕЖДУ
          КОММУТАЦИЯМИ»
50 MAT INPUT C
60 REM    ЗАДАНИЕ ИСХОДНОГО СОСТОЯНИЯ КОММУТИРУЕ-
          МЫХ ЭЛЕМЕНТОВ И КОНСТАНТЫ ПРЕОБРАЗОВАНИЯ
          УПРАВЛЯЮЩЕГО СИГНАЛА
70 W⊙=HEX(2F):A⊙=HEX(01)
80 REM    ОРГАНИЗАЦИЯ ЦИКЛА КОММУТАЦИИ
90 FOR J=1TO N
100 XOR(B⊙,A⊙)
110 PRINT «СОСТОЯНИЕ КОММУТИРУЕМЫХ ЭЛЕМЕНТОВ НА»;
J;«—ОМ ШАГЕ»;
120 NEXPRINT TAB(20);B⊙
130 C=C(J)
140 INPUT0S1
150 GOSUB5000
160 INPUT⊙S2
170 PRINT «ВЫДЕРЖКА ВРЕМЕНИ НА»;J;«— ОМ ШАГЕ СОСТАВИ-
          ЛА»; (S1 — S2)/2000;«СЕКУНД»
180 ROTATE(A⊙,1)
190 NEXT J
200 PRINT «ПЕРВАЯ СЕКЦИЯ РУ-3,3 КВ ОТКЛЮЧЕНА»
210 END
500 REM   ПОДПРОГРАММА ВРЕМЯ
510 C=C—1
520 IFC<=0THEM540
530 GOTO510
540 RETURN
```

Поясним работу программы.

В операторе 20 задается длина символьных переменных A⊙ и B⊙, равная 1 байту (8 битов) каждая, и описывается одномерный цифровой массив (вектор) выдержек времени. В операторе 30 в процессе работы программы с клавиатуры терминала вводится число коммутируемых элементов.

Операторы 40 и 50 предназначены для ввода массива выдержек времени.

В строке 70 при помощи HEX-функции задается начальное значение символьных переменных A⊙ и B⊙, которые после выполнения строки примут следующие значения:

A ⊙ = 0 0 1 0 1 1 1 1
B ⊙ = 0 0 0 0 0 0 0 1

В операторе 90 организуется цикл по числу коммутации в схеме РУ 3,3 кВ.

В операторе 100 используется оператор XOR, реализующий логическую функцию «исключающее ИЛИ». После выполнения оператора будет «выключен» бит в разряде № 0 и значение символьной переменной A⊙ станет равным 0 0 1 0 1 1 1 0.

В операторах 110 и 120 выводится на экран (или на печать) номер шага и символьной переменной B ⊙ в шестнадцатеричном виде (оператор HEXPRINT), начиная с позиции № 20 строки. На первом шаге на экране инициируется информация:

СОСТОЯНИЕ КОММУТИРУЕМЫХ ЭЛЕМЕНТОВ НА 1-М ШАГЕ 2Е.

В операторе 130 цифровой переменной C присваивается очередное значение константы, определяющей выдержку времени между коммутируемыми элементами. В операторе 140 цифровой переменной S1 присваивается текущее значение таймера. В операторе 150 осуществляется передача управления подпрограмме ВРЕМЯ, начинающейся со строки с номером 500. В операторе 160 цифровой переменной S2 присваивается текущее значение таймера. В операторе 170 для последующего контроля вычисляется и инициируется временная выдержка, реализованная подпрограммой ВРЕМЯ. В операторе 180 при помощи оператора ROTATE осуществляется циклический сдвиг влево на один разряд константы, записанной в символьной переменной A ⊙. В операторе 190 закрывается цикл по числу коммутаций. В операторах 200 и 210 завершается работа программы.

При реализации подпрограммы ВРЕМЯ цикл организован в отличие от ранее приведенного примера при помощи оператора условной передачи управления IF.

4.6. Отладка программ

Отладка программ — это поиск, локализация и устранение ошибок, допущенных при ее разработке. Практика разработки программ свидетельствует, что этап ее отладки составляет значительную часть в общих затратах времени и средств на разработку. Поэтому решение проблемы повышения производительности труда программистов на этом этапе является одной из актуальных задач,

находящихся в поле зрения науки программирования. В частности, в ряде трудов [3,15,22] развивается идея верификации, т.е. математического доказательства корректности (правильности) программ в процессе разработки (правильность программы определяется как соответствие между программой и ее заданной функцией). Реализация идеи позволит получать готовый программный продукт с первого предъявления (с первого запуска). Однако разработанность методов математического доказательства программ на сегодняшний день находится на том уровне, когда их применение экономически не оправдано из-за сложности и трудоемкости.

Разработка и внедрение в практику программирования автоматических и полуавтоматических систем доказательств правильности программ — перспективное направление научных разработок, но это дело будущего. Сегодня же проверка правильности работы программ производится экспериментально на ЭВМ на основе сравнения получаемых результатов и специально разработанных задач (тестовых задач), результаты решения которых заранее известны или могут быть легко получены. В данном случае мы имеем дело ни с чем иным, как с процессом «ручного» моделирования этапов выполнения программы.

Однако при решении сложных инженерных задач далеко не всегда удается осуществить полноценное тестирование программ. В этих случаях большую помощь в проверке правильности работы программ могут оказать физические представления о процессе работы устройства, системы (в нашем случае электрифицированной железной дороги), о значениях их параметров. Эти сведения позволяют нам судить с достаточно высокой вероятностью о правильности расчетов по программе.

Следует помнить, что при помощи тестовых задач можно доказать наличие ошибок в программе, но не их отсутствие. Или, иначе, совпадение результатов «ручного» моделирования и испытаний на ЭВМ является необходимым, но недостаточным условием правильности программ. В программах, сдаваемых в промышленную эксплуатацию, всегда остается определенный процент неустраненных ошибок. Этим объясняется необходимость поиска и устранения ошибок в процессе эксплуатации.

Бейсик, как и другие алгоритмические языки, располагает стандартными средствами отладки программ. К ним в первую очередь относится оператор TRACE, который предназначен для пооператорной отладки программы.

Конструкция оператора нс TRACE [OFF]. Оператор TRACE OFF отменяет действие оператора TRACE. Пользователь получает возможность проследить за последовательностью выполнения фрагмента программы, заключенного между операторами TRACE и TRACE OFF. При отсутствии оператора TRACE OFF состояние TRACE будет сохраняться, начиная от номера строки с этим оператором до конца программы.

При работе оператора TRACE на экран выводится текст каждой выполненной строки и результаты присвоения значений переменным. Если в программе встречаются операторы управления программой (GOTO, GOSUB, IF), то на экране появляется сообщение TRANSFER TO нс о переходе к номеру строки, указанному в этом сообщении. Эта информация позволяет проконтролировать своевременность и правильность передач управления в программе и организации разветвляющихся и циклических процессов.

Следует, однако, отметить, что смена информации на экране происходит очень быстро. Поэтому для уменьшения скорости смены строк на экране увеличивают паузу после каждой строки при помощи оператора SELECT.

Конструкция оператора SELECT при задании продолжительной паузы для вывода информации на экран (печать) нс SELECT P [a]. Здесь a — цифра, определяющая величину паузы (a=0,1, ..., 9). Длительность паузы составляет a/6 с. Запись операторов SELECT P или SELECT P∅ определяет безостановочный режим.

При необходимости можно организовать так называемый пошаговый просмотр выполнения программы. В этом случае пауза задается пользователем и переход к выполнению следующей строки происходит только после нажатия клавиши пошагового выполнения программ HALT/STEP. Наиболее эффективно использование пошагового выполнения программы при наличии в программе оператора TRACE. Для перевода программы в указанный режим необходимо:

ввести оператор GOTO с указанием номера строки, в которой находится оператор TRACE (сам оператор GOTO набирается без номера), и нажать клавишу CR/LF (перевод строки/возврат каретки);

поочередным нажатием клавиши HALT/STEP проследить выполнение нужного фрагмента программы.

Отмена пошагового режима выполнения программы осуществляется нажатием клавиши CONTINUE (продолжать), а состояние TRACE — оператором TRACE OFF.

Для оценки текущих результатов в какой-либо точке программы можно воспользоваться также оператором STOP, предназначенным для программируемого останова ЭВМ при выполнении программы.

Конструкция оператора STOP имеет вид нс STOP [a], где a — символьная константа, используемая для напоминания пользователю о том, в какой точке произошел останов и какие действия следует выполнить.

Для продолжения счета по программе используют клавишу CONTINUE.

Число используемых в одной программе операторов STOP не ограничивается.

Как показывает опыт, большой процент ошибок при выполнении программы связан с неполнотой и некорректностью представления входных данных. Поэтому рекомендуется, во-первых, обязательно выводить на печать всю входную информацию после ее ввода в ЭВМ для последующего визуального контроля, а во-вторых, при больших объемах массивов входной информации дополнять программу средствами автоматического контроля этой информации, используя ее физические характеристики (например, диапазон изменения, неубывание, возрастание, четность, нечетность, положительность, отрицательность и т. п.).

Много неприятностей доставляют программистам ошибки, связанные с неправильными передачами управления при разветвляющихся и циклических процессах. Для устранения этих ошибок целесообразно использовать стандартные средства отладки, описанные выше.

Желательно также на время отладки включать в программу операторы PRINT для выдачи на печать промежуточных результатов в узловых точках программы (при входе и выходе из управляющих базовых структур, а при необходимости и внутри их), которые можно сопоставить с результатами ручного счета, подготовленными в соответствии с тестовой задачей. Следует вводить промежуточную печать и в места наиболее вероятного появления ошибок в программе, например, где есть операция деления и возможно появление нулевого значения в знаменателе, где есть операция извлечения квадратного корня и возможно появление отрицательного значения подкоренного выражения, где есть операции возведения в вещественную степень и возможно появление отрицательного основания и т. д.

Не следует, конечно, включать такую печать в тело цикла при большом числе повторений (например, сто или тысяча), В этом случае лучше воспользоваться возможностью проверки его работы в пошаговом режиме.

После окончания процесса отладки вся промежуточная (отладочная) печать должна быть из программы удалена или деактивирована. В последнем случае предполагается, что все операторы (или часть) промежуточной печати не удаляются из программы, а лишь отключаются впредь до возникновения в них потребности. Эта возможность реализуется при помощи условных выражений и операторов IF. Строка промежуточной печати должна выглядеть примерно следующим образом:

```
нсIFA %=∅THENнс1:PRINT«x=»;X
нс1————————————————————————
```

В этом операторе целочисленная переменная A % сравнивается с нулем. Если результат сравнения истина, то выполняется следующая строка ($нс_1$) программы, а печать значения переменной X не происходит. Если же значение A % отлично

от нуля, будет работать отладочная печать X в операторе PRINT. Таким образом, задавая значение A % в начале программы, появляется возможность управления совокупностью операторов промежуточной печати результатов.

Рассмотрим применение средств отладки программы на конкретном примере. Пусть требуется разработать и отладить программу расчета закона распределения числа поездов на межподстанционной зоне по формуле [24]

$$p(m) = C_n^m p^m q^{n-m} = C_n^m \left(\frac{N}{N_0}\right)^m \left(1 - \frac{N}{N_0}\right)^{n-m}$$

где $p(m)$ — вероятность появления m поездов на межподстанционной зоне; n — максимально возможное число поездов; p — вероятность занятия «нитки» графика поездом; N — число поездов, проходящих по межподстанционной зоне в сутки; N_0 — число «ниток» графика; g — вероятность противоположного события ($g = 1 - p$); C_n^m — число сочетаний из n по m.

При расчете C_n^m на ЭВМ целесообразно пользоваться рекуррентной формулой:

$$C_n^{m+1} = C_n^m \frac{n-m}{m+1},$$

которая выводится на основе использования основного свойства факториала $(m+1)! = (m+1)\,m!$

Число сочетаний C_n^{m+1} может быть представлено в виде:

$$C_n^{m+1} = \frac{n!\,(n-m)}{(m+1)\,m!\,(n-m)!},$$

где все члены, содержащие знак факториала, могут быть заменены одним выражением C_n^m. В результате получается приведенное выше рекуррентное соотношение, выражающее последующий член через предыдущий.

Рассмотрим алгоритм решения задачи (рис. 4.13). Он включает следующие структуры: следование (блоки *1—5*), «цикл-до» (блоки *6—18*), «если-то-иначе» (блоки *7—14*), которая вместе с блоками *6, 15, 16* и *17* составляет тело структуры «цикл-до». В свою очередь альтернатива структуры «если-то-иначе» (блоки *8—14*) представляет собой структуру «цикл-до», в теле которой содержится структура «если-то-иначе» (блоки *11* и *12*).

Отметим, что в общей структуре имеется конструкция из двух так называемых вложенных (друг в друга) циклов, не встречавшаяся ранее.

В блоках *2—5* вводятся исходные данные, рассчитываются параметры p и q, которые не изменяются при выполнении программы, и задается начальное значение управляющей переменной цикла по числу поездов. Переменная A % предназначена для управления отладочной печатью (блоки *11* и *12*), включенной в цикл расчета C_n^m по рекуррентному соотношению (блоки *8—14*).

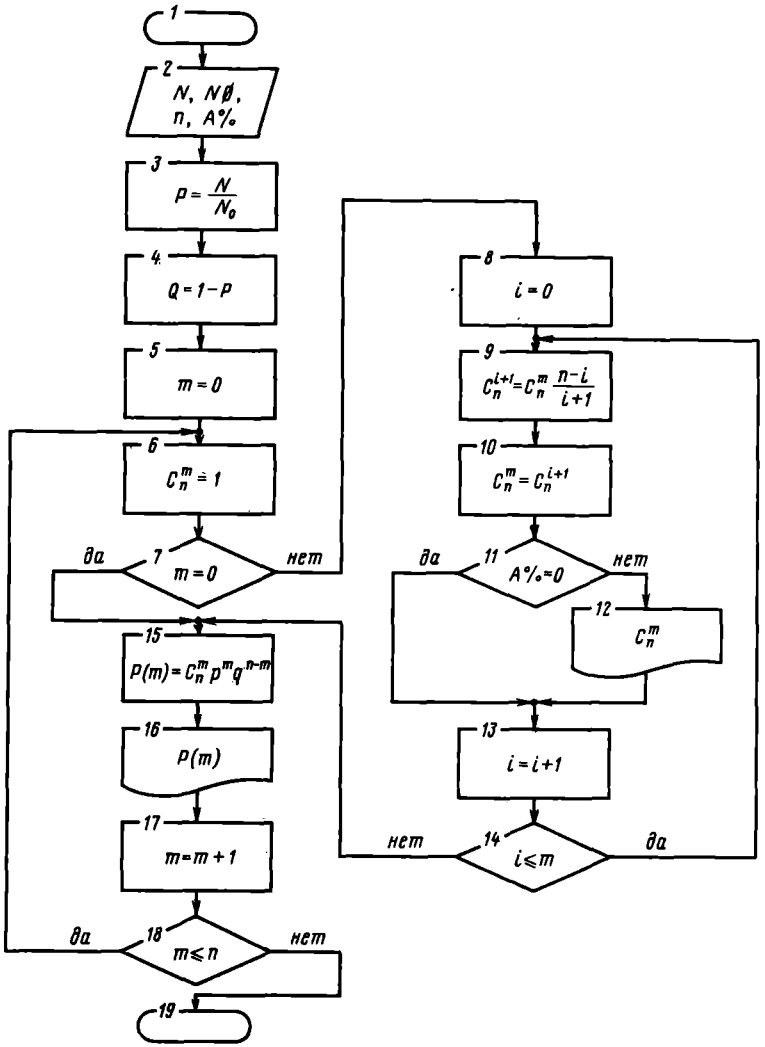


Рис. 4.13. Алгоритм расчета закона распределения числа поездов на межподстанционной зоне

Если вероятность рассчитывается для случая отсутствия поездов на межподстанционной зоне ($p(0)$), то рекуррентное соотношение не используется и управление из блока *7* передается сразу блокам *15*, *16* и *17*, в которых вычисляется вероятность заданного события и выполняются операции по дальнейшей организации циклических вычислений.

Составим таблицу идентификаций:

N	N_0	n	p	q	$p(m)$	m	C_n^m	C_n^{m+1}
N	NØ	N1	P	Q	P1	M	C	C1

В соответствии со схемой алгоритма (см. рис. 4.13) и таблицей идентификаций составим программу, включив в нее операторы отладки.

```
1ØREM  РАСЧЕТ ЗАКОНА РАСПРЕДЕЛЕНИЯ ЧИСЛА ПОЕЗДОВ
2ØSELECT P 9
3ØTRACE
4ØINPUT «ВВЕДИТЕ ЧИСЛО ПОЕЗДОВ В СУТКИ — N»,
        ЧИСЛО НИТЕЙ ГРАФИКА          — NØ,
        МАКСИМАЛЬНОЕ ЧИСЛО ПОЕЗДОВ НА МЕЖПОД-
        СТАНЦИОННОЙ ЗОНЕ — N1»,N,NØ,N1
5ØINPUT «ВВЕДИТЕ УПРАВЛЕНИЕ ПРОМЕЖУТОЧНОЙ ПЕЧАТЬЮ,
        1 — ВКЛЮЧИТЬ, Ø — ОТКЛЮЧИТЬ», A %
6ØPRINT «N=»; N, «NØ≠»; NØ, «N1≠»; N1
7ØPRINT «УПРАВЛЕНИЕ ПЕЧАТЬЮ РАВНО»; A %
8ØP=N/NØ:Q=1-P:M=Ø
9ØREM  НАЧАЛО ЦИКЛА РАСЧЕТА ВЕРОЯТНОСТЕЙ
1ØØFOR M=ØTO N 1
11ØC=1
12ØIF M=ØTHEN 19Ø
13ØREM  НАЧАЛО ЦИКЛА РАСЧЕТА ЧИСЛА СОЧЕТАНИЙ
14ØFOR I=ØTO M
15ØC1=C*(N1-I)/(I+1)
16ØC=C1
17ØIF A %=ØTHEN 18Ø:PRINT «ЧИСЛО СОЧЕТАНИЙ ИЗ»;
        N1; «ПО»; I+1; «РАВНО»; C
18ØNEXT I
19ØP1=C*P↑M*Q↑(N1-M)
2ØØPRINT «ВЕРОЯТНОСТЬ»; M; «ПОЕЗДОВ НА МЕЖПОД-
        СТАНЦИОННОЙ ЗОНЕ РАВНА»; P1
21ØNEXT M
22ØEND
```

Для отладки программы составим контрольный пример с таким расчетом, чтобы объем ручных вычислений был минимальным, но в то же время были проверены все ветви алгоритма и каждый цикл был выполнен не менее 2 раз.

Пусть исходные данные имеют следующие значения:

$$N=72,\ N_0=144,\ n=2.$$

Тогда внешний цикл должен выполниться 3 раза (при $m=0,1,2$) и должны быть получены следующие значения вероятностей:

$$p(0)=C_2^0p^0q^2=1\cdot1\left(\frac{1}{2}\right)^2=\frac{1}{4};$$

$$p(1)=C_2^1p^1q^1=2\,\frac{1}{2}\,\frac{1}{2}=\frac{2}{4};$$

$$p(2) = C_2^2 p^2 q^0 = 1\left(\frac{1}{2}\right)^2 1 = \frac{1}{4}$$

При выполнении внешнего цикла в первый раз рекуррентное соотношение для расчета C_n^m не должно использоваться, т. е. в структуре «если-то-иначе» управление должно быть передано в направлении истинности условия. При втором и третьем выполнении внешнего цикла, наоборот, должен работать внутренний цикл в структуре «если-то-иначе», причем при втором выполнении внешнего цикла — 1 раз, а при третьем — 2 раза. Соответственно результат промежуточной печати должен быть равен 2 при втором цикле и 2 и 1 при третьем.

Запуск программы командой RUN позволяет визуально проследить правильность выполнения всего алгоритма с экрана. В этом случае промежуточная отладочная печать может быть отключена, поскольку в состоянии TRACE все переменные, принимающие новые значения, автоматически выводятся на экран. Смена строк на экране будет происходить с интервалом, равным 1,5 с.

При необходимости можно перейти к пошаговому просмотру работы всей программы или ее фрагментов, используя оператор GOTO нс и клавишу HALT/STEP.

Отметим, что данная программа может быть дополнена еще проверкой правильности результата исходя из того, что сумма всех вероятностей должна равняться 1.

4.7. Организация диалога с ЭВМ

В параграфе 4.2 отмечалось, что для организации диалога человека с ЭВМ может быть использован оператор INPUT, при встрече с которым ЭВМ приостанавливает свои вычисления, инициирует на экране знак «?» и ожидает реакции пользователя. В зависимости от того, какая информация будет введена в ЭВМ в ответ на ее запрос, определяется дальнейшая работа программы.

Приведем пример небольшой программы (фрагмента), часто используемой для организации диалога:

```
4∅INPUT «ЕСЛИ ВЫ ХОТИТЕ ПОЛУЧИТЬ ОТВЕТ НА ЭКРАНЕ,
        НАЖМИТЕ ∅ CR/LF,
        ЕСЛИ ВЫ ХОТИТЕ ПОЛУЧИТЬ ОТВЕТ НА ПЕЧАТИ, НАЖМИТЕ 1 CR/LF», E
5∅IF E=∅ THEN 6∅:SELECT PRINT ∅ C (6∅):GOTO 7∅
6∅SELECT PRINT ∅5 (8∅)
7∅————————————————————————————————————————
```

Этот несложный диалог облегчает пользователю выбор и переключение каналов вывода информации на внешние устройства. Имя простой переменной E служит для хранения и последующего анализа ответа пользователя на запрос ЭВМ. Поясним конструкцию использованного в приведенном фрагменте программы оператора SELECT. Она имеет вид нс SELECT PRINT$u[(b)]$.

Здесь b — длина строки, а u — физический адрес устройства, на который будет осуществляться вывод информации.

Приведем перечень некоторых из этих адресов: Ø1 — клавиатура ЭВМ; Ø5 — экран; 18 — накопитель на гибких магнитных дисках; ØC — печатающее устройство.

В приведенном примере вывод информации на печать будет осуществляться с длиной строки, равной 60 символов, а на экран — 80.

Для вывода текста программы на внешние устройства может быть организован аналогичный диалог с использованием оператора SELECT LIST, не отличающегося по конструкции от оператора SELECT PRINT.

При организации диалога с ЭВМ оператор INPUT часто используют совместно с оператором PRINT. Оператор PRINT так же, как и оператор INPUT, допускает запись в теле оператора текстовой (символьной) информации и при своем выполнении инициирует ее без изменения на экране ЭВМ. Отличие оператора PRINT от оператора INPUT заключается в том, что при его выполнении не происходит прерывания работы ЭВМ и не появляется запрос. С помощью этого оператора на экран (или печать) может быть выведена порция информации, необходимая для активизации действий пользователя.

Рассмотрим пример организации диалога при вводе информации, необходимой для производства тягового расчета. Сценарий диалога может выглядеть следующим образом:

1. Запрос ЭВМ: Введите число элементов профиля на участке.
2. Ответ пользователя.
3. Запрос ЭВМ: Введите информацию об участке в такой последовательности: длина 1-го элемента профиля, уклон, ограничение по скорости движения на 1-м элементе и нажмите клавишу CR/LF; длина 2-го элемента профиля, уклон, ограничение по скорости движения на 2-м элементе и нажмите клавишу CR/LF и т. д. по 3 числа в каждой строке до окончания ввода всей информации.
4. Ответ пользователя.

Фрагмент программы.

```
1ØREM   ПРОГРАММА ПРОИЗВОДСТВА ТЯГОВОГО РАСЧЕТА
2ØREM   ВВОД ИНФОРМАЦИИ ОБ УЧАСТКЕ В РЕЖИМЕ ДИАЛОГА
3ØDIM S (1ØØ),U(1ØØ),V(1ØØ)
4ØINPUT «ВВЕДИТЕ ЧИСЛО ЭЛЕМЕНТОВ ПРОФИЛЯ НА УЧАСТКЕ»,N
5ØPRINT «ВВЕДИТЕ ИНФОРМАЦИЮ ОБ УЧАСТКЕ В СЛЕДУЮЩЕЙ ПОСЛЕДОВАТЕЛЬНОСТИ:»
6ØPRINT «ДЛИНА 1-ГО ЭЛЕМЕНТА ПРОФИЛЯ, УКЛОН, ОГРАНИЧЕНИЕ ПО СКОРОСТИ ДВИЖЕНИЯ НА 1-М ЭЛЕМЕНТЕ И НАЖМИТЕ КЛАВИШУ CR/LF»
```

```
7∅PRINT «ДЛИНА 2-ГО ЭЛЕМЕНТА ПРОФИЛЯ, УКЛОН, ОГРАНИ-
        ЧЕНИЕ ПО СКОРОСТИ ДВИЖЕНИЯ НА 2-М ЭЛЕМЕНТЕ
        И НАЖМИТЕ КЛАВИШУ CR/LF»
8∅PRINT «И Т. Д. ПО 3 ЧИСЛА В КАЖДОЙ СТРОКЕ ДО ОКОН-
        ЧАНИЯ ВВОДА ВСЕЙ ИНФОРМАЦИИ»
9∅FOR J=1 TO N
1∅∅INPUT SJ,U(J),V(J)
11∅PRINT «S(»;J;«)=»; S(J), «U(»;J;«)=»; U(J),«V(»;J;«)=»; V(J)
12∅NEXT J
```

Приведенная программа рассчитана на ввод не более 100 чисел по каждой переменной величине и в этих пределах универсальна, поскольку цикл ввода организован по переменной N. При необходимости ввода большего числа элементов надо увеличить размер массивов в операторе DIM.

В рассмотренных выше примерах ответ пользователя представлялся в цифровом виде. Оператор INPUT дает возможность воспринимать ответ пользователя и в символьной форме. В этом случае в теле оператора записываются символьные переменные, отличительной особенностью записи которых (напомним) является наличие в обозначении имени переменной знака денежной единицы.

Значение символьной переменной, которое присваивается ей в ответе пользователя, может быть представлено символьной константой в кавычках; без кавычек; в апострофах; при помощи HEX-функции.

Если значение символьной переменной не заключено в кавычки при вводе в машину, то запятые рассматриваются как окончания символьных констант, а пробелы, стоящие перед константами, игнорируются.

Приведем примеры фрагментов программ и ответов пользователя:

а) 4∅INPUT «НАЗОВИТЕ ВАШУ ФАМИЛИЮ И ИНИЦИАЛЫ», A⊙

В результате выполнения оператора 4∅ на экране инициируется запрос ЭВМ—НАЗОВИТЕ ВАШУ ФАМИЛИЮ И ИНИЦИАЛЫ? за которым должен последовать, например, ответ пользователя — ИВАНОВ И. И.

б) 5∅INPUT «НАЗОВИТЕ КОД И НОМЕР УЧЕБНОЙ ГРУППЫ», B⊙

В результате выполнения оператора 5∅ на экране инициируется запрос ЭВМ — НАЗОВИТЕ КОД И НОМЕР УЧЕБНОЙ ГРУППЫ? Ответ пользователя — ЭЛ-341.

Для оформления результатов работы по программе в качестве примера могут быть записаны операторы:

```
3∅∅PRINT TAB(1∅); «РАБОТУ ВЫПОЛНИЛ»; A⊙
31∅PRINT  TAB(1∅); «ГРУППА»; B⊙
```

На листинге будет напечатано, начиная с 10-й позиции строки:

```
РАБОТУ ВЫПОЛНИЛ ИВАНОВ И. И.
ГРУППА ЭЛ-341
```

Операторы 4∅ и 5∅ могут быть объединены в один, например:

```
4∅INPUT «НАЗОВИТЕ ВАШУ ФАМИЛИЮ И ИНИЦИАЛЫ, КОД И
        НОМЕР ГРУППЫ», A⊙, B⊙
```

Ответ пользователя может иметь следующий вид:

```
ИВАНОВ И. И., ЭЛ-341.
```

Учитывая, что запятая в ответе без кавычек рассматривается как разделитель между символьными константами, переменной A⊙ будет присвоено значение ИВАНОВ И. И., а переменной B⊙ — ЭЛ-341. Если же в состав самой константы входят (по смыслу) запятые и пробелы, то такие символьные константы следует заключать в кавычки.

Символьные константы, заключенные в апострофы, позволяют производить обработку последовательностей строчных букв, отсутствующих в алфавите языка. Следует помнить, что стандартная длина символьной переменной или элемента символьного массива равняется 16 символам. Если ответ пользователя будет превышать 16 символов, символы сверх 16 игнорируются.

Длину символьной переменной (отличную от стандартной) можно объявить операторами описания размеров DIM и COM в пределах от 1 до 253, например:

```
1∅DIM B⊙4∅
5∅INPUT «НАЗОВИТЕ ВАШУ ФАМИЛИЮ, ИНИЦИАЛЫ, КОД И
        НОМЕР ГРУППЫ, А ТАКЖЕ НОМЕР ВАРИАНТА ЗАДАНИЯ», B⊙
```

На запрос ЭВМ последует ответ пользователя:

```
?«ИВАНОВ И. И., ЭЛ-341, ВАРИАНТ № 25».
```

Далее в результате выполнения оператора 31∅PRINT TAB(5); B ⊙ на листинге, начиная с 5-й позиции строки, получим распечатку ИВАНОВ И. И., ЭЛ-341, ВАРИАНТ № 25.

Используя условные выражения и оператор IF, можно производить анализ ответов пользователя, представленных в символьной форме, и организовать разветвления в программе в зависимости от результатов анализа. Ниже приводится пример программы простейшего диалога с анализом ответов пользователя (типа ДА-НЕТ), представленного в символьной форме:

```
1∅PRINT «ОТВЕЧАЙТЕ НА ВОПРОСЫ ЭВМ В ОДНОЙ ИЗ СЛЕ-
        ДУЮЩИХ ФОРМ: ДА ИЛИ НЕТ»
2∅A  =«ДА»
3∅INPUT «ПРЕДПОЛАГАЕТСЯ ЛИ ВВОД ДОПОЛНИТЕЛЬНОЙ ИН-
        ФОРМАЦИИ», B⊙
4∅IF B⊙=A⊙ THEN 1∅∅
5∅.

1∅∅.
```

Рассмотрим еще один пример диалоговой программы, в которой анализируется ответ пользователя, представленный так:

```
1∅DIM A⊙2∅, B⊙2∅, E⊙2∅
2∅A⊙=«F=2↑((V∅-V1)/D)»
25E⊙=A⊙
3∅GOSUB 5∅∅
4∅C1=S
5∅PRINT «ВВЕДИТЕ ФОРМУЛУ ДЛЯ РАСЧЕТА ОТНОСИТЕЛЬНОГО
        ИЗНОСА ВИТКОВОЙ ИЗОЛЯЦИИ ТРАНСФОРМАТОРА»
6∅PRINT «ПРИ НАБОРЕ ФОРМУЛЫ ИСПОЛЬЗУЙТЕ СЛЕДУЮ-
        ЩИЕ ОБОЗНАЧЕНИЯ:»
7∅PRINT «ОТНОСИТЕЛЬНЫЙ ИЗНОС — F;
        ТЕМПЕРАТУРА НАИБОЛЕЕ НАГРЕТОЙ ТОЧКИ ОБМОТ-
        КИ — V∅;
        БАЗОВАЯ ТЕМПЕРАТУРА НАИБОЛЕЕ НАГРЕТОЙ ТОЧ-
        КИ ОБМОТКИ — V1»
8∅PRINT «ТЕМПЕРАТУРНЫЙ ИНТЕРВАЛ — D»
9∅INPUT B⊙
95E⊙=B⊙
1∅∅COSUB 5∅∅
11∅C2=S
12∅IFC2=C1 THEN130:GOTO 14∅
13∅PRINT «ФОРМУЛА ВЕРНА»:GOTO 16∅
14∅PRINT «ФОРМУЛА»; B⊙; «ВВЕДЕНА НЕВЕРНО»
15∅PRINT «НАДО БЫЛО ВВЕСТИ»; A⊙
16∅END
5∅∅REM ПОДПРОГРАММА ПОЛУЧЕНИЯ ДЕСЯТИЧНОГО ЭКВИ-
        ВИВАЛЕНТА ЗНАЧЕНИЯ СИМВОЛЬНОЙ ПЕРЕМЕННОЙ
51∅S=∅
52∅FOR J=1TO2∅
53∅S1=VAL(STR(E⊙,J,1))
54∅IF S1=2∅ THEN 56∅
545IF S1=3∅ THEN 56∅
55∅S=S+S1
56∅NEXT J
57∅RETURN
```

Пояснения к программе:

в операторе 1∅ задается нестандартная длина символьных переменных A⊙, B⊙, E⊙ (по 20 символов каждая);

в операторе 2∅ в символьную переменную A⊙ записывается формула для расчета относительного износа витковой изоляции тягового трансформатора;

при помощи операторов 25, 3∅ и 4∅ организуется обращение к подпрограмме и получение десятичного эквивалента формулы износа;

в операторах 5∅—8∅ содержится порция информации для активизации пользователя;

в операторе 9∅ осуществляется запрос на ввод формулы;

в операторах 95 и 11∅ организуется обращение к подпрограмме и получение десятичного эквивалента формулы, введенной пользователем;

в операторе 12∅ осуществляется контроль правильности введенной пользователем формулы (сравниваются десятичные эквиваленты эталонной формулы оператора 2∅ и введенной в операторе 9∅);

в операторах 13∅—14∅ выводится информация о результатах сравнения и при необходимости дается правильный ответ в операторе 15∅.

Подпрограмма получения десятичного эквивалента значений символьных переменных основана на представлении каждого символа в виде десятичного числа при помощи функции VAL и суммирования получаемых чисел в цикле, организованном по числу символов в переменной.

Символы ∅ и пробел из рассмотрения исключаются при помощи операторов 54∅ и 545.

4.8. Программирование обработки файлов на магнитных дисках

Организация файлов на магнитных дисках. Магнитные диски представляют собой устройства внешней памяти прямого доступа. Использование программами на Бейсике внешней памяти существенно расширяет возможности и удобства эксплуатации компьютеров. Информация на магнитных дисках хранится в виде файлов.

Файлы — это совокупность логически связанных данных. Различают два типа файлов: файлы данных (D) и программные файлы (P).

Каждый файл состоит из одной или нескольких записей. Запись на Бейсике определяет список данных в одном операторе обмена информацией DATA SAVE DC (запись в файл) или DATA LOAD DC (чтение из файла).

Одна запись занимает на диске целое число секторов. На одной дорожке гибкого магнитного диска размещаются 13 секторов. Общее количество дорожек — 77. Таким образом, на гибком диске имеется 1001 сектор. Каждый сектор имеет информационную емкость 256 байтов. Емкость магнитного диска — 250 Кбайтов.

Сектора нумеруются, начиная с нуля. Номер сектора называется его адресом. На дисковое устройство могут быть установлены одновременно два диска: типа F (фиксированный) и типа R (съемный).

Бейсик позволяет работать с дисковыми устройствами в режиме каталога и в режиме адресации секторов.

Режим каталога является более простым для пользователя, поскольку позволяет обращаться к файлам по их символическим именам. ЭВМ автоматически следит за размерами и размещением каждого каталогизированного файла.

Режим адресации секторов является более гибким, но и более сложным. Поскольку в учебном процессе предпочтительнее режим каталога, в дальнейшем будем рассматривать именно этот режим.

Режим каталога файлов. Каталогом называется структура хранения файлов на диске. Каталог состоит из двух зон: зоны каталога, где хранятся файлы, и зоны указателя каталога, где хранится вся справочная информация о файлах.

Признаком режима каталога являются буквы DC (DISK CATALOG) в операторах управления файлами, представленных в табл. 4.7

Указатель каталога можно вывести на экран или напечатать, используя оператор LIST DC, например:
LIST DC F или LIST DC R

Указатель каталога имеет вид:

```
INDEX SECTORS
END CAT. AREA
CURRENT END
NAME TYPE START END USED
```

Здесь INDEX SECTORS — число секторов, отведенных под зону указателя каталога; END CAT. AREA — адрес последнего сектора, отведенного под зону каталога; CURRENT END — адрес первого свободного сектора зоны каталога; NAME — имя программного файла или файла данных; TYPE — тип файла (D — файл данных, P — программный файл, SD или SP — ликвидируемый файл); START — начальный адрес сектора файла; END — конечный адрес сектора файла; USED — число секторов, использованных в файле.

Таблица 4.7

Обозначение оператора	Назначение	Конструкция оператора
DATA SAVE DC OPEN	Открытие вновь создаваемых файлов данных и временных рабочих файлов	нсDATA SAVE DC OPEN $a[\odot][b,]\begin{cases}(s)c\\ \text{TEMP},\\ d_1, d_2,\end{cases}$ где a — тип диска (F или R); $\odot$ — параметр, требующий контрольного считывания данных после записи; b — номер файла (#∅, #1, #7); s — если s — цифровая величина, то она означает число секторов, отводимых под создаваемый файл, если же s — символ, то это имя файла, отмеченного как вычеркнутый (на месте которого будет создан новый файл); c — имя создаваемого файла (символьная константа, переменная или элемент массива этого типа), $c \leqslant 0$ символов;

Окончание табл. 4.7

Обозначение оператора	Назначение	Конструкция оператора
		TEMP — параметр организации временного рабочего файла; d_1, d_2 — адрес соответственно начального и конечного сектора
DATA LOAD DC OPEN	Открытие файла для записи или чтения, созданного ранее	нсDATA LOAD DC OPEN $a[b]\begin{cases} c \\ \text{TEMP}, d_1, d_2 \end{cases}$
DATA SAVE DC CLOSE	Закрытие файла	нс DATA SAVE DC CLOSE $\begin{bmatrix} b \\ \{\text{ALL}\} \end{bmatrix}$ где ALL — параметр, указывающий на закрытие всех файлов
DATA SAVE DC	Запись в файл в виде одной логической записи	нс DATA SAVE DC $[\odot]\,[b,]\begin{cases} c_1, c_2, \ldots c_n \\ \text{END} \end{cases}$ где c_1, c_2, ..., c_n — список данных для записи на диск (константы, переменные, выражения, элементы массивов или массивы); END — параметр окончания файла
DATA LOAD DC	Считывание логической записи из файла	нс DATA LOAD DC $[b,]c_1, c_2, \ldots c_n$
SAVE DC	Запись программы на диск	нс SAVE DC $a[\odot]\,[b,]\begin{bmatrix} \text{P} \\ \text{T} \\ [G] \end{bmatrix}[(s)]c[\text{нс}_1]\,[,\text{нс}_2]$ где P, T и G — параметры, обозначающие признак защиты программы при записи на диск; s — если s — цифровая переменная, то это число секторов, резервируемых сверх необходимого количества, занимаемого программой; если же s — символ, то это имя старого (ликвидируемого) файла; c — имя записываемой программы; нс_1, нс_2 — соответственно номер начальной и конечной строки программы
LOAD DC	Загрузка программы в ОЗУ	нс LOAD DC $a[b,]c[\text{нс}_1]\,[,\text{нс}_2]$
LIST DC	Индикация (распечатка) зоны указателя каталога	нс LIST DC $a[b]\,[c]$ где c — символьная константа или переменная
SCRATCH	Удаление файла из указателя каталога	SCRATCH, $a[b]c_1, c_2, \ldots, c_n$, где c_1, c_2, ..., c_n — имена файлов или программ, подлежащих удалению из каталога

Наличие символьной константы или переменной позволяет просматривать каталог выборочно, например:

```
1∅A⊙=«ЭЛ321»
2∅LIST DC F A⊙
```

В результате выполнения этих операторов на экране инициируются программные файлы и файлы данных, принадлежащие студентам группы с шифром ЭЛ-321.

Данные или программа могут быть удалены из указателя каталога при помощи оператора SCRATCH. Для этого в теле оператора указывается перечень имен файлов, подлежащих удалению.

Имя файла представляет из себя символьную константу, не превосходящую восьми символов.

Запись и считывание программных файлов. Для записи (чтения) программ используются операторы SAVE DC (LOAD DC).

Приведем пример простейшей формы записи операторов (опущены все необязательные параметры):

```
SAVE DCR «ЭЛ321∅15»
```

При считывании этой программы с диска необходимо SAVE заменить на LOAD.

Если требуется откорректировать программу, загруженную с диска в оперативную память ЭВМ, и вновь записать на старое место на диске под тем же (или другим) именем, необходимо:

удалить программный файл из указателя каталога оператором SCRATCH;

записать откорректированную программу на старое место, используя параметр *S* в операторе SAVE DC.

Если количество секторов, занятых старой программой, недостаточно для записи откорректированной, то будет выдано сообщение об ошибке.

Если в операторе SAVE DC не задан параметр *s*, то откорректированная программа записывается в свободной области каталога и ей отводится столько секторов, сколько ей требуется. Однако при многократной перезаписи может произойти быстрое переполнение диска.

Параметры $нс_1$ и $нс_2$ в операторах записи (чтения) позволяют записывать (считывать) не всю, а часть программы, например

```
1∅LOAD DCR «ЭЛ321∅15» 1∅∅,5∅∅
```

Если задан только первый параметр $нс_1$, то записывается (считывается) программа, начиная с указанного номера строки и до конца. Если задан только второй параметр, то программа считывается с первой строки и до строки $нс_2$. В приведенном примере в память ЭВМ загружается часть программы, начиная с номера строки 100 и до номера 500.

Параметры P, T и G обозначают признак защиты программы при записи на диск:

P — запись нетранслированной программы с защитой;

T — запись транслированной программы без защиты;

G — запись транслированной программы с защитой.

Использование параметров P или G указывает на то, что программа защищена от распечатки и повторной записи. Защищенная программа может быть лишь загружена и выполнена.

Запись и считывание файлов данных. Для записи и считывания файлов данных используются дисковые операторы DATA SAVE DC OPEN, DATA LOAD DC OPEN, DATA SAVE DC CLOSE, DATA SAVE DC и DATA LOAD DC.

Первые два оператора обеспечивают открытие файлов. В процесс открытия файла входит заполнение таблицы устройств, представленной в памяти ЭВМ в виде строк.

Программой могут одновременно обрабатываться 8 файлов. Для открытия нескольких файлов данных им необходимо присвоить разные номера. Эти номера являются логическими номерами устройств, на которых размещаются файлы. Приведем некоторые из них: Ø1 — клавиатура; Ø5 — экран; Ø8 — кассетный магнитофон; 18 — накопитель на гибких магнитных дисках; 1C — накопитель на жестких магнитных дисках; ØC — печатающее устройство.

Соответствие между логическими номерами и физическими адресами устройств устанавливается при помощи оператора SELECT, имеющего следующую форму:

нс SELECT*b u a*

где *b* — логический номер устройства (номер файла); *u* — физический адрес устройства (ФАУ); *a* — тип диска.

Например, в операторе 5ØSELECT # Ø18R,#5 18 логическому номеру устройства Ø назначается сменный гибкий магнитный диск с физическим адресом 18, а логическому номеру # 5 — фиксированный гибкий магнитный диск с тем же физическим адресом.

Работа с файлом данных возможна только после его открытия, т. е. занесения его параметров в таблицу устройств в строку, соответствующую объявленному номеру файла. Если файл создается на диске впервые, то при помощи оператора DATA SAVE DC OPEN одновременно с открытием файла резервируется заданное в этом операторе число секторов, например:

```
5ØSELECT #18R, #1 18R, # 2 18R
3ØØDATA SAVE DC OPEN R #Ø,(2Ø)«ТОК»
31ØDATA SAVE DC OPEN R# 1, (2Ø) «ВРЕМЯ»
32ØDATA SAVE DC OPEN R # 2, (2Ø) «ДЛИНА»
```

В этих примерах открываются сразу три файла, в которых будут размещены результаты тягового расчета — ток электровоза, время его хода и расстояние от начала участка. Все три файла

создаются на сменном гибком диске и под каждый из них резервируется по 20 секторов.

Возможна несколько иная форма записи оператора DATA SAVE DC OPEN, в которой в качестве параметров *s* и *c* выступают не константы, а переменные, например:

```
37ØDIM A ⊙ 8
38ØA⊙= «ТОК» : K=2Ø
39ØDATA SAVE DC OPEN R#Ø, (K) A ⊙
```

Следующий пример иллюстрирует возможность создания нового файла на месте, занимаемом другим файлом:

```
 5ØSCRATCH R#Ø «ТОК»
1ØØDATA SAVE DC OPEN R#Ø («ТОК») «ВРЕМЯ»
```

В операторе 5Ø файл с именем ТОК удаляется из указателя каталога и по оператору 1ØØ создается новый файл ВРЕМЯ на том же самом месте.

Для открытия ранее созданного файла данных на запись или чтение используется оператор DATA LOAD DC OPEN. В этом операторе, естественно, уже отсутствует параметр *s* (число секторов или имя файла, имеющего статус «удаляемый»).

В качестве обобщающего примера использования операторов записи и считывания файлов данных рассмотрим две небольшие программы. Первая программа создает на диске файл данных с результатами тягового расчета, а вторая считывает и распечатывает содержимое файла.

Программа создания файла:

```
1ØREM  СОЗДАНИЕ ФАЙЛА «ТЯГОВЫЙ РАСЧЕТ»
2ØDATA 5ØØ, 8ØØ, 1Ø5Ø, 25ØØ, 125Ø, Ø, Ø,
       Ø, 2.5, 3.5, 5.Ø, 6.2, 7.Ø, 2Ø,
       Ø, 3.1, 4.2, 6.5, 8.Ø, 2.5, 25
3ØDIM  I(7), L(7), T(7)
4ØMAT READ I, L, T
5ØDATA SAVE DC OPEN F#5, (2) «РАСЧЕТ»
6ØDATA SAVE DC «ТЯГОВЫЙ РАСЧЕТ», 7,I(), L (), T()
7ØDATA SAVE DC END
8ØEND
```

Программа считывания и распечатки содержимого файла:

```
1ØREM  ЗАГРУЗКА ФАЙЛА В ОЗУ
2ØDIM A ⊙ 16, I(7), L(7), T(7)
3ØDATA LOAD DC OPEN F # 5, «РАСЧЕТ»
4ØDATA LOAD DC A ⊙, N,I(), L(), T()
5ØDATA SAVE DC CLOSE # 5
6ØPRINT TAB (2Ø), A⊙
7ØMAT PRINT I
8ØMAT PRINT L
9ØMAT PRINT T
1ØØPRINT «ЧИСЛО СЕЧЕНИЙ ТЯГОВОГО РАСЧЕТА=»; N
```

Операторы 2∅ — 4∅ в программе создания файла предназначены для заполнения массивов результатами тягового расчета.

Оператор 5∅ открывает вновь создаваемый файл с номером 5 на фиксированном гибком магнитном диске (F) с именем «РАСЧЕТ».

Резервируются два сектора. Расчет потребного числа секторов производится следующим образом:

рассчитывается общее число цифр в записи и умножается на 10 байтов, отводимых под каждое вещественное число (22 × 10 = = 220 байтов),

рассчитывается число символов в символьной константе, включая пробел, и прибавляется 2 (14 + 2 = 16 байтов);

определяется суммарный потребный объем памяти в байтах (220 + 16 = 236 байтов);

определяется число секторов исходя из того, что каждый сектор содержит 256 байтов, т. е. в нашем примере вся информация уместится в одном секторе.

Однако число секторов, резервируемых под файл, должно быть хотя бы на единицу больше требуемого, так как последний сектор файла отводится для служебной информации.

В операторе 6∅ формируется одна запись на диске.

По оператору 7∅ в указатель каталога заносится начальный и конечный адреса сектора, отведенного под файл и фактическое их число.

В операторе 2∅ программы загрузки файла описываются размеры массивов I, L, и T, в которые будут считаны данные из файла на диске, а также длина символьной переменной. Отметим, что в нашем примере описание символьной переменной необязательно, так как ее длина совпадала со стандартной длиной.

В операторе 3∅ открывается ранее созданный файл «РАСЧЕТ» с номером 5 на фиксированном диске.

В операторе 4∅ происходят считывание данных из файла и присвоение: символьной переменной A⊙ — значения символьной константы «ТЯГОВЫЙ РАСЧЕТ»; простой переменной N — числа точек тягового расчета; массивам I, L и T — результатов тяговых расчетов.

В операторах 6∅ — 1∅∅ производится распечатка считанной информации.

После завершения операции считывания данных из файла # 5 в операторе 4∅ в следующем операторе 5∅ выполняется операция закрытия этого файла. При закрытии файла начальному, конечному и текущему адресам секторов файла в нашем случае присваиваются нули. Последующее обращение дисковых операторов к этому файлу приведет к выдаче сообщения об ошибке. Тем самым исключается возможность ошибочного доступа и разрушения данных в этом файле.

4.9. Дополнительные возможности языка Бейсик

Помеченные подпрограммы. Такие подпрограммы не имеют принципиальных отличий от подпрограмм, рассмотренных ранее. Использование помеченных подпрограмм позволяет сделать запись программ на Бейсике более компактной и наглядной. Способ оформления и вызова помеченных подпрограмм на Бейсике очень похож на аналогичные средства языка Фортран.

Помеченные подпрограммы оформляются при помощи оператора DEF FN′, имеющего следующую конструкцию:

нс DEF FN′a (c_1, c_2, c_n)

тело подпрограммы

нс$_1$RETURN

В этой конструкции a — номер подпрограммы из диапазона от 0 до 255, c_1, c_2, ..., c_n — формальные параметры.

Вызов подпрограмм производится для помеченных подпрограмм не по номеру строки, а по номеру подпрограммы, записанному в операторе GOSUB′, имеющем следующую конструкцию:

нс GOSUB′a (d_1, d_2, d_n)

В ней d_1, d_2, ...,d_n — фактические параметры, присваиваемые формальным c_1, c_2,..., c_n в операторе DEF FN′

Необходимо соблюдать следующие правила соответствия формальных и фактических параметров:

число параметров в операторе GOSUB′ должно совпадать с числом параметров, указанных в операторе DEF FN′;

должно выполняться соответствие параметров по типу в порядке их следования в списке, т.е. если формальный параметр — символьная (цифровая) величина, то и соответствующий ему фактический параметр — тоже величина символьного (цифрового) типа;

в качестве формального параметра может использоваться простая переменная или элемент массива;

в качестве фактического параметра может использоваться константа, переменная, элемент массива или арифметическое выражение;

переменные, массивы и элементы массивов — результаты выполнения подпрограммы, в список параметров не включаются.

В качестве иллюстрации приведем текст программы вычисления полного комплексного сопротивления тяговой сети переменного тока z_0 (см. п. 4.3) с использованием помеченной подпрограммы:

```
1ØREM  ПРОГРАММА РАСЧЕТА Z1КС
2ØDEF FNL (X) = Ø.Ø628 * LOG (1.28 * X)
3ØDATA  1ØØ, 12Ø, 6, 1.3, 7.2, Ø.177, Ø.158
4ØREAD S1, S2, D1, D2, D3, K, T
```

```
5ØREM   РАСЧЕТ ИНДУКТИВНЫХ СОПРОТИВЛЕНИЙ
6ØR1=SQR(S1/#PI)*1E-3:R2=SQR(S2/#PI)*1E-3
7ØX1=FNL(D1/R1):X2=FNL(D2/R2):X3=FNL(D2/R1)
8ØX4=Ø.Ø628*LOG(D3/D2)
9ØREM   РАСЧЕТ ПО ФОРМУЛЕ
1ØØGOSUB' 5(K, X1, T, X2, 3)
11ØS1=A3:S2=B3
12ØGOSUB' 5(K, X3, Ø, X4, 3)
13ØGOSUB' 5(S1, S2, A3, B3, 1)
14ØS1=A3:S2=B3
15ØGOSUB' 5(T, X2, K, X1, 1)
16ØGOSUB' 5(S1, S2, A3, B3,4)
17ØPRINT «АКТИВНОЕ СОПРОТИВЛЕНИЕ КОНТАКТНОЙ СЕТИ=»;
         A3; «ОМ/КМ»
18ØPRINT «ИНДУКТИВНОЕ СОПРОТИВЛЕНИЕ КОНТАКТНОЙ СЕ-
         ТИ=»; B3; «ОМ/КМ»
19ØPRINT TAB(1Ø);«R+JX=»; A3;«+J»; B3; «ОМ/КМ»
48ØREM   АРИФМЕТИКА КОМПЛЕКСНЫХ ЧИСЕЛ
49ØDEF FN' 5(A1, B1, A2, B2, N)
5ØØON N GOTO 51Ø, 53Ø, 55Ø, 57Ø
51ØREM   СЛОЖЕНИЕ
52ØA3=A1+A2:B3=B1+B2: RETURN
53ØREM   ВЫЧИТАНИЕ
54ØA3=A1-A2:B3=B1-B2:RETURN
55ØREM   УМНОЖЕНИЕ
56ØA3=A1*A2-B1*B2:B3=A1*B2+A2*B1:RETURN
57ØREM   ДЕЛЕНИЕ
58ØC=A2↑2+B2↑2
59ØA3=(A1*A2+B1*B2)/C
6ØØB3=(A2*B1-A1*B2)/C:RETURN
```

В этой программе операторы 1Ø — 9Ø приведены без изменений. В операторах 1ØØ, 12Ø, 13Ø, 15Ø и 16Ø производится обращение к подпрограмме, выполняющей операции над комплексными числами и оформленной при помощи оператора DEF FN' (строки 48Ø — 6ØØ). В качестве фактических параметров в нашем примере использованы константы и переменные. Из программы исключены операторы присвоения, присутствовавшие ранее в перечисленных выше строках, поскольку они выполняются автоматически при обращении к помеченной программе. В отличие от ранее рассмотренного примера в данном примере имеется лишь одна подпрограмма, осуществляющая операции сложения, вычитания, умножения или деления комплексных чисел с четырьмя выходами. Конкретная операция над комплексными числами определяется значением, присваиваемым формальному параметру N (1— сложение, 2 — вычитание, 3 — умножение, 4 — деление).

Помеченные подпрограммы могут располагаться в любом месте программы, но они не должны встречаться по ходу вычислений. Вход в помеченные подпрограммы должен осуществляться только по оператору GOSUB'

Выход из подпрограммы осуществляется при помощи оператора RETURN. Управление передается всегда оператору, следующему за оператором GOSUB′, по которому осуществлялся вызов помеченной подпрограммы.

Специальные функции*. Для сокращения времени набора программы при вводе часто встречающихся выражений, терминов, понятий и т. п. применяется вторая форма оператора DEF FN′, имеющая конструкцию:

нс DEF FN′$ac_1[;c_2;...;c_n]$, в которой a — номер специальной функции из диапазона от 0 до 31, c_1, c_2,...,c_n — символьные константы.

Подпрограмма-функция состоит из одной строки, оператор RETURN опускается. Вызов подпрограммы-функции осуществляется с клавиатуры.

Примеры записи специальных функций:

```
1∅DEF FN∅  «МИИТ»; «Электрификация железнодорожного транспорта»
2∅DEF FN1  «МИКРОПРОЦЕССОР КР580ИК80А»
3∅DEF FN2  «MAT PRINT»
4∅DEF FN3  «MAT INPUT»
```

Если требуется ввести оператор 1∅∅ MAT INPUT A, то при наличии строки с номером 3∅ действия пользователя сводятся к следующим операциям:

набрать номер строки — 1∅∅;

нажать клавишу 2 зоны СФ (специальных функций);

нажать клавишу, соответствующую символу A;

нажать клавишу «перевод строки/возврат каретки».

В приведенном примере достигается экономия времени за счет того, что слово MAT INPUT набирается одним нажатием клавиши.

Оператор PRINTUSING. Этот оператор используется для вывода цифровой и символьной информации на экран (печать) в виде, задаваемом пользователем с помощью специального формата. Этот оператор расширяет возможности оформления результатов, представляемые оператором PRINT.

Конструкция оператора имеет вид:

$$\text{нс PRINTUSING}\,[u,]\left\{\begin{matrix}c\\ \text{нс}\end{matrix}\right\}\,[\,,a_1\{;\}\,a_2\{;\}\,\ldots\{;\}\,a_n]$$

В этой конструкции u — устройство (логический номер или физический адрес устройства); c — символьная константа или переменная; нс — номер строки, в которой находится оператор IMAGE, задающий формат для вывода элементов списка; a_1, a_2,..., a_n — список вывода.

Оператор PRINTUSING может использоваться самостоятельно или совместно с оператором IMAGE (%), который может нахо-

* Следует отличать от общепринятого понятия этого термина в математике.

диться в любом месте программы и использоваться несколькими операторами PRINTUSING.

Применение оператора PRINTUSING рассмотрим на примере вывода на печать результатов пересчета тяговой и скоростной характеристики (см. п. 3.4):

```
11∅FOR J=1TO N
12∅PRINTUSING 14∅,J,F2(J),J,V2(J)
13∅NEXT J
14∅%F2(##)=###.#V2(##)=##.#
```

В операторе 12∅, помимо элементов списка вывода, записываемых и в операторе PRINT, присутствует ссылка (номер строки) на оператор IMAGE, в котором записан формат выводимых значений (символами #) и сопроводительный поясняющий текст.

Отметим, что если значение силы тяги или скорости будет превышать отведенное число знаков в целой части формата, то на печать будет выдан формат (сообщение об ошибке).

Фрагмент приведенной программы может быть представлен в ином виде с использованием символьной переменной:

```
11∅DIM A⊙25
12∅A⊙=«F2(##)=###.#V2(##)=##.#»
13∅FOR J=1TO N
14∅PRINTUSING A⊙,J,F2(J),J,V2(J)
15∅NEXT J
```

Оператор KEYIN. Он позволяет управлять выполнением программы с помощью клавиш терминала ЭВМ.

Оператор имеет следующую конструкцию:

нс KEYIN c, $нс_1$, $нс_2$, в которой c — символьная переменная, а $нс_1$, $нс_2$ — номера строк.

Оператор работает следующим образом. Если к моменту выполнения оператора нажата какая-либо клавиша, то шестнадцатеричный код этой клавиши записывается в первый байт символьной переменной, указанной в теле оператора KEYIN, и управление передается строке с номером $нс_1$. Строке $нс_2$ управление передается в случае, если нажата одна из клавиш зоны специальных функций (СФ). При отсутствии нажатых клавиш оператор KEYIN пропускается.

Рассмотрим пример, в котором возможности, представляемые оператором KEYIN, реализуются при построении имитационной игры «Управление движением поезда» (рис. 4.14).

Задание режима движения поезда на каждом шаге интегрирования дифференциального уравнения движения поезда осуществляется нажатием соответствующих клавиш консольной клавиатуры.

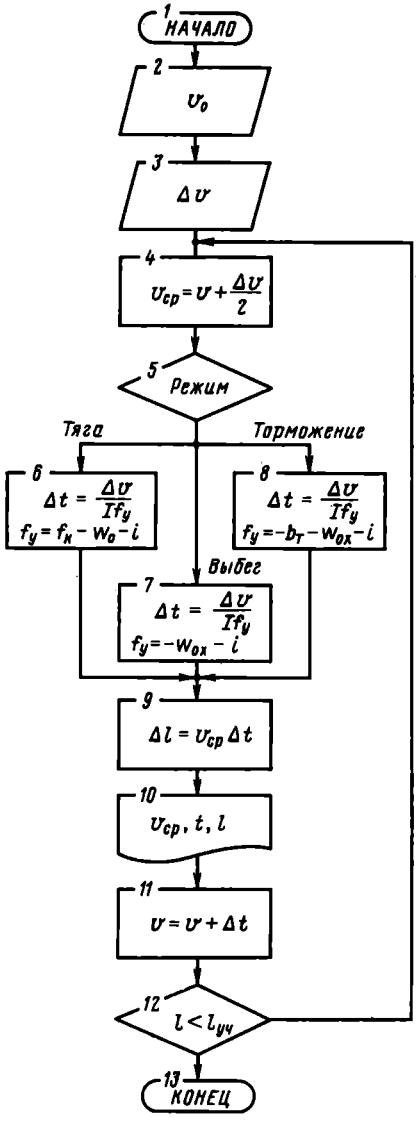


Рис. 4.14. Алгоритм имитационной игры «Управление движением поезда»

Укажем один из возможных вариантов программной реализации блока 5:

Например, нажатие клавиши с цифрой 1 будет означать переход (или продолжение) к режиму тяги, 2 — к режиму выбега, 3 — к режиму торможения. В режиме тяги по такому же принципу может быть реализовано управление движением локомотива на характеристиках полного или ослабленного возбуждения.

Рассмотрим работу схемы алгоритма (см. рис. 4.14):

в блоке *2* задается начальная скорость движения поезда;

в блоке *3* вводится приращение скорости относительно начальной;

в блоке *4* определяется средняя скорость движения, необходимая для определения значения удельного тягового усилия f_y на данном шаге численного интегрирования уравнения движения поезда;

в блоке *5* назначается режим движения поезда нажатием соответствующей клавиши клавишного устройства ЭВМ;

в одном из блоков *6* — *8* рассчитывается приращение времени решения дифференциального уравнения движения поезда;

в блоке *9* рассчитывается приращение пути;

в блоке *10* выводятся на экран (или печать) текущие значения v, t и l для возможности визуальной оценки состояния движения поезда;

в блоке *11* вводится приращение скорости движения для выполнения очередного шага интегрирования уравнения движения;

в блоке *12* производится проверка на окончание процесса.

```
1ØDIM A⊙
-----------------------------------------------------
1ØØKEYIN A⊙, 11Ø, 11Ø: GOTO 1ØØ
11Ø IF A⊙=«31» THEN 2ØØ
12ØIF A⊙=«32» THEN 3ØØ
13ØIF A⊙=«33» THEN 4ØØ
14ØPRINT «НАЖАТА КЛАВИША С НЕПРЕДУСМОТРЕННЫМ КО-
         ДОМ=»;
15ØHEXPRINT TAB (42);A⊙
16ØPRINT «ПОВТОРИТЕ НАЖАТИЕ КЛАВИШЫ»: GOTO 1ØØ
2ØØREM  РЕЖИМ ТЯГИ
.
.
GOTO 5ØØ
3ØØREM  РЕЖИМ ВЫБЕГА
.
.
GOTO 5ØØ
4ØØREM  РЕЖИМ ТОРМОЖЕНИЯ
```

В приведенном фрагменте программы оператором 1Ø задается длина символьной переменной, равная 1 байту.

В строке 1ØØ при помощи операторов KEYIN и GOTO реализуется режим ожидания ЭВМ до момента нажатия одной из клавиш.

В строке 11Ø анализируется значение символьной константы на равенство шестнадцатеричному числу 31, являющемуся кодом клавиши с цифрой 1. При истинности проверяемого условия управление передается оператору 2ØØ, начиная с которого, располагается блок программы интегрирования уравнения движения поезда в режиме тяги. В противном случае выполняется оператор с номером 12Ø.

Действия оператора 12Ø аналогичны предыдущему с той лишь разницей, что при истинности условия, записанного в операторе IF, обеспечивается интегрирование уравнения движения в режиме выбега.

Оператор 13Ø предназначен для перехода к режиму торможения.

Операторы 14Ø—16Ø выполняют функции защиты от неверно нажатой клавиши, выдают об этом сообщение оператору ЭВМ, указывают код ошибочно нажатой клавиши и организуют повторное (правильное) нажатие клавиши.

Отметим, что второй номер строки ($нс_2$=11Ø) в операторе KEYIN в данном примере не используется.

Контрольные вопросы

1. Дайте общую характеристику языка Бейсик. Предполагаемые области его применения.
2. С какими типами данных можно работать на Бейсике?
3. Дайте определение следующих понятий: программа, подпрограмма и помеченная подпрограмма, встроенная функция, функция пользователя. Поясните сходство и различие этих понятий.

4. Как организовать на Бейсике работу с комплексными числами при решении задач электроснабжения в системе переменного тока?

5. Что такое структурное программирование? Какие цели достигаются при его использовании. Является ли Бейсик структурированным языком?

6. Какие недостатки Бейсика при программировании сложных задач специальности?

7. Какими возможностями располагает Бейсик для решения задач управления устройствами электрифицированной железной дороги?

8. Перечислите основные приемы и стандартные средства Бейсика, используемые при отладке программ.

9. Как организовать на Бейсике диалог с ЭВМ?

10. Как записать программу и данные на магнитный диск и считать в оперативную память?

Глава 5

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ФОРТРАН

5.1. Общие сведения о языке Фортран

Фортран является одним из самых распространенных алгоритмических языков высокого уровня, ориентированных на решение научно-технических задач. Название языка происходит от сокращения английского словосочетания FORmulaTRANslator (переводчик формул). Язык был разработан в 1958 г. сотрудниками фирмы IBM и быстро получил широкое распространение.

В последующие годы язык развивался, сформировались различные версии Фортрана. К 1966 г. большинством пользователей ЭВМ была принята в качестве стандартной версия Фортран IV, дано ее официальное описание (эталон).

В СССР трансляторы с Фортрана IV разрабатывались для ЭВМ БЭСМ-6 и других отечественных машин второго поколения. Широко распространенные в организациях МПС машины серии ЕС также имеют в составе стандартного программного обеспечения трансляторы с Фортрана IV (кроме малых ЭВМ ЕС-1010 и ЕС-1011), причем по сравнению с эталонным языком возможности Фортрана ЕС ЭВМ несколько расширены.

Развитие Фортрана продолжается в сторону его обогащения, так что все программы, написанные на более ранних версиях, могут использоваться в последующих. К наиболее удачным можно отнести, например, Фортран-77 [25]; на больших машинах серии ЕС можно пользоваться Фортраном ОЕ. Существуют также сокращенные версии Фортрана, приспособленные к использованию персональных ЭВМ.

Трансляция программ с Фортрана осуществляется обычно с помощью компиляторов, которые входят в состав большинства операционных систем.

В организациях МПС принята операционная система ОС ЕС, поэтому при изложении материала данной главы будут рассматриваться возможности языка Фортран IV при использовании стандартного транслятора Фортран ST или оптимизирующего Фортран ОР в ОС. В некоторых вычислительных центрах ВЦ на ЕС ЭВМ используется также операционная система ДОС, транслятор которой позволяет работать с той же версией языка Фортран IV, что и в ОС с несущественными отличиями. В ДОС имеется также транслятор с более ранней версии языка — Базисного Фортрана, который в настоящее время практически не используется, хотя написанные на нем программы могут без изменения транслироваться и компиляторами с Фортрана IV

По сравнению с Бейсиком Фортран представляет гораздо бо́льшие вычислительные возможности при решении сложных задач. Недостатками его являются отсутствие языковых средств поддержки диалога, сложности в обработке текстовой и битовой информации.

Материал настоящей главы будет излагаться в предположении, что читатель знаком с программированием на языке Бейсик, поэтому похожие понятия Фортрана подробно поясняться не будут. В рамках данного учебника невозможно полностью описать особенности работы всех операторов Фортрана, поэтому он не может заменить специальных изданий [26, 27]. Ниже будет дано подробное объяснение наиболее употребительных элементов языка, для остальных будет даваться лишь краткое описание конструкции (синтаксиса) и назначение.

5.2. Основные элементы языка

Запись программы. Разработчики Фортрана ориентировались на кодирование исходных записей программ на 80-колонных перфокартах. Каждая карта позволяет, таким образом, закодировать 80 символов или одну строку текста. Колонки карты (позиции строки) занумерованы 1,2,...,80, а символы языка расположены в колонках по определенным правилам.

При подготовке программ на дисплеях эти правила сохраняются, так как одна строка экрана содержит 80 символов и соответствует одной карте.

На одной карте записывается не более одного оператора Фортрана или комментарий. Операторы встречаются двух родов: исполнимые и неисполнимые. Исполнимые при трансляции, порождают определенные команды в машинной программе, под воздействием которых во время решения задачи происходят арифметические действия, операции ввода/вывода и т. п.

Неисполнимые операторы команд не порождают, они только дают транслятору необходимую информацию для правильной компиляции.

Текст оператора Фортрана должен располагаться в пределах колонок 7—72 включительно. Комментарий отличается от оператора тем, что содержит символ C (от Comment) в 1-й колонке. Тогда в колонках 2—80 можно записывать любой текст, который будет распечатываться вместе с программой при работе транслятора, не оказывая влияния на работу самой программы.

Для того чтобы можно было ссылаться на оператор в других операторах, ему может быть присвоен номер или *метка*. В отличие от Бейсика метки операторов не определяют порядок их выполнения, т. е. присваиваются в произвольном порядке, кроме того, метка может вообще отсутствовать. Метки выбираются из диапазона целых чисел от 1 до 99999 и записываются в колонках 1—5. Если метка содер-

жит менее пяти цифр, она может располагаться по колонкам произвольно.

Колонка 6 нормально содержит пробел или нуль. Если она содержит любой другой символ, то текст, содержащийся в колонках 7—72 этой карты, рассматривается как продолжение текста оператора предыдущей карты сразу вслед за ее 72-й колонкой. Допускается до 19 строк продолжения, т. е. один оператор Фортрана не может занимать более 20 карт (строк). Метка, если она нужна, ставится только перед текстом первой строки.

Колонки 72—80 могут содержать любые символы, они не воспринимаются транслятором, а только выводятся на распечатку. В эти колонки заносят обычно нумерацию карт по порядку, а также 3—4 символа, идентифицирующих программу. Для небольших программ делать эту работу не имеет смысла.

Подготовку программ удобно выполнять на специально разграфленных бланках, где выделены поля 1—5, 7—72 и 73—80 (рис. 5.1), что позволяет безошибочно размещать текст программ.

Символы Фортрана. При записи операторов используются следующие символы:

26 заглавных (прописных) букв латинского алфавита A — Z, а также символ денежной единицы ⊙, приравненный по своим функциям к букве. Заметим, что буквы русского алфавита, одинаковые по начертанию с латинскими — А, В, С, Е, Н, К, М, О, Р, Т, Х (не У, отличающееся от Y!), кодируются на картах и в дисплеях одинаково и поэтому воспринимаются как латинские. Прочие русские буквы можно использовать только в комментариях и символьных данных;

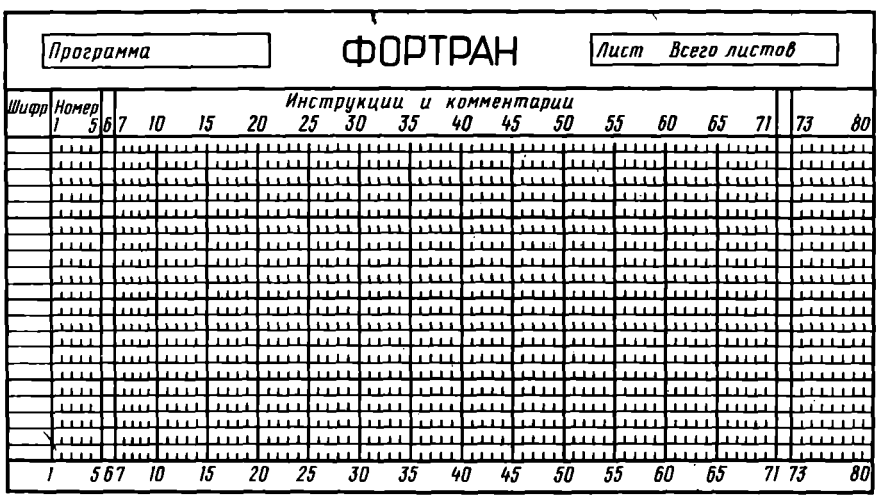


Рис. 5.1. Бланк для программирования на Фортране

арабские цифры 0,1,...,9;

знаки и другие специальные символы: пробел, = равно, + плюс, — минус, * звездочка, ' запятая, точка, ' апостроф, &, коммерческое «И» (амперсанд). Эти символы могут иметь различное назначение. «Пробел», за исключением отдельных случаев, когда он будет обозначаться знаком ␣, не имеет самостоятельного значения, а используется для удобства чтения программы;

ключевые слова, используемые для обозначения большинства операторов:

ASSIGN...TO — назначить...для; BACKSPACE — шаг назад;
BLOCK DATA — блок данных; CALL — вызвать;
COMMON — общий; COMPLEX — комплексный; CONTINUE — продолжать; DATA — данные;
DEFINE FILE — определить файл; DIMENSION — размеры;
DO — выполнять; DOUBLE PRECISION — двойная точность;
END — конец; END FILE — конец файла;
ENTRY — вход; EQUIVALENCE — эквивалентность;
EXTERNAL — внешнее (имя); FIND — искать;
FORMAT — формат; FUNCTION — функция;
GO TO — идти к; IF — если;
IMPLICIT — неявное; INTEGER — целый;
LOGICAL — логический; NAMELIST — список имен;
PAUSE — пауза; PRINT — печатать;
PUNCH — перфорировать; READ — читать;
REAL — вещественный; RETURN — возврат;
REWIND — перемотка; STOP — стоп;
SUBROUTINE — подпрограмма; WRITE — писать;

знаки операций над данными, которые составляются из специальных символов и букв (будут приведены ниже).

Данные. Так же, как и в Бейсике, в Фортране используются константы, переменные и массивы. Эти данные различаются по типу в зависимости от способа представления их в памяти ЭВМ. В Фортране IV используются данные следующих типов: целые, вещественные, удвоенной точности, комплексные, логические, текстовые и шестнадцатеричные. В более поздних версиях используются также данные повышенной точности и битовые данные.

Константы. Тип константы и ее значение определяются записью константы.

Целые константы (типа INTEGER) — это любое число (со знаком или без него), записанное без десятичной точки. В отличие от Бейсика никаких дополнительных символов (%) в записи целых констант Фортрана нет. Примеры записи целых констант:

Правильно	Неправильно
78	78. (точка недопустима)
0	0.0 (точка недопустима)
—1479	—1,479 (запятая недопустима)
+25	+25 (пробел недопустим)

По абсолютному значению целая константа не должна превышать 2 147 483 674 (т. е. $2^{31} - 1$), поскольку она занимает в памяти 4 байта (1 слово).

Упражнение 1. Какие из следующих целых констант записаны правильно: а) —2147493640; б) +601534892; в) 9,575; г) 2.000.000; д) 6315; е) —1?

Вещественные (действительные) константы (типа REAL) — основной вид констант, используемых в вычислениях. Различают (как и в Бейсике) запись в естественной форме (с фиксированной точкой) и в экспоненциальной форме (с порядком, отделенным от мантиссы символом E). В отличие от Бейсика мантисса может быть ненормализованной, т. е. любым числом в естественной форме, а также целым (без точки). Кроме того, положительный порядок можно записать без знака.

Примеры записи вещественных констант:

В естественной форме	В экспоненциальной форме
0.0	.0E0
1987.	+1.987E3
8.3	.00083E4
—487.35	—48735E—3

Константа 1987 не воспринимается как целая, так как содержит десятичную точку.

Точность представления вещественной константы в ЭВМ — приблизительно 7 десятичных значащих цифр. Например, константы 134512349E—11 и 134512352E—11 могут оказаться представленными в памяти совершенно одинаково. Абсолютные значения чисел допускаются в пределах примерно от 5.4E—79 до 7.2E75, а также в виде 0. Одна константа вещественного типа занимает в памяти 4 байта (1 слово).

Упражнение 2. Какие из следующих вещественных констант записаны правильно: а) 3.14; б) —.387; в) —387E—3.; г) 0; д) —135783E71; е) 0.000001; ж) 125; з) 1074452+70; и) —1074452E—81?

Комплексная константа типа COMPLEX представляет собой упорядоченную пару вещественных констант в любой форме записи (заключенную в скобки и разделенную запятой). Первая из констант соответствует действительной части комплексного числа, а вторая — коэффициенту при мнимой единице.

Например:

Математическая запись	На Фортране
$7{,}43+j8{,}1$	(7.43,8.1)
$-0{,}0001-j0{,}00357$	(—1E—4,—357E—5)
$25000+j247$	(25E3,247.)

В памяти ЭВМ комплексная константа размещается в двух последовательных словах по 4 байта, диапазон и точность представления действительной и мнимой частей соответствуют вещественным константам.

Вещественные константы удвоенной точности типа DOUBLE PRECISON или REAL*8 отличаются от вещественных тем, что они записываются только в экспоненциальной форме, а вместо символа E в записи порядка используется символ D. В памяти ЭВМ такая константа занимает двойное слово (8 байт), поэтому точность представления числа — приблизительно 17 десятичных значащих цифр. Например, приведенные ранее константы, будучи записаны 134512349D — — 11 и 134512352D — 11, в машине получат разное представление.

Диапазон констант удвоенной точности — такой же, как и у обычных вещественных констант. В задачах, решаемых в области электрифицированных железных дорог, чаще всего достаточно точности четырехбайтовых чисел. Использования данных типа REAL*8 лучше избегать, так как они занимают вдвое больше места в памяти ЭВМ, и операции над ними осуществляются медленнее.

Комплексные константы удвоенной точности типа COMPLEX*16 составляются подобно обычным комплексным константам из двух вещественных чисел, каждое их которых является константой удвоенной точности. Например, выражение — 0,0001 — j 0,00357 можно записать в виде — 1D — 4,— 357D — 5. В памяти такая константа занимает два последовательных двойных слова, т. е. 16 байт.

Логические константы типа LOGICAL записываются в виде .TRUE. (истина) или .FALSE. (ложь). Логическая константа занимает в памяти 4 байта (1 слово).

Текстовые константы не имеют специального обозначения типа. Они представляют собой последовательность любых символов, имеющихся на клавиатуре внешних устройств ЭВМ, причем пробел является значащим символом в константе. Используются две формы записи:

в виде последовательности символов $nHs_1s_2...s_n$, где n — целая константа без знака ($n \leqslant 255$), указывающая число символов константы, следующих за определителем константы H;

в виде последовательности символов $'s_1s_2...s'_n$ в апострофах (в отличие от двойных кавычек в Бейсике).

Если нужно представить в константе сам апостроф, он удваивается, например:

```
26H ПРИМЕР␣ТЕКСТОВОЙ␣КОНСТАНТЫ
'ЭТО␣ТОЖЕ␣ТЕКСТ.␣КОНСТ.!'
'А␣ВОТ␣''— АПОСТРОФ'
```

Последняя константа фактически составляет текст:

```
А␣ВОТ␣'— АПОСТРОФ
```

Каждый символ текстовой константы занимает 1 байт в памяти ЭВМ.

Шестнадцатеричные константы записываются в виде последовательности шестнадцатеричных цифр, перед которыми ставится буква Z.

Две цифры занимают в памяти 1 байт. Если в константе записано нечетное число шестнадцатеричных цифр, слева добавляется 0. Длина константы зависит от типа переменной, которой она присваивается. Если в константе больше пар цифр, чем байтов в переменной, крайние слева цифры константы не заносятся. Если длина переменной больше длины константы, последняя дополняется слева нулями. Примеры шестнадцатеричных констант:

Константа	Двоичное представление в памяти
Z3F	00111111
ZABCD	1010101111001101

Переменные. Как указывалось ранее, в алгоритмических языках для обозначения переменных используются символические имена (идентификаторы). В Фортране в отличие от Бейсика имя может содержать до шести символов — либо букв, либо цифр, причем первым символом обязательно должна быть буква. Очевидно, число имен, которые могут быть использованы в программах, практически не ограничено.

При написании программ имена переменных выбираются произвольно, но обычно используются имена, сходные с математическим обозначением или с названием физических величин.

В зависимости от способа представления переменных в памяти ЭВМ и их использования различают целые, вещественные, комплексные, двойной точности и логические переменные. Тип переменной обязательно должен быть описан в программе. Существуют два способа описания типа переменной.

Явное описание типа требует от программиста до первого исполнимого оператора записи одного или нескольких операторов явной спецификации типа. Эти операторы имеют следующий синтаксис (конструкцию):

ТИП *a*, *b*, *c*,...

где ТИП — ключевое слово, указывающее тип переменных; *a*, *b*, *c*,... — имена переменных, разделенных запятыми (список имен).

Ключевые слова, используемые для описания типа, приведены при описании типов констант. Используются также описатели: REAL*4, который эквивалентен просто REAL; INTEGER*4, эквивалентный просто INTEGER.

Например, если в начале программы записать

```
INTEGER X4, AB, UI, K12
REAL JA, MIX, IFIKT, UFIKT, NP
REAL*8 DELTA, Z6
COMPLEX Y, W
```

транслятор отведет переменным X4, AB, UI, K12 по 4 байта в памяти, а при арифметических действиях будет рассматривать их как целые, переменным JA, MIX, IFIKT, UFIKT, NP отведет также 4 байта, но будет рассматривать их, как вещественные; переменным DELTA, Z6 — с двойной точностью — по 8 байтов; переменным Y, W — по 8 байтов комплексного типа.

В отличие от Бейсика каждое имя в программе должно быть уникальным, т. е. нельзя обозначать одним и тем же именем переменные разных типов.

Помимо типов переменных, совпадающих с типами констант, описанными выше, в Фортране используются переменные нестандартной длины: целые INTEGER*2 (2 байта) и логические LOGICAL*1 (1 байт), пользоваться которыми начинающему программисту не рекомендуется.

Символьные и шестнадцатеричные переменные не имеют специальной спецификации. Значение символьной или шестнадцатеричной константы может быть присвоено переменной любого типа (целой, вещественной и т. д.) по определенным правилам, которые будут рассмотрены далее.

Неявное описание типа или описание по умолчанию наиболее удобно и широко используется. Этот способ предполагает, что все имена, начинающиеся с букв I, J, K, L, M или N, относятся к

переменным целого типа, а начинающиеся с букв A, B...H, O, P...Z или ⊙ — к вещественным. Если в программе какие-либо имена описаны явно, на них это соглашение не распространяется.

В приведенном выше примере явного описания имена K12 и UFIKT в списках являются лишними, так как этим переменным транслятор присвоит целый и вещественный типы соответственно при первом же упоминании их имен в программе.

В тех случаях, когда обозначение физической величины (вещественной) начинается с букв I — N, многие программисты добавляют впереди букву A, или X, или другую, чтобы переменная была отнесена к вещественному типу, например, длину l можно обозначить XL, ток нагрузки $I_н$ AIH и т. п.

Переменные с двойной точностью, комплексные и логические, а также переменные нестандартной длины по умолчанию не описываются. Для уменьшения работы по описанию таких переменных программист может использовать неисполнимый оператор IMPLICIT, устанавливающий правила неявного описания переменных в пределах одной программы. Например, если написать IMPLICIT COMPLEX (U, X — Z), LOGICAL (B, D), то все переменные, начинающиеся с букв U, X, Y, Z, будут считаться комплексного типа, а переменные, начинающиеся с букв B, D,— логического.

Оператором IMPLICIT можно также изменить стандартное описание по умолчанию переменных целого и вещественного типов. В нем нельзя применять описатель типа DOUBLE PRECISION, а только REAL*8.

Упражнение 3. Какие из следующих имен относятся к переменным целого типа, какие — к вещественным, а какие вообще недопустимы: а) HET86; IBM360; в) EC1033; г) UKOPOTK; д) IVAN5; е) POTERI; ж) X9Ю7; з) NEKTAR; и) MF-100; к) ALFA; л) INDEKS?

Массивы. Как и в Бейсике, в Фортране используются массивы, т. е. совокупности переменных с индексами. Счет индексов начинается с единицы, но в отличие от Бейсика число индексов может достигать семи (семимерные массивы).

Элементы массивов записываются следующим образом:

ИМЯ (i_1 [,i_2,i_3...,i_7])

где ИМЯ — имя массива; i_1,i_2..., i_7— индексные выражения, в качестве которых обычно используются целые константы, целые переменные, а также арифметические выражения, дающие результат целого типа. Примеры обращения к элементам массива:

U (5) — пятый элемент массива U;

TOK(2,7) — элемент матрицы TOK, расположенный во 2-й строке, 7-м столбце;

K23(I,2*K+4) — элемент матрицы K23 в i-й строке, (2 K+4)-м столбце (например, если I=3, K=2, это эквивалентно записи K23 (3,8).

Как и простые переменные, массивы могут быть целого,

вещественного и других типов и определяться явно или неявно (по первой букве имени). В отличие от Бейсика, имена массивов не должны совпадать ни с одним именем простых переменных.

В программе обязательно должны быть описаны размерность (число измерений) и размер (верхняя граница индексов) всех массивов. Для этой цели служит специальный неисполнимый оператор DIMENSION, который записывается в начале программы до первого исполнимого оператора. Например, фрагмент

```
DIMENSION U(12), TOK(3,1∅), K13(5,2∅), Z(7) COMPLEX Z
```

описывает: вещественные массивы U (12 элементов) и TOK (матрицу 3×10); массив целого типа K13 (матрицу 5×20); массив Z комплексного типа (семь элементов).

Для описания размеров и размерностей массивов можно пользоваться явными описателями типа. Например, данное выше описание эквивалентно следующему:

```
REAL U(12), TOK(3,1∅)
INTEGER K13(5,2∅)
COMPLEX Z(7)
```

В памяти ЭВМ (в отличие от Бейсика) элементы массивов располагаются по столбцам. При этом один из описанных выше массивов расположится так: TOK (1,1), TOK(2,1), TOK(3,1), TOK(1,2), ..., TOK(2,10) TOK(3,10). Вообще, если индексов несколько, «быстрее» всех меняется первый индекс, затем второй и т. д., «медленнее» всех — последний. Например, массив REAL A(3,2,5) расположится в памяти следующим образом: A(1,1,1), A(2,1,1), A(3,1,1), A(1,2,1), A(2,2,1), A(3,2,1), A(1,1,2) и т. д. Знать расположение массива в памяти необходимо при программировании некоторых операций ввода-вывода.

5.3. Операции над данными

Процесс решения задачи — это есть получение результата путем преобразования исходных данных по известному алгоритму. В простейшем случае алгоритм определяется знаками операций, которые нужно совершить над данными. Конструкция алгоритмического языка, содержащая данные, связанные знаками операций, называется выражением. В Фортране различают два вида выражений — арифметическое и логическое.

Арифметическое выражение. Арифметические данные (операнды), т. е. данные целого, вещественного и комплексного типов, объединенные знаками операций и символами функций, образуют арифметическое выражение. Знаки большинства операций в Фортране сходны с Бейсиком: + сложение,— вычитание, $\times$ умножение, / деление. В отличие от Бейсика возведение в степень обозначается двумя звездочками **.

Субординация операций такая же, как в Бейсике и в математике (с учетом скобок). Операции одного ранга выполняются слева направо, за исключением возведения в степень, которое выполняется справа налево. Синтаксис обращения к встроенным функциям также совпадает с Бейсиком. Примеры записи математических выражений на Фортране:

Математическая запись	На Фортране
$2\cos(x-\pi/6)$	2.*COS(X—.525599)
I_j^2R	I(J)**2*R
$e^{\lvert x-y\rvert}+\lvert x-y\rvert^{x+y}$	EXP(ABS(X—Y))+ ABS(X—Y)**(X+Y)
$y^{\sin^2 x}$	Y**SIN(X)**2
$(y^{\sin x})^2$	(Y**SIN(X))**2

Следует обратить внимание на следующие особенности записи арифметических выражений на Фортране:

константа π в ЭВМ не хранится;

если в выражении переменные — вещественные, константы лучше употреблять с точкой;

если показатель степени — целый, его лучше записывать без точки, так как в этом случае транслятор выполняет многократное умножение, а при записи с точкой — вычисляет выражение с использованием логарифмов, что дольше по времени и требует положительного аргумента;

при делении целых чисел берется целая часть частного, никаких округлений не делается. Так, результат вычисления выражения I/M при I=99 и M=100 будет равен нулю;

не могут стоять рядом два знака операции, например, нельзя писать A*—B, надо A*(—B).

Вообще в эталоне языка Фортран IV не разрешается смешивать переменные и константы разных типов. Трансляторы ДОС и ОС ЕС такое смешение допускают, однако пользоваться этим не рекомендуется, так как машинная программа удлиняется и работа ее замедляется из-за перевода чисел из одной формы в другую.

В табл. 5.1 приведены стандартные функции Фортрана, аргументами которых являются данные типа REAL*4 или INTEGER*4. Если функция предусмотрена и для аргументов типа COMPLEX*8, ее имя или приставка C указывается в квадратных скобках. Аргументы целого типа обозначены N, вещественного или комплексного — X.

Тип результата, получаемого в результате обращения к функции, определяется первой буквой ее имени подобно типам переменных.

Для функций комплексного аргумента результат комплексный, за исключением фукнции CABS (X), которая дает вещественный результат.

Таблица 5.1

Обращение к функции	Математическое описание	Обращение к функции	Математическое описание
ALOG(X) [CLOG(X)]	$y=\ln x$	MOD(N1, N2) AMOD(X1, X2)	Остаток от деления $\frac{n_1}{n_2}$ (или $\frac{x_1}{x_2}$)
ALOG10(X)	$y=\lg x$		
[C]EXP(X)	$y=e^x$	FLOAT(N)	Преобразование целого в вещественное
[C]SQRT(X)	$y=\sqrt{x}$		
ARSIN(X)	$y=\arcsin x$	IFIN (X)	Преобразование вещественного в целое
ARCOS(X)	$y=\arccos x$		
ATAN(X)	$y=\operatorname{arctg} x$	ISIGN(N1, N2) SIGN(X1, X2)	Присвоение знака $y=$(знак n_2)$\cdot\lvert n_1\rvert$ или (знак x_2)$\cdot\lvert x_1\rvert$
ATAN2(X1, X1)	$y=\operatorname{arctg}\frac{x_1}{x_2}$		
[C]SIN(X)	$y=\sin x$	IDIM(N1, N2) DIM(X1, X2)	Положительная разность $y=n_1-\min(n_1, n_2)$ или $x_1-\min(x_1, x_2)$
[C]COS(X)	$y=\cos x$		
TAN(X)	$y=\operatorname{tg} x$		
COTAN(X)	$y=\operatorname{ctg} x$	REAL(X)	Вещественная часть комплексного числа
SINH(X)	$y=\operatorname{sh} x=\frac{e^x-e^{-x}}{2}$	AIMAG (X)	Мнимая часть комплексного числа
COSH(X)	$y=\operatorname{ch} x=\frac{e^x+e^{-x}}{2}$	CMPLX(X1, X2)	Получение комплексного числа $y=x_1+jx_2$
TANH(X)	$y=\operatorname{th} x=\frac{e^x-e^{-x}}{e^x-e^{-x}}$	CONJG(X)	Получение сопряженного комплексного числа $y=x_1-jx_2$
	Функции ошибки		
ERF(X)	$y=\operatorname{erf}(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-u^2}du$		
ERFC(X)	$y=1-\operatorname{erf}(x)$	MAX0(N1, N2, ..., NM) AMAX0(N1, N2...NM)	Выбор максимального из m значений Аргументы целые
	Гамма-функция		
GAMMA(X)	$y=\Gamma(x)=\int_0^\infty u^{x-1}e^{-u}du$		
ALGAMA(X)	$y=\ln\Gamma(x)$	MAX1(X1, X2...XM) AMAX1(X1, X2...XM)	Аргументы вещественные
IABS(N)	$y=\lvert n\rvert$ — для целых		
ABS(X)	$y=\lvert x\rvert$ — для вещественных		Выбор минимального из
CABS(Z)	$y=\lvert z\rvert=\sqrt{a^2+b^2}$, где $z=a+jb$	MIN0(N1, N2 ...NM) AMIN0(N1, N2...NM)	m значений Аргументы целые
INT(X) AINT(X)	$y=$(знак x)$\cdot m$, где m — наибольшее целое $\leqslant\lvert x\rvert$	MIN1 (X1, X2 ... XM) AMIN1 (X1, X2 XM)	Аргументы вещественные

Логическое выражение. В логических выражениях операнды логического типа, а также отношения связаны между собой символами логических операций. Результатом вычисления логического выражения могут быть два логических значения: .TRUE. («истина») и .FALSE. («ложь»).

Логические операции могут быть записаны в следующем виде:

На языке Фортран	Математическое обозначение	Название операции
.NOT.	$\neg$	НЕ (отрицание, инверсия)
.AND.	$\wedge$	И (логическое умножение, конъюнкция)
.OR.	$\vee$	ИЛИ (логическое сложение, дизъюнкция)

Две логические операции могут быть записаны подряд только в том случае, если второй из них является операция .NOT., например .AND..NOT. и .OR..NOT.

Результат логической операции над двумя операндами определяется известными правилами алгебры логики. В отличие от аналогичных операций над отдельными битами в Бейсике в Фортране операции выполняются над константами, переменными или массивами логического типа, которые занимают 4 или 1 байт в памяти и имеют два упомянутых выше логических значения.

В качестве одного из обоих операндов логической операции могут быть так называемые отношения. Отношение представляет собой два арифметических выражения, дающих вещественный или целый результат, соединенных одним из следующих символов отношений:

На языке Фортран	Математические обозначения	Название отношения
.LT.	$<$	Меньше (less than)
.LE.	$\leqslant$	Меньше или равно (less than or equal to)
.GT.	$>$	Больше (greater than)
.GE.	$\geqslant$	Больше или равно (greater than or equal to)
.EQ.	$=$	Равно (equal to)
.NE.	$\neq$	Не равно (not equal to)

Результат вычисления отношений всегда логического типа длиной 4 байта. Порядок вычисления логического выражения (если он не определен скобками) следующий: вычисление арифметических выражений; вычисление отношений; выполнение операций .NOT. .AND. .OR.

Пример логического выражения:

A**2—B**2.LT.C**2—D**2.AND.A.NE.D.

Пусть вещественные переменные в этом выражении имеют значения A=2., B=4., C=3. Тогда при D=4. значением логического выражения будет .TRUE.; при D=5.—.FALSE. (из-за ложности первого отношения $a^2-b^2<c^2-d^2$), а при D=2.— то же .FALSE. (из-за ложности отношения $a \neq d$).

Если в логической операции участвуют логические данные разной длины (1 и 4 байта), результат получается длиной 4 байта.

Операции над текстовыми данными. Возможности Фортрана IV в части операций над текстовыми данными значительно скромнее, чем в Бейсике. Так, например, если некоторым переменным A и B одинакового арифметического типа присвоено текстовое значение, можно выполнить над ними только следующие операции отношения:

A.EQ.B или A.NE.B

Никакие числовые эквиваленты символьных данных в Фортране не установлены, поэтому результаты других операций отношения непредсказуемы. Функции для преобразования текстовых данных также не предусмотрены. Разрешается присваивание одним переменным значений других, которым ранее было присвоено

текстовое значение. Например, если A и B уже содержат текстовые значения, допустим 'ТЕСТ' и 'СТОП', то в результате выполнения операторов C=A; D=B; E(5)=C; переменная C и 5-й элемент массива E также будут содержать текст 'ТЕСТ', а переменная D—'СТОП' При этом необходимо строго следить, чтобы переменные по обе стороны знака присваивания (=) были одного типа, иначе транслятор вместо команды пересылки кодов составит программу для перевода данных.

5.4. Операторы Фортрана

Программа на Фортране, как отмечалось, состоит из последовательности операторов, которые могут быть исполнимыми и неисполнимыми. Примером неисполнимых операторов являются операторы явного описания типа переменных.

Исполнимые операторы можно подразделить на три группы: операторы присваивания, операторы управления и операторы ввода-вывода.

Неисполнимые операторы описания типа рассмотрены выше. Большинство остальных неисполнимых операторов используют совместно с операторами ввода-вывода и будут описаны вместе с ними. Другие операторы описания свойств данных и подпрограмм рассматриваются далее в соответствующих разделах.

Операторы присваивания. К ним относятся арифметический и логический операторы присваивания, а также оператор ASSIGN...TO, который используется совместно с оператором управления GO TO и будет рассмотрен ниже.

Арифметический и логический операторы присваивания — единственные, не начинающиеся с ключевого слова — имеют одинаковый синтаксис: $a=b$, где a — переменная или элемент массива целого, вещественного или комплексного типа в арифметическом (или логическом) операторе присваивания; b — арифметическое (логическое) выражение.

Примеры арифметических операторов присваивания:

Математическая запись	На языке Фортран
$I=\frac{U}{R}$	TOK=U/R
$z=\sqrt{r^2+x^2}$	Z=(R**2+X**2)** **Ø.5 или Z=SQRT(R*R+X*X)

Вторая запись дает более короткую и быстродействующую программу. Если результат арифметического выражения b не совпадает по типу с переменной (элементом массива) a, он преобразуется следующим образом:

если a — целое, b — вещественное — присваивается целая часть;

если a— типа REAL*4, b— типа REAL*8— берется старшая часть мантиссы b;

если a— комплексное, b — вещественное или целое, то оно присваивается действительной части a (мнимая часть принимается равной 0);

если a — целое или вещественное, b — комплексное, то присваивается действительная часть b (мнимая не используется).

Начинающим программистам лучше избегать смешивания типов данных в арифметических выражениях и операторах присваивания.

Нельзя в одном операторе присваивать значение арифметического выражения одновременно нескольким переменным.

Логический оператор присваивания отличается тем, что переменная a должна быть только логического типа и никакие преобразования результата вычисления логического выражения b не делаются. Напомним, что логические данные могут принимать только значения .TRUE. и .FALSE. Нелишне напомнить также, что в алгоритмических языках оператор присваивания неэквивалентен обычному математическому уравнению. Знак «=» надо читать «становится равным» или «заменяется на...». Например, на Фортране можно записать J=J+2, тогда как уравнение j=j+2 бессмысленно. Приведенный оператор означает следующую инструкцию для ЭВМ: «взять число из области памяти, обозначенной J, прибавить к нему 2 и результат записать в область J, т. е. текущее значение J увеличить на 2.

Упражнение 4. Ниже приведен текст некоторой программы. Какие операторы составлены неверно и в чем состоят ошибки?

```
COMPLEX Z(2)
LOGICAL D,E,F
Z(1)=(3.,—2.7)
Z(2)=5.
D=Z(1).GT.Z(2)
A=7
B=—1Ø.
C=24.*D+A*—B
E=A**2—B**2.LE.(A—B)**2
F=E.AND.ABS(A).NE.C
```

Упражнение 5. Переменные имеют следующие значения: A=2.75; B=4.; C=.5; E=1Ø; H=—2.; K=2; M=—4. Какие значения получат элементы 1—5 массива X в результате работы следующего фрагмента программы?

```
X(1)=(A*B—E/C)**K
X(2)=(B—A)/C*H+E
X(3)=(A*B*C*H+E)**K**2
X(4)=25/M*K
X(5)=25/(M*K)
```

Упражнение 6. Однопутный участок электрифицированной железной дороги длиной *l* питается с двух сторон от подстанций *A* и *B* с одинаковым напряжением на шинах. На участке расположены три поезда на расстояниях *X1, X2, X3* от подстанции *A*, потребляющие токи *I1, I2, I3* соответственно. Сопротивление 1 км контактной сети *R* Ом. Составьте программу вычисления токов подстанций ТОКА, ТОКВ и потерь напряжения до поездов *U1, U2, U3*, считая, что *L, X1, X2, X3, I1, I2* и *I3* известны. Имена переменных оставьте такими, как они записаны в условии.

Операторы управления. Исполнимые операторы Фортран-программы выполняются в пордке их написания (в отличие от Бейсика, где этот порядок определяется нумерацией строк)

Для изменения порядка исполнения операторов, т. е. для реализации управляющих структур, в Фортране используется ряд специальных операторов, которые называют операторами управления.

Операторы GO TO. Используют три вида операторов GO TO (идти к), которые позволяют передать управление любому исполнимому оператору программы, имеющему метку.

Безусловный оператор GO TO имеет вид:

GO TOm

Здесь m— метка другого исполнимого оператора, который должен выполняться вслед за GO TO.

Этот оператор Фортрана одинаков с GO TO... Бейсика. Название его можно писать слитно: GOTO.

Очевидно, исполнимый оператор, записанный в программе вслед за GO TO, обязательно должен иметь метку, так как иначе он никогда не сможет получить управление.

Вычисляемый оператор GO TO имеет конструкцию:

GO TO (m_1, m_2, ..., m_n), i

где m_1, m_2, m_n— метки некоторых исполнимых операторов; i— целая переменная.

Этот оператор передает управление на метки m_1, m_2, ..., m_n, если к моменту его исполнения i равна соответственно 1,2..., n. Если $i<1$ или $i>n$, выполняется оператор, записанный непосредственно за GO TO. Вычисляемый GO TO по своим функциям аналогичен оператору ON...GO TO... Бейсика и предназначен для реализации структуры «выбор».

Приведенная в п. 4.3 программа расчета эффективного тока обмотки трансформатора на Фортране запишется так:

```
C РАСЧЕТ ЭФФЕКТИВНОГО ТОКА ОБМОТКИ ТРАНСФОРМАТОРА
REAL I
<операторы ввода J, S1, S2, E1, E2>
GO TO (4Ø, 6Ø, 8Ø), J
4Ø I=SQRT(4.*E1**2+E2**2+2.*S1*S2)/3.
GO TO 9Ø
6Ø I=SQRT (E1**2+4.*E2**2+2.*S1*S2)/3.
GO TO 9Ø
8Ø I=SQRT(E1**2+E2**2-S1*S2)/3.
9Ø <операторы печати результата>
END
```

В программе сохранены имена и метки, равные номерам строк программы на Бейсике.

Если ошибочно будет задано $J<1$ или $J>3$, будет выполняться оператор с меткой 4Ø, так как он следует непосредственно за вычисляемым GO TO. Можно усовершенствовать этот фрагмент следующим образом:

```
GO TO (4Ø, 6Ø, 8Ø),J
< операторы, обеспечивающие печать сообщения «НЕВЕРНО ЗАДАН НОМЕР
ОБМОТКИ J=....»>
STOP
4Ø I=SQRT(4.*E1**2...<...и т. д. по прежнему тексту>
```

Операторы STOP и END, примененные в программе, описаны ниже. Они, в частности, прекращают работу программы.

Назначенный оператор GO TO и оператор ASSIGN...TO используются в программах совместно. Первым по ходу выполнения программы должен стоять оператор

ASSIGN m TOi

где m — метка исполнимого оператора, i — целая переменная. Под действием этого оператора метка m запоминается в области памяти, обозначенной i. Никаких арифметических действий с переменной i в этом случае делать нельзя.

Если после этого встретится оператор

GO TO i, (m_1, m_2...m_n)

то выполняется действие, аналогичное безусловному переходу по оператору GO TO m. Метка m, употребляемая в операторе ASSIGN, должна быть одной из перечисленных в списке m_1, m_2, ..., m_n. После назначенного GO TO должен следовать оператор с меткой. Назначение метки m, не перечисленной в списке оператора GO TO, приводит к ошибке. Рассмотренная конструкция также может применяться при программировании структуры «выбор».

Операторы условного перехода IF. Подобно Бейсику эти операторы позволяют программировать структуры «если-то-иначе». В Фортране IV используются два вида операторов IF (если).

Арифметический оператор IF имеет конструкцию

IF(a) m_1, m_2, m_3

где a — арифметическое выражение целого или вещественного типа; m_1, m_2, m_3 — метки исполнимых операторов. Следующий за арифметическим IF оператор должен иметь метку.

Выполнение этого оператора осуществляется в такой последовательности: вычисляется значение выражения a; полученное значение анализируется — если $a<0$, переход осуществляется на метку m_1, если $a=0$ — на метку m_2, если $a>0$ — на метку m_3.

Программа на Бейсике для расчета зависимости коэффициента сцепления колеса с рельсом от скорости, приведенная в п. 4.3, на Фортране может выглядеть так:

```
C РАСЧЕТ КОЭФФИЦИЕНТА СЦЕПЛЕНИЯ КОЛЕСА С РЕЛЬСОМ
      REAL K
      V=0.
   30 IF(V-40.)60, 60, 40
   40 K=.09+95./(413.+3.*V)
      GO TO 70
   60 K=.228+7./(53+3.*V)
   70 <операторы печати V и K>
      V=V+5.
      IF (V-100.)30, 30, 100
  100 STOP
      END
```

Операторы STOP и END поясняются далее.

Логический оператор IF имеет конструкцию

IF(l)S

где l — логическое выражение;

S — любой исполнимый оператор Фортрана, кроме DO и другого IF.

Последовательность выполнения оператора: вычисляется значение логического выражения l; если l=.TRUE., выполняется оператор S, если l=.FALSE., оператор S пропускается.

С применением логического IF рассмотренные в описании арифметического IF фрагменты программы расчета коэффициента сцепления можно записать так:

```
3∅  IF (V.LE.4∅.)GO TO 6∅
    K=.∅9+95./413.+3.*V)
    ...
    IF(V.LE.1∅∅.)GO TO 3∅
    END
```

Ту же задачу можно решить и с помощью программы:

```
C РАСЧЕТ КОЭФФИЦИЕНТА СЦЕПЛЕНИЯ (ВАРИАНТ 2)
    REAL K
    V=∅.
4∅  K=.∅9+95./(413.+3.*V)
    IF(V.LE.4∅)K=.228+7./(53.+3.*V)
    < оператор печати V и K>
    V=V+5.
    IF(V.LE.1∅∅.)GO TO 4∅
    END
```

С применением логического IF программа становится более наглядной, но, как правило, менее эффективной, чем с арифметическим IF. Во втором из рассмотренных вариантов под меткой 4∅ всегда выполняется вычисление K, а если $V \leqslant 40$ км/ч, это значение потом «затирается» значением, вычисленным по формуле, записанной в операторе IF.

Как видно, если в качестве логического выражения l оператора IF Фортрана используется отношение, а оператором S является безусловный GO TO, он почти совпадает с оператором IF Бейсика.

Упражнение 7. Имеется нелинейный резистор, вольт-амперная характеристика которого описывается уравнением $u=Ai^2$. Известен ток, протекающий по нему (направление тока определяется знаком переменной TOK). Написать фрагмент программы для определения падения напряжения U (со своим знаком): а) используя арифметический IF; б) используя логический IF; в) используя стандартную функцию Фортрана.

Упражнение 8. К цепочке, содержащей последовательно включенные диод и резистор R, приложено напряжение U (положительным считается прямое направление для диода). Написать фрагмент программы для вычисления тока в цепи: а) с использованием логического IF; б) со стандартной функцией.

Упражнение 9. Импульсное напряжение $u(t)$ изменяется с периодом 100 мс по следующему закону: 1) от 0 до 20 мс $u=0$; 2) от 20 до 30 мс $u=50$ В; 3) от 30 до 40 мс $u=0$; 4) от 60 до 70 мс $u=50$ В; 5) от 70 мс до конца периода $u=0$. Написать фрагмент программы для вычисления мгновенного значения U при произвольном $T \geqslant 0$ с использованием встроенных функций и логического IF.

Оператор цикла DO, *оператор* CONTINUE. Эти операторы предназначены для реализации структуры «цикл-до» и аналогичны оператору FOR...TO...STEP Бейсика. Конструкция оператора

$$\text{DO } m\ i=i_0,\ i_n[,\Delta i]$$

где m — метка исполнимого оператора, являющегося последним оператором тела цикла, кроме операторов GO TO, IF, PAUSE, STOP, RETURN или другого DO; i — управляющая переменная цикла только целого типа; i_0, i_n, Δi — целые положительные константы или целые переменные; i_0 — начальное значение переменной цикла; i_n — конечное значение; Δi — шаг (если Δi с предшествующей запятой опущен, принимается $\Delta i=1$). Величины i, i_0, i_n, Δi не могут быть элементами массивов.

Оператор DO выполняется следующим образом: переменной i присваивается значение i_0; выполняются операторы, начиная со следующего за DO до оператора с меткой m включительно; i увеличивается на Δi; проверяется условие $i \leqslant i_n$ и, если оно соблюдается, все упомянутые операторы выполняются снова; если не соблюдается, выполняется оператор, следующий за оператором с меткой m.

Из этого алгоритма работы оператора DO следует, в частности, что если $i_n \leqslant i_0$, тело цикла все же выполнится один раз.

В отличие от оператора FOR...TO...STEP Бейсика в Фортране переменная цикла может быть только целого типа, шаг Δi — только положительным, а i_0, i_n, Δi не могут быть любыми арифметическими выражениями.

Циклы могут быть вложенными, причем один и тот же оператор может быть указан в качестве конца цикла (в Бейсике надо писать несколько операторов NEXTi). Например, для сложения матриц C=A+B размером 5×10 можно написать следующую программу:

```
C СЛОЖЕНИЕ МАТРИЦ
      DIMENSION A (5,1Ø), B(5,1Ø), C(5,1Ø)
<ввод значений матриц A и B>
      DO 1 I=1,5
      DO 1 J=1,1Ø
1     C(I,J)=A(I,J)+B(I,J)
<печать результатов>
      END
```

Вначале при I=1 выполняется 10 раз внутренний цикл, в котором J меняется от 1 до 10, т. е. складываются все элементы 1-й строки. Затем при I=2 складываются элементы 2-й строки и т. д.

Как и в Бейсике, вход в цикл можно осуществлять только через оператор DO. Выход из цикла по операторам GO TO, IF можно осуществлять до полного завершения цикла, при этом переменная цикла i сохраняет то значение, которое она имела не-

посредственно перед выполнением GO TO или IF. Если цикл заканчивается «естественным» путем, переменная i считается неопределенной.

Внутри цикла не должно быть операторов, изменяющих значения переменных i, i_0, i_n и Δi. Передача управления на последний оператор нескольких вложенных циклов (с меткой m) разрешается только из самого внутреннего цикла. Разрешается обращение к подпрограмме из тела цикла DO с последующим возвратом в этот же цикл, а также переход по IF или GO TO в части программы, написанные вне цикла, с возвратом в этот цикл.

Пример использования «досрочного» выхода из цикла. Пусть имеются N результатов вычислений (или измерений) мгновенных значений напряжения на токоприемнике локомотива в виде массива U, причем $N \leqslant 1000$. Требуется определить номер значения и величину U в первом же случае, когда оно будет ниже, например, 2700 В. Если такого значения не обнаружится, напечатать об этом сообщение.

```
DIMENSION U (1ØØØ)
<ввод или вычисление N, U(1), U(2)...U(N)>
      DO 1 I-1,N
      IF (U(I).LT.27ØØ.)GO TO 2
1     CONTINUE
<печать сообщения «U<2700 В НЕ НАЙДЕНО»
      STOP
2<печать значений I и U(I)>
      END
```

В данном примере использован оператор CONTINUE (продолжать), который может завершать цикл, но не совершает явных действий над данными и по управлению программой. Он как бы указывает метку для операторов перехода и DO, применяется всегда, когда надо избежать окончания цикла операторами GO TO, IF, PAUSE, STOP, RETURN.

Упражнение 10. На двухпутном участке железной дороги постоянного тока с односторонним питанием, параллельным соединением контактной сети имеются N_1 поездов в нечетном и N_2 в четном направлениях с токами T_{11}, T_{12}, ..., T_{1N1} и T_{21}, T_{22}, ..., T_{2N2} (N_j не более 10). Написать фрагмент для вычисления тока F фидера, питающего участок (он равен сумме токов всех поездов).

Упражнение 11. Тяговая характеристика электровоза $F(V)$ задана по 20 точкам (узлам) нерегулярной таблицей (т. е. значения аргумента в последовательных точках функции отстоят друг от друга на неодинаковый шаг). Написать фрагмент для вычисления $F_0 = F(V_0)$ (при произвольной V_0), считая, что между узлами таблицы функция $F(V)$ линейна.

Операторы STOP, PAUSE, END используются для остановки программы.

Оператор STOP или STOP n,

где n — целая константа без знака ($1 \leqslant n \leqslant 99999$), прекращает выполнение программы. Если указана n, на пульт оператора (дежурного) ЭВМ выдается текст «STOP n». Следующий за STOP исполнимый оператор должен иметь метку.

Оператор PAUSE может иметь следующие формы записи:

```
PAUSE
PAUSE n
PAUSE 'сообщение'
```

Здесь $1 \leqslant n \leqslant 99999$; 'сообщение'— текстовая константа.

По этому оператору работа программы приостанавливается, дежурному на ЭВМ выдается текст «PAUSE», «PAUSE n» или «PAUSE сообщение». Дежурный может продолжить работу программы по команде с пульта.

Оператор PAUSE используется при необходимости организовать какие-либо действия дежурного: сменить бобину с магнитной лентой, подогнать бумагу на печатающем устройстве и т. п.

Оператор END неисполнимый и должен быть физически последним в тексте программы. Он не может содержать метку. Если в программе не содержится оператора STOP и в процессе ее выполнения ЭВМ доходит до END, работа программы также прекращается. Начинающему программисту не следует пользоваться оператором PAUSE. Примеры применения оператора STOP приведены выше.

5.5. Ввод-вывод информации

Под вводом-выводом понимается обмен информацией между внешними носителями данных и оперативной памятью ЭВМ. Носителями данных являются, например, перфокарты, бумага с напечатанными символами, магнитные ленты, магнитные диски, буферная память дисплея, содержимое которой отображается на экране (кратко говорят «экран дисплея») и т. п.

Одни носители, например бумага для печати, предназначаются только для вывода информации. Перфокарты чаще всего используются для ввода; магнитные ленты, диски, дисплеи — для ввода и вывода.

С «точки зрения» программы на Фортране все данные, вводимые и выводимые, организованы в файлы (file — ряд, вереница). Файл состоит из совокупности записей, т. е. логически связанных между собой единиц информации.

В зависимости от организации доступа к записям файлы Фортрана различают последовательные и прямого доступа. Последние будут рассмотрены отдельно.

Файл последовательного доступа содержит n информационных записей $R_1 - R_n$ (рис. 5.2), которые каким-либо образом отделяются друг от друга (разделители показаны штриховкой). Вслед за последней информационной записью R_n помещена запись специального вида EOF (end of file), которая распознается программой как признак конца файла.

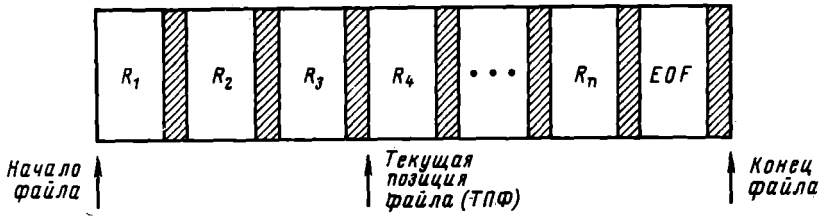


Рис. 5.2. Файл последовательного доступа

При запуске программы файл «открывается», т. е. указатель Текущая позиция файла (ТПФ) ставится в начало файла. При первом обращении к файлу считывается или записывается R_1, ТПФ устанавливается перед R_2; при повторном обращении к этому же файлу считывается (записывается) R_2 и т. д. При попытке считать EOF в качестве информационной записи в программе возникает ситуация «считан EOF», которая может приводить либо к прекращению работы программы, либо к другим результатам, предусмотренным программистом. В выводном файле вслед за R_n записывается EOF либо автоматически по окончании работы программы, либо по специальному оператору.

Стандартным вводным файлом является совокупность перфокарт. Каждая карта рассматривается как одна запись, содержащая 80 байтов информации (1 символ в колонке кодируется одним байтом) Карты физически отделены друг от друга, ТПФ определяется номером карты, лежащей первой в приемном кармане устройства ввода. Карта EOF имеет специальную кодировку, например, в операционной системе ОС ЕС ЭВМ она содержит символы/* или//.

В случае перфокарточного ввода ТПФ перемещается по мере чтения карт и возвратится к предыдущим записям после того, как они прочитаны, нельзя.

Стандартным выводным файлом в Фортране является печать с помощью алфавитно-цифрового печатающего устройства (АЦПУ), размещающего на бумаге до 120 символов в строчке. Запись в этом файле содержит до 121 байта (1 символ — для управления печатью), EOF не печатается.

При использовании магнитных лент и дисков длина отдельной записи может быть различной. ТПФ перемещается автоматически, т. е. программист не должен заботиться о подсчете введенных или выведенных записей. Фортран располагает средствами возвращения ТПФ и повторного открытия файла.

Обращение к различным файлам осуществляется по их номерам. Номер файла в Фортране может принимать значения от 1 до 15 в ДОС и от 1 до 99 в ОС ЕС ЭВМ. В конкретной реализации ОС в различных вычислительных центрах максимальный номер файла может быть установлен меньшим 99. Кроме того, существует

стандартное закрепление некоторых номеров файлов за системными устройствами ввода-вывода.

Устройство	Номер файла в ДОС	Номер файла в ОС
Перфокарточный ввод	1	5
Вывод на перфокарты	2	7
Вывод на АЦПУ	3	6
Вывод на пульт дежурного ЭВМ	15	—

Операционные системы располагают средствами, позволяющими программисту отказаться от стандартного закрепления номеров файлов. В дальнейшем изложении материала данной главы будут подразумеваться только стандартные номера системных файлов в ОС ЕС.

Операторы ввода-вывода последовательного доступа. Начинающему программисту необходимо освоить употребление операторов, позволяющих программировать ввод с перфокарт и вывод на АЦПУ, поэтому в данном разделе операторы ввода-вывода будут поясняться на соответствующих примерах. Файлы последовательного доступа на магнитных носителях обрабатываются теми же операторами.

Отличия состоят в длине записей и способах кодирования информации в них.

Оператор ввода READ. Он имеет следующую конструкцию:

$$\text{READ } (k[,f][,\text{ERR}=m_1][,\text{END}=m_2])\,n_1,\ n_2,\ \ldots\ n_j$$

где k — номер файла в виде целой константы без знака или целой переменной; f — метка оператора FORMAT, или имя массива форматов, или имя списка NAMELIST; m_1 — метка, на которую передается управление при возникновении ошибок (errors) в операциях чтения и передачи данных. Параметр $\text{ERR}=m_1$ необязательный; m_2 — метка, на которую передается управление при чтении записи EOF Параметр $\text{END}=m_2$ необязательный.

Параметры $\text{ERR}=m_1$ и $\text{END}=m_2$ могут записываться в любом порядке. Начинающему программисту пользоваться ими не следует. Параметры n_1, n_2, ..., n_j — список ввода, т. е. имена переменных, элементов массивов, имена массивов, а также список типа «неявная форма DO», поясняемый далее. Список может быть пустым, тогда очередная запись файла (например, перфокарта) пропускается.

Данные, содержащиеся на носителе файла k, преобразуются из внешнего кода во внутренний код ЭВМ в соответствии с указаниями, записанными в операторе FORMAT, с меткой f. Их значения присваиваются последовательно объектам n_1, n_2, ... n_j, перечисленным в списке ввода. Если в списке содержатся элементы массивов, индексы которых определяются арифметическим выражением, всем входящим в него переменным должны быть присвоены значения до выполнения операции ввода. Если в списке

записано имя массива (без индексов в скобках), вводится весь массив в том порядке, в каком он расположен в памяти.

Пример. Пусть имеется фрагмент программы:

```
DIMENSION A(4), B(3,2), K(5), D(5)
I=3
J=2
READ (5,1ØØ) N, K, D(1), D(I), D(J), D(5), A, B
1ØØ FORMAT...
...
```

Пусть на перфокартах, положенных в приемный карман устройства ввода, последовательно набиты числа: 3 11 12 13 14 15 1. 2. 3. 4. 5. 6. 7. 8. 11. 12. 13. 14. 15. 16. Тогда в результате выполнения приведенного фрагмента запускается устройство чтения перфокарт, а переменным и элементам массивов присваиваются следующие значения:

N=3; K(1)=11; K(2)=12; ...K(5)=15; D(1)=1.0; D(2)=3.0;
D(3)=2.0; D(4) — не определен; D(5)=4.0; A(1)=5.0;
A(2)=6.0; A(3)=7.0; A(4)=8.0; B(1,1)=11.0; B(2,1)=12.0;
B(3.1)=13.0; B(1,2)=14.0; B(2,2)=15,0; B(3,2)=16.0

Список «неявная форма DO» выглядит следующим образом:

$$(n_1,\ n_2,\ \ldots\ n_j,\ i=i_0,\ i_n,\ \Delta i)$$

где n_1, n_2, ..., n_j — переменная, элемент массива, имя массива, список «неявная форма DO'»; i, i_0, i_n, Δi имеют тот же смысл, что в операторе DO.

Неявную форму DO обычно применяют для ввода массивов, если требуется вводить их нецеликом, или данные на картах расположены не в том порядке, как они размещаются в памяти ЭВМ. Например, если нужно ввести только три элемента массива K предыдущего примера, оператор READ можно записать:

READ (5, 1ØØ) N, (K(L), L=1,3), D(1) и т. д.

Соответственно на картах надо исключить числа 14 и 15, а элементы K(4) и K(5) не будут определены.

Тот же результат можно получить в данном примере, если записать READ (5,1ØØ)N, (K(L), L=1, N), D(1), и т. д., так как в процессе работы оператора N получает сначала значение 3 (оно считывается с карты первым) и дальше используется в качестве конечного значения переменной цикла L.

Неявные циклы DO, как следует из определения их структуры, могут быть вложенными. Например, если данные для массива B(3,2) подготовить на картах по строкам: 11. 14. 12. 15. 13. 16., то ввод надо организовать следующим образом:

READ (5,1ØØ) ((B(L, M), M=1,2), L=1,3)

и тогда элементы массива получат те же значения, что в последнем примере.

Если используется дополнительная форма оператора READ

READ *f*,<список>

то номер файла не указывается, а подразумевается устройство ввода с перфокарт.

Такая запись эквивалентна следующим записям:

в ДОС

READ(1,*f*) <список>

в ОС

READ (5, *f*) <список>

Если программист не знает, в какой операционной системе он будет использовать программу, ввод с перфокарт лучше записывать с помощью дополнительной формы оператора READ.

Операторы вывода WRITE, PRINT, PUNCH. Структура оператора WRITE (основного) имеет вид:

WRITE (*k*[,*f*]) [<список вывода>]

Параметры *k*, *f* имеют тот же смысл, что и в операторе READ; <список вывода> аналогичен списку ввода.

По оператору WRITE значения переменных, элементов массивов в порядке их перечисления в списке вывода преобразуются во внешний код в соответствии с указаниями оператора FORMAT с меткой *f* и выводятся на внешний носитель в файл *k*.

Пример. Нерегулярную таблицу функции $F(V)$ (см. упр. 11) вывести на печать, снабдив пары значений $V(I)$ и $F(I)$ номером точки I:

```
...
WRITE (6, 2∅∅) (I, V(I), F(I), I=1,2∅)
2∅∅ FORMAT...
```

Дополнительные операторы PRINT и PUNCH используются для вывода на системную печать (АЦПУ) и перфоратор соответственно. Например, в ОС следующие операторы эквивалентны:

PRINT *f*, <список> WRITE (6,*f*) <список>
PUNCH *f*, <список> WRITE (7,*f*) <список>

Оператор WRITE (предыдущего примера можно записать

PRINT 2∅∅, (I, V(I), F(I), I=1,2∅)

Лучше всегда пользоваться последней формой оператора для вывода на АЦПУ.

Оператор FORMAT. Это неисполнимый оператор, он может помещаться в любом месте программы и обязательно имеет метку, на которую ссылаются в операторах ввода-вывода. На один и тот же оператор FORMAT можно ссылаться в нескольких операторах READ и/или WRITE(PRINT, PUNCH).

Оператор FORMAT содержит информацию для преобразования данных из внешнего представления во внутреннее при вводе и обратно при выводе.

Внешнее представление — это, например, сочетание пробивок на перфокарте, или набор символов на бумаге, отпечатанный с помощью АЦПУ, или набор символов на экране дисплея.

Внутреннее представление в памяти ЭВМ — это двоичные числа с фиксированной или плавающей точкой для числовых и логических данных, занимающие 1, 2, 4, 8 или 16 байтов в зависимости от

типа. Каждый символ текстовых данных кодируется одним байтом в соответствии с кодом ДКОИ (двоичным кодом для обмена информацией).

Конструкция оператора:

$$f \quad \text{FORMAT } (c_1, c_2, \quad c_n)$$

где f — метка; c_1, c_2, ..., c_n — спецификаторы поля (форматы), разделенные запятыми или символами «/», а также группы форматов.

Спецификаторы поля (форматы) определяют, в каком виде располагаются данные на внешнем носителе и правила преобразования. Каждому данному в списке ввода (вывода) оператора READ (WRITE) должен соответствовать формат оператора FORMAT. Наоборот, некоторые форматы могут не иметь соответствующих объектов в списках ввода-вывода.

При вводе-выводе данных целого типа используется формат вида Iw, где I — форматный код преобразования данных типа INTEGER, w — число позиций (колонок перфокарты или символов в строке АЦПУ), отводимых данным на внешнем носителе. Позиции, занимаемые одним числом, называют полем числа (или другого данного).

При вводе по формату Iw в поле перфокарты длиной w должно быть набито число в виде целой константы, причем пробелы рассматриваются как нули.

Например, запись в программе

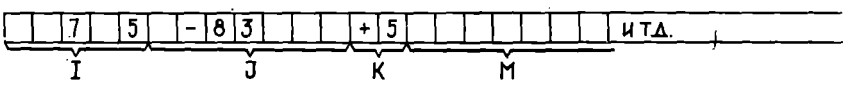
```
  ...
  READ 35, I, J, K, M, N1, N2
  ...
35 FORMAT (I5, I7, I2, I1Ø, I1Ø, I1Ø)
```

предполагает, что данные на перфокарте подготовлены следующим образом: числовое значение переменной I отперфорировано в первых пяти колонках карты (т. е. в колонках 1—5); значение J — в следующих семи колонках (6—12); значение K — в колонках 13, 14; значения M, N1, N2 — по 10 колонок каждое, т. е. до 44-й колонки включительно. Таким образом, в данном примере колонки с 45 по 80 не содержат полезной информации, они не воспринимаются программой. Данные на карте можно располагать как угодно, нужно только следить, чтобы общая длина полей не превышала 80.

Повторяющиеся форматы можно не выписывать подряд, достаточно перед форматом поставить коэффициент повторения (в примере 3). Тогда оператор будет выглядеть так:

```
35 FORMAT (I5, I7, I2, 3I1Ø)
```

Если на карте, например, набито



то в результате работы программы переменные получат значения: I=705, J=—83000, K=5 и т. д.

Во избежание ошибок при программировании и подготовке информации целесообразно отводить одинаковые поля под однотипные данные, пробивать нули и число всегда «прижимать» к правой границе отведенного ему поля.

При выводе по формату Iw числовое значение переменной целого типа печатается в виде целой константы (знак «+» опускается), «прижатой» к правой границе поля. Если отведенного поля недостаточно для размещения числа, все поле заполняется звездочками.

Пусть переменные I, J, K имеют значения, указанные выше. По операторам

```
...
PRINT 36, I, J, K
...
36 FORMAT (3 I 5)
```

будет отпечатана одна строка, содержащая ␣7∅5∗∗∗∗∗ ␣␣␣␣5 <в остальных 106 позициях строки — пробелы>

Здесь не выведен первый пробел внешнего представления I, так как при обращении в АЦПУ первый символ выводимой записи управляет печатью. Управляющие символы следующие: ␣ (пробел) — продвинуть бумагу на 1 строку перед печатью; 0— продвинуть на 2 строки; 1— продвинуть до первой строки на следующей странице; + — не продвигать.

Другие символы могут вызвать сбои в работе АЦПУ В некоторых ВЦ из-за особенности операционных систем АЦПУ не реагируют на управляющие символы, а всегда продвигают бумагу на 1 строку, причем первый символ записи все равно не печатается.

Вместо значения J напечатаны звездочки, так как число «—83000» требует 6 позиций, а мы отвели только 5. Вообще лучше отводить поля «с запасом», потому что малые пропуски между соседними числами или их отсутствие делает печать результатов неудобочитаемой.

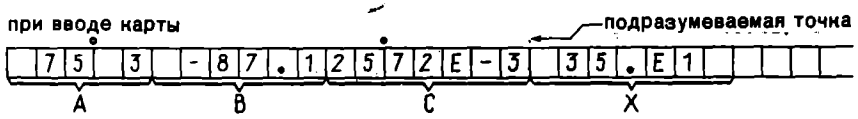
Данные вещественного типа длиной 4 байта (REAL∗4) преобразуются по форматам Fw.d и Ew.d, где F и E — форматные коды; w— длина поля данного на внешнем носителе; d— число позиций в дробной части числа или количество значащих цифр (при выводе по формату E).

Формат F описывает числа на внешнем носителе, записанные в виде вещественных констант с фиксированной точкой, формат E— в экспоненциальной форме. По формату F можно вводить числа и в экспоненциальной форме.

При вводе по форматам F и E не обязательно перфорировать точку, она как бы подразумевается перед d последними цифрами числа (пробелы эквивалентны нулям). Знак «—» надо ставить левее точки, а порядок при экспоненциальной форме записи прижимать к правой границе поля.

Например, по программе

```
      READ 17, A, B, C, X
17    FORMAT (F5.2, F6.4, 2E7.2)
```



переменным будут присвоены значения: A=75.03; B=−87.1; C=25. 72E−3 (т. е. 0,02575); X=35E10 (т. е. $35 \cdot 10^{10}$).

Из примера видно, что параметр d. игнорируется, если точка набита на карте явно. Числу X присвоен порядок, равный 10, потому что самый правый пробел в поле этого числа воспринят, как 0.

Так же как и при подготовке целых чисел, рекомендуется однотипные данные готовить всегда в одинаковом формате. Точку лучше всегда набивать явно, это исключает задание неверной информации при смещении по колонкам. Из этих же соображений лучше не пользоваться форматом E при вводе.

При выводе по формату F число располагается так, что младший разряд занимает крайнюю правую позицию в поле, причем число предварительно округляется. Если целая часть числа, десятичная точка и знак «−» занимают меньше w−d символов, левая часть поля заполняется пробелами; если больше — отведенное поле мало́ и заполняется звездочками.

Пусть переменная X выводится по формату F5.1, тогда:

X в памяти	X на печати
−81.6	−81.6
3.567	␣␣3.6
0.004	␣␣0.0
209.	209.0
−209.	*****

Формат F имеет преимущество, заключающееся в том, что вводимые и выводимые данные более удобочитаемы, чем в экспоненциальной форме. Однако он требует от программиста предварительного знания порядка величин, участвующих в вычислениях.

При выводе по формату E число всегда нормализуется — преобразуется так, чтобы первая значащая цифра мантиссы стояла после точки и последний разряд округлялся, т. е. при выводе мантисса будет иметь вид правильной дроби с d знаками после точки. Нуль печатается без указания порядка:

Число в памяти	Число на печати по E10.3
−0.00742899	−0.743E−02
5237000.0	␣0.524E+07
25.12	␣0.251E+02
0.	␣0.0␣␣␣␣␣␣␣

Так как семь позиций при выводе по формату E используются для знака числа, нуля, точки, символа E, знака порядка и его значения, то при составлении программы надо писать $w \geqslant d+7$. Формат E гарантирует вывод любого числа с заданной относительной точностью.

Комплексные данные вводятся-выводятся по форматам F и/или E, только для каждого комплексного данного в списке ввода (вывода) нужно указывать два формата в операторе FORMAT. Пусть при работе программы

```
      COMPLEX U, Z, TOK
      READ 1Ø, U, Z
1Ø    FORMAT (4F5.Ø)
      TOK=U/Z
      TOKM=CABS (TOK)
      PRINT 11, U, Z, TOK, TOKM
11    FORMAT (7F7.1)
      END
```

карта с данными содержала следующую запись:

␣5Ø.Ø.␣Ø.␣␣Ø.4␣␣Ø.3␣␣

Тогда на печать выйдет такая запись

␣5ØØ.Ø␣␣␣␣Ø.Ø␣␣␣␣Ø.4␣␣␣␣Ø.3␣␣8ØØ.Ø
␣−6ØØ.Ø␣1ØØØ.Ø

(первый пробел не напечатался, как управляющий символ)

Иными словами, действительная и мнимая части комплексного числа преобразуются каждая в отдельности по правилу преобразования вещественных данных.

Текстовые данные могут записываться непосредственно в виде текстовых констант в качестве элемента списка формата. Обычно эта возможность используется для снабжения выводимых данных пояснениями, для формирования заголовков таблиц и т. п. Текстовые константы можно записывать как в виде wH<w символов>, так и в виде литерала в апострофах. Например, если заменить оператор FORMAT в программе предыдущего примера (написан в двух строках программы):

```
11 FORMAT (3H␣U=,2F7.1,'␣␣Z=',2F7.1,
     *'␣␣␣TOK=',2F7.1,'␣␣МОДУЛЬ␣ТОКА=',F7.1)
```

(6-я колонка)

то будет напечатано (в одной строке):

U=␣␣5ØØ.Ø␣␣␣␣Ø.Ø␣␣Z=␣␣␣␣Ø.4␣␣␣␣Ø.3
␣␣␣TOK=␣8ØØ.Ø␣−6ØØ.Ø␣␣МОДУЛЬ␣ТОКА=␣
1ØØØ Ø

(пробел перед «U=» используется, как управляющий).

Если текстовая константа записана в операторе FORMAT взаимодействующем с оператором READ, с карты считываются

символы, число которых равно числу символов в константе, и эти символы с карты заменяют константу в операторе FORMAT.

Начинающему программисту не следует пользоваться текстовыми константами в качестве форматов при вводе.

Если обмен текстовой информацией осуществляется между внешними носителями и областями памяти, обозначенными именами (переменными или массивами), для преобразования используется формат вида Aw, где w — длина поля текста на внешнем носителе. Количество передаваемых символов зависит от длины l соответствующей переменной в байтах (т. е. от ее типа). Как правило, употребляют $w=l$, но в случае их неравенства обмен происходит между правой частью поля w (при $w>l$) и левой частью области памяти (при $w<l$). Пусть, например, по программе

```
      COMPLEX B
      INTEGER*2 K(2)
      X=35.72
      READ 43, A, B, K
43    FORMAT (A4, A8, 2A2)
      PRINT 44, A, X, K
44    FORMAT ('␣', A4, F5.1, 2A2)
      END
```

вводится карта с текстом (с 1-й колонки) ПРОГРАММИРОВАНИЕ. Переменные получат следующие значения:
X=35.72; A='ПРОГ'; B='РАММИРОВ'; K(1)='АН'; K(2)='ИЕ', а на печать будет выведено

ПРОГ␣35.7АНИЕ

(текстовая константа '␣' используется как управляющий символ).

Данные двойной точности (REAL*8) преобразуются по формату Fw.d аналогично обычным вещественным числам типа REAL*4, а также по формату Dw.d, имеющему тот же смысл, что и формат Fw.d для чисел REAL*4. Отличие заключается в том, что вместо символа E перед порядком печатается символ D, а при вводе можно применять как разделить E, так и D.

Универсальный формат для числовых данных Gw.d может применяться при вводе-выводе целых и вещественных чисел любой длины (для комплексных он записывается дважды).

При передаче целых числе Gw[.d] эквивалентен Iw (d игнорируется и может быть опущен).

При вводе вещественных чисел Gw.d эквивалентен Fw.d, т. е. допускает запись числа на карте в любой форме, разрешенной для вещественных констант.

При выводе вещественных чисел формат Gw.d работает в зависимости от значения данного X следующим образом: если $X<0.1$, то Gw.d соответствует Ew.d; если $0{,}1\leqslant X\leqslant 1$, то число печатается в формате с фиксированной точкой F(w−4).d, а в правой части поля (на месте порядка) печатаются четыре пробела; если $1<X<10^d$, число печатается в фиксированном формате F (без порядка) в поле w −4 так, чтобы выводилось d значащих цифр; при $X\geqslant 10^d$ Gw.d эквивалентен Ew.d.

При выводе данных двойной точности, когда $X<0.1$ или $X\geqslant 10^d$ Gw.d, эквивалентен Dw.d.

Примеры вывода числа X по формату G10.3:

X в памяти	X на печати
0.037	␣0.370E—01
0.374	␣0.374␣␣␣␣
37.4	␣37.4␣␣␣␣
374.2	␣374.␣␣␣␣
37420.	␣0.374E+05

Формат G особенно удобен при отладке программ.

Данные логического типа передаются по формату Lw и Gw[.d]. При вводе значение .TRUE. набивается на карте в поле из w символов в виде буквы T с любым числом пробелов перед ней; за буквой T до конца поля могут следовать любые символы. Аналогично значение .FALSE. задается буквой F.

При выводе печатаются буквы T и F соответственно, прижатые к правой границе поля.

Шестнадцатеричные данные представляются на внешнем носителе цепочкой из w шестнадцатеричных символов (в виде константы) и передаются в память ЭВМ и обратно по формату Zw. Длина l переменной, с которой осуществляется передача, как правило, равна w. При $l \neq w$ действуют те же правила, что и при обмене текстовыми данными.

Дополнительные форматные коды позволяют более гибко управлять размещением данных.

Формат wX используется для пропуска w символов при вводе или для вывода w пробелов.

Например, по операторам PRINT 74,X и

74 FORMAT ('␣␣␣␣␣␣␣␣␣␣', F5.2)

или

74 FORMAT (1∅X, F5.2)

сначала будет выведен управляющий символ, продвигающий бумагу, затем девять пробелов, а начиная с 10-й позиции в строке — числовое данное.

Формат Tw указывает, что передача данных, описываемых следующими за ним форматами, должна начинаться с позиции № w. Обычно этот формат используется при печати таблиц. При этом следующие в порядке написания T не обязательно должны иметь возрастающие w. Нужно учитывать, что первый пробел является управляющим, поэтому фактически номер позиции на 1 меньше. Например, приведенным выше операторам эквивалентен

74 FORMAT (T11, .F5.2)

Перед любым числовым форматом для передачи вещественных чисел можно записать код nP, где n— целая константа ($-127 \leqslant n \leqslant 127$). Эта конструкция называется «масштабный множитель», он применяется в основном с форматом F для изменения числового значения данного на внешнем носителе по сравнению с внутренним в 10^n раз. Отсутствие масштабного множителя эквивалентно ∅P. Повторитель форматов, если нужно, записывается после масштабного множителя, например, 3P5F8.3.

При вводе число, набитое на карте, перед присвоением его переменной делится на 10^n Например, на карте набито 371.852; по формату 3PF7.3 в память будет введено значение 0.371852, а по формату —2PF7.3 вводится 37185.2. Если число набито в экспоненциальной форме, масштабный множитель игнорируется.

При выводе число в памяти умножается на 10^n При выводе по форматам E и D меняется положение десятичной точки в мантиссе на n позиций ($n > 0$— вправо, $n < 0$— влево), но порядок соответствует значению числа в памяти. Например, пусть в памяти X=371.852; при выводе по E13.6 будет напечатано ␣0.371852E+03; по 1PE13.6 напечатается ␣ 3.718520E+02.

При выводе по формату G и $0.1 \leqslant X \leqslant 10^d$ масштабный множитель игнорируется, а в прочих случаях действует так же, как в форматах E и D.

Взаимодействие операторов ввода-вывода с оператором FORMAT; *группы форматов, разделители.* Когда количество данных в списке ввода (вывода) равно числу форматов (исключая форматы X, T, H и литералы) взаимодействие элементов обоих списков очевидно: последовательно просматривается список форматов опера-

тора FORMAT от левой до правой скобки; каждому формату последовательно ставится в соответствие одно данное из списка ввода (вывода). Форматы X, T вызывают пропуск соответствующего числа позиций на носителе и изменение позиции в записи. При формате H и литерале текстовые данные обмениваются между оператором FORMAT и носителем. Например, при I=7, X=28.4, B(1)='ИВАН', B(2)='ОВ␣␣' по программе

```
          DIMENSION B(2)
          . . .
     47   FORMAT (T15, 2A4, 4X, 'ТАБЕЛЬНЫЙ␣N', I3, 5X,
'СДЕЛАЛ', F6.1, '%')
          PRINT 47, B, I, X
          END
```

будет отпечатано

```
          14 позиция
          ↓
␣␣...␣ ИВАНОВ ␣␣␣␣ТАБЕЛЬНЫЙ␣N␣␣7␣␣␣␣␣␣
13 пробелов (1—управляющий)
СДЕЛАЛ ␣␣28.4%
```

Если в списке ввода (вывода) данных меньше, чем соответствующих форматов в операторе FORMAT, после ввода (вывода) последнего элемента списка оставшиеся форматы игнорируются. Например, если в приведенном выше примере записать PRINT 47, B, I, на печать выйдет

␣␣...␣ИВАНОВ␣␣␣␣ТАБЕЛЬНЫЙ␣N␣␣7␣␣␣␣␣СДЕЛАЛ

Если в операторе FORMAT исчерпан список форматов, а список ввода (вывода) не исчерпан, просмотр FORMAT'*а* возобновляется от левой скобки, при этом осуществляется переход на следующую запись — карту (строку). Для примера печати таблицы *F(V)*, приведенного выше, можно предложить такой оператор

```
20 FORMAT(15,2F8.2,15,2F8.2,15,2F8.2)
```

Тогда таблица будет выглядеть следующим образом (в скобках <> подразумеваются значения соответствующих элементов массивов в формате F8.2):

␣␣␣1<V(1)> <F(1)>␣␣␣␣2<V(2)> <F(2)>␣␣␣␣
3<V(3)> <F(3)>␣␣␣4<V(4)> <F(4)>

␣␣19<V(19)> <F(19)>␣␣␣20<V(20)> <F(20)>

В данном примере 3 раза повторена одна и та же последовательность форматов I5, 2F8.2. Можно записать их 1 раз, заключить в скобки и снабдить повторителем:

```
20 FORMAT (3(I5, 2F8.2))
```

Список форматов, заключенный в скобки и снабженный, если надо, повторителем, называется группой форматов. Группы могут быть вложенными.

Если в операторе FORMAT имеются группы форматов, то повторение просмотра FORMAT' *а* возобновляется с последней внешней группы. Рассмотрим еще вариант печати таблицы *F(V)* с оператором

```
2∅∅ FORMAT('␣ТАБЛИЦА␣F(V)',3(I5, 2F8.2))
```

Печать будет выглядеть так:

```
ТАБЛИЦА␣F(V)␣␣␣␣I<V(1)> <F(1)>␣␣␣␣2...
␣␣␣4<V(4)> <F(4)>                и т. д.
```

Как видно, текст «ТАБЛИЦА␣F(V)» вывелся один раз, при первом просмотре FORMAT'*а*. Из-за этого элементы первой строки не совпали по вертикали с элементами следующих строк (таблица «некрасивая»).

В рассмотренных до сих пор примерах элементы списка форматов разделялись запятыми. Существует другой разделитель — дробная черта «/», причем в отличие от запятой она может ставиться несколько раз подряд.

Разделение форматов с помощью «/» означает, что необходимо перейти на новую запись (новую перфокарту или строку печати). Стоящие подряд разделители «/» вызывают пропуск соответствующего количества записей.

Если последний пример усовершенствовать (оператор записан в двух строках программы)

```
2∅∅ FORMAT(//'␣ТАБЛИЦА␣F(V)'/
   ∗3('␣␣I=␣␣␣␣V(I)=␣␣␣F(I)=␣')/3(I5,2F8.2))
   ↑
   (6-я колонка)
```

Тогда начало печати таблицы будет выглядеть следующим образом:

```
   <две пустые строки для отделения таблицы от предыдущего
текста>
   ТАБЛИЦА␣F(V)
␣I=␣␣␣V(I)=␣␣␣F(I)=␣␣␣I=␣␣␣V(I)=
␣␣␣F(I)=␣␣<и т. д.>␣␣␣I<V(1)> <F(1)>␣␣␣␣2
<V(2)> <F(2)>
```

При программировании ввода-вывода большого числа данных (в частности, массивов) следует учитывать, что длина одной записи на перфокарте не может быть более 80, а на АЦПУ более 121 байта. Например, программа

```
  DIMENSION X (2∅)
  READ 7, X
7 FORMAT (2∅ F5.∅)
```

составлена неверно, так как предполагает 100 символов на карте. Массив X можно разместить на двух картах в формате F5.∅, каждое значение будет включать 16 элементов на первой и четыре элемента на второй карте. Оператор 7 можно записать в виде:

```
7 FORMAT (16F5.∅)
```

Также неверно было бы для вывода таблицы $F(V)$ написать

```
2∅∅ FORMAT (6(15,2F8.2))
```

поскольку такой формат описывает выводную запись длиной 126 символов.

Упражнение 12. В примере программы, приведенном на с. 185 оператор FORMAT составлен следующим образом:

а) 1∅∅ FORMAT (615,4F3.1/4F5.2/6F4.∅)

б) 1∅∅ FORMAT (11/512/(4F4.1))

Как должны располагаться на картах числа, приведенные в примере в случаях а) и б)? Точки явно не набиваются.

Упражнение 13. Составьте фрагмент для печати сообщения к программе упр. 11.

Упражнение 14. Составьте фрагмент для программы расчета коэффициента сцепления (см. п. 5.4), обеспечивающий печать результатов.

Запись форматов в массиве. Как следует из конструкции операторов ввода-вывода, в качестве f вместо метки оператора FORMAT может указываться имя массива любого типа. В этом случае данные, присвоенные следующим друг за другом элементам массива f, должны содержать текст списка форматов (вместе со скобками).

Пример 1. Пусть программист не хочет связывать пользователя формой представления на картах элементов некоторого массива X, состоящего из 100 элементов. Тогда в первой карте, стоящей перед картами с числовыми значениями X, пользователь должен будет задать формат, в котором он приготовил прочие карты. Программа будет выглядеть так:

```
    DIMENSION X(1∅∅), F(3)
   ...
    READ 16,F
16  FORMAT (3A4)
   ...
18  READ F,X
   .
```

Предположим, на первой карте набиты символы: (5E12.3). Тогда этот текст первым оператором READ введется в массив F, а выполнение оператора READ с меткой 18 будет эквивалентно выполнению операторов

```
18    READ 18∅,X
18∅   FORMAT (5E12.3)
```

Другими словами, пользователь набил по пять значений массива X на карте в формате E12.3 каждое, т. е. всего 20 карт.

Если при выполнении той же программы первая карта будет содержать текст: (1∅F8.2), то это значит, что массив X набит по 10 значений на карте в формате F8.2 (всего 10 карт).

Пример 2. В процессе выполнения тягового расчета с шагом Δs требуется строить график функции $v(s)$, причем v изменяется от 0 до 100 км/ч, а график строится символом $*$, печатаемым в позициях 20 — 120 строки. На каждый шаг Δs бумага продвигается на одну строку. Создадим массивы: F из четырех элементов и T из 101 элемента, причем присвоим им заранее текстовые значения

```
F(1)='(␣␣␣',F(2)='␣␣␣␣␣',F(3)=',1H*',F(4)='␣␣␣)',T(1)='
=T21␣',T(2)='T22␣',T(3)='T23␣'...,T(101)='T121'
```

Тогда на каждом шаге Δs достаточно выполнить фрагмент

```
   .
   I=INT(V+∅.5)+1
   F(2)=T(I)
   PRINT F
```

Предположим, на каком-то шаге V=2.2 км/ч. Тогда I=3 и следующим оператором элементу F(2) будет присвоено текстовое значение 'T23␣' Выполнение оператора PRINT F в этом случае будет эквивалентно выполнению фрагмента

```
      PRINT 7Ø Ø
7Ø Ø  FORMAT (␣␣␣T23␣,1H*␣␣␣)
```

т. е. отпечатается символ «*» в позиции 22 (пробелы между форматами не имеют значения).

Оператор NAMELIST. В операторах READ (k,f) и WRITE (k,f) с пустым списком переменных в качестве f может задаваться также «имя списка» неисполнимого оператора NAMELIST, имеющего форму

NAMELIST /n_1/список$_1$/n_2/список$_2$...

где n_i — «имя списка» (состоит из 1 — 6 символов, как обычное имя переменной); список$_i$, озаглавленный именем n_i, — список имен переменных или массивов (элементы массивов в списках не разрешаются) Каждое n_i должно быть уникальным в программе, но переменные могут упоминаться в разных списках.

Например, по фрагментам

```
      DIMENSION X(1Ø)
      NAMELIST/LIST1/K,M,X/LIST2/M,SE,FD
      ..
      READ (5,LIST1)
      ...
      WRITE (6,LIST2)
```

вводятся с перфокарт значения переменных K, M, массив X и выводятся на печать M, SE и FD.

Карты, вводимые с помощью READ(k,f) и оператора NAMELIST, обязательно должны содержать: пробел в 1-й колонке; во 2-й колонке первой карты — символ &, вслед за которым набивается «имя списка»; на первой и других картах данные, разделенные запятыми. Данные записываются в форме $a=c$ или $b=<$список$>$, где a — имя переменной или элемент массива, b — имя массива, c — константа любого типа, $<$список$>$ — константы, разделенные запятыми. Если одинаковые константы в списке встречаются m раз подряд, можно писать $m*$ константа. Общее число констант в списке не должно превышать размер массива.

Вслед за последней константой должны стоять символы ␣&END (признак конца данных).

Для приведенного выше фрагмента программы карты могут выглядеть так:

```
␣&LIST1␣M=35, K=12,
␣X=58.3,-57E-5,258.,7*Ø.␣&END
```

Заметим, что порядок перечисления переменных на картах может не совпадать со списком оператора NAMELIST. Более того, некоторые, а то и все переменные могут вообще не упоминаться на карте, например

```
␣&LIST1␣&END
```

В этом случае те переменные и элементы массива, которые не вводятся, сохраняют в памяти свои текущие значения.

При подготовке карт нельзя употреблять пробелы внутри имен и констант. Пробелы в конце числа (перед запятой) воспринимаются, как нули.

Оператор WRITE(k,f) с оператором NAMELIST выводит в первой строке символы ␣&f; во второй и последующих — данные в порядке их перечисления в списке, снабженные символами «ИМЯ=» и разделенные запятой; в последней строке — символы ␣&END. Например, по приведенному выше фрагменту получим

```
␣&LIST2
␣M=35,SE=38.3Ø Ø Ø Ø Ø Ø, FD=Ø.Ø Ø5Ø Ø Ø Ø
␣&END
```

Форма представления чисел при выводе выбирается самой программой.

Бесформатный ввод-вывод. Этот способ передачи данных предпочтителен при использовании магнитных лент (МЛ) и дисков (МД). В принципе с лентами и дисками можно обмениваться информацией и по оператору FORMAT, однако это нерационально, так как требует большего объема носителя и замедляет программу преобразованиями данных.

При бесформатном выводе-вводе данные переносятся из памяти на носитель и обратно в двоичном виде и занимают на носителе столько же байтов, сколько и в памяти. Например, целые и вещественные числа обычной длины — по 4 байта, комплексные — 8 байтов и т. д.

Оператор для бесформатного ввода имеет вид

READ(k [,ERR$=m_1$] [,END$=m_2$]) $n_1,n_2,...,n_j$

Конструкция оператора бесформатного вывода

WRITE(k)$n_1,n_2,...n_j$

«Подключение» файла №k к конкретному физическому устройству (накопителю на МЛ или МД) осуществляется средствами операционных систем. Длина записей определяется длиной списка ввода-вывода и с точки зрения программы на Фортране практически не ограничена. Если в списке ввода длина записи меньше длины записи на носителе, оставшаяся информация в каждой записи пропадает.

Записи обрабатываются по одной при выполнении каждого оператора READ/WRITE. При работе с МЛ и МД можно «возвращать» *ТПФ* (см. рис. 5.2) на одну запись с помощью оператора

BACKSRACE␣k

или в начало файла с помощью оператора

REWIND␣k

Повторением в цикле оператора BACKSPACE можно вернуть *ТПФ* на любое число записей. Если *ТПФ* уже находится в начале, операторы BACKSPACE и REWIND игнорируются.

Оператор END FILE используется с МЛ в системе ОС и позволяет создавать несколько физических файлов, записываемых операторами WRITE с одним тем же номером файла k.

Ввод-вывод прямого доступа. Этот метод доступа возможен только с магнитными дисками.

Магнитные диски ЭВМ серии ЕС представляют собой пакеты скрепленных на одной оси с промежутками дисков, поверхность которых покрыта магнитно-твердым материалом подобно поверхности магнитных лент. Так, в устройствах типа ЕС-5061 используются пакеты из 11 дисков, образующие 20 рабочих поверхностей (самые наружные поверхности — нерабочие). В пространство между отдельными дисками вводятся головки записи/считывания, которые перемещаются по радиусу дисков и могут занимать 200 фиксированных положений. Таким образом, на одной поверхности записывается 200 концентрических дорожек, а всего на пакете образуется 4000 дорожек. Пакеты ЕС-5061 позволяют записывать до 7294 байтов на дорожке, т. е. общая емкость такого пакета около 29 Мбайт. Более совершенные накопители на МД серии ЕС позволяют хранить до 400 Мбайт на пакете.

Удобство хранения информации на МД состоит в том, что головки можно довольно быстро подвести к нужной дорожке и в течение одного оборота диска (около 1/50 с) ее прочитать.

Для реализации этой возможности программными средствами в Фортране установлены файлы прямого доступа.

В файле прямого доступа все записи пронумерованы, начиная с 1, и обращение к ним производится по номеру записи i в произвольном порядке. Конструкция операторов ввода-вывода прямого доступа

READ($k'i$[,f][ERR$=m$]) <список>

и

WRITE(k'i[,f]) <список>

Здесь i — выражение целого типа. При обмене по формату может обрабатываться несколько записей, начиная с i-й; k — номер файла, определенного как файл прямого доступа; f — метка оператора FORMAT или имя массива форматов при обмене с преобразованием по формату. Как правило, при обмене с МД используется бесформатный ввод-вывод, поэтому «,f» опускается; m — метка оператора, которому передается управление при возникновении ошибок чтения данных (необязательный параметр); «список» — список ввода-вывода (может быть пустым, тогда выполняется фиктивное чтение/запись).

Чтобы файлы k_1,k_2... определить как файлы прямого доступа, до первого обращения к ним в программе должен встретиться оператор описания их характеристик

DEFINE FILE$k_1(m_1,l_1,p_1,n_1),k_2(m_2,l_2,p_2,n_2)$... где m_j — максимальное число записей в файле k_j; l_j — максимальная длина одной записи (задается в байтах при $p_j=L$ или E и в четырехбайтовых словах при $p_j=U$); p_j — параметр, описывающий способ передачи данных. Он обозначается одной из трех букв: L — передача по формату и без формата, E — передача только по формату, U — передача только без формата; n_j — имя некоторой целой переменной, называемой связанной переменной. После выполнения оператора READ или WRITE n_j принимает значение, на 1 большее номера последней записанной или считанной записи.

На файлах прямого доступа удобно строить различные справочники, каталоги и т. п.

Для ускорения работы программы с файлами прямого доступа применяется оператор FIND(k'i), обеспечивающий поиск записи i. Например, в программе

```
. . .
    DEFINE FILE 9(1Ø Ø Ø,5,U,KEY)
1   FIND (9'357)
. .
1Ø4 READ (9'357) K,X,A,B,F
```

во время вычислений по операторам, работающим между метками 1 и 104, будет производиться поиск записи № 357 в файле 9. После выполнения оператора FIND переменная KEY=357, а после READ — KEY≠358; K,X,A,B,F примут значения, прочитанные из записи № 357

5.6. Аппарат подпрограмм

Все современные ЭВМ в той или иной степени обладают возможностями использования подпрограмм. Подпрограммы позволяют экономить труд программиста, исключая написание повторяющихся конструкций, а также уменьшают общий объем памяти, занимаемый программой.

Фортран располагает довольно обширным набором подпрограмм математических функций, которые автоматически подключаются к программе пользователя при трансляции. Как и в Бейсике, пользователь может сам составить подпрограммы, но Фортран представляет здесь более богатые возможности.

Оператор-функция. Его конструкция аналогична функции пользователя DEFFN в Бейсике и имеет вид:

$$\text{ИМЯ}(x_1,x_2,\ldots,x_n)=a,$$

где ИМЯ — выбранное программистом имя функции;

$x_1, x_2, \ldots, x_n$ — формальные аргументы $(1 \leqslant n \leqslant 20)$;

a — арифметическое или логическое выражение.

Оператор-функция записывается до первого исполнимого оператора программы, но после операторов описания типа и DIMENSION. Тип результата определяется описанием идентификатора ИМЯ (явным или неявным — по первой букве имени). Формальными аргументами обычно являются имена переменных, но могут быть и имена массивов, если в выражении a эти массивы употребляются без индексов (как аргументы в обращении к подпрограммам FUNCTION).

Имена формальных аргументов могут совпадать с именами любых объектов программы, не оказывая никакого влияния на значения последних. Остальным переменным, встречающимся в операторе-функции, должны быть присвоены значения до обращения к этой функции в программе.

Обращение к функции в тексте программы осуществляется по имени, так же как и к стандартным функциям:

ИМЯ$(b_1, b_2, \ldots, b_n)$.

Здесь $b_1, b_2, \ldots, b_n$ — фактические аргументы, которые могут быть любыми выражениями соответствующего типа.

Пример 1. Имеется N нагрузок сети переменного тока, для которых известны TOK(I),U(I) и угол сдвига фаз FI(I). Требуется определить суммарную активную мощность PSUM.

Без операторов ввода-вывода программа может выглядеть так:

```
      DIMENSION TOK (1ØØ),U(1ØØ),FI(1ØØ)
      P(H,T,F)=H*T*COS(F)
      ...
      PSUM=Ø.
      DO 1 I=1,N
1     PSUM=PSUM+P(U(I),TOK(I),FI(I)).
```

В данном примере использование оператора-функции не имеет особого смысла, так как обращение к ней осуществляется 1 раз. Однако, если задачу усложнить, например рассчитывать активные мощности отдельных групп потребителей, это дало бы выигрыш.

Как уже указывалось, фактические аргументы могут быть любыми выражениями. Пусть в программе приведенного выше примера описана функция P(H,T,F), а на некотором этапе расчета нужно вычислить мощность PH трех нагревателей, потребляющих токи H1, H2, H3 от общей сети напряжением UC (нагреватели имеют $\varphi = 0$). Тогда в программе достаточно записать

```
PH=P(UC,H1+H2+H3,Ø.)
```

Итак, выполнение оператора-функции можно представить таким образом, что вместо него в оператор основной части программы как бы записывается арифметическое выражение a, в котором

вместо каждого формального аргумента записан соответствующий фактический.

Упражнение 15. Составьте оператор-функцию и запишите с ее использованием фрагмент программы для вычисления $z_{кр1}$, $z_{тк}$ и $z_{кт}$, необходимых для определения эквивалентного комплексного сопротивления тяговой сети (см. п. 4.3) при следующих принятых именах: $r_к$ — RK, $r_т$ — RT, $R_к$ — RRK, $R_т$ — RRT (остальные имена соответствуют обозначениям в формулах).

Упражнение 16. Усовершенствуйте программу, составленную в упражнении 15 так, чтобы исключить повторяющиеся записи в выражениях.

Подпрограмма FUNCTION. Если функцию пользователя не удается описать одним оператором-функцией, применяют подпрограмму FUNCTION.

Основная особенность ее состоит в том, что подпрограмма FUNCTION является самостоятельной программной единицей, т. е. имена переменных в ней и метки никак не связаны с другими программами, из которых к ней обращаются, и она транслируется отдельно. Эта единица всегда начинается с оператора

[ТИП] FUNCTION ИМЯ(x_1,x_2,...,x_n)

должна содержать хотя бы один оператор присваивания вида
ИМЯ=ВЫРАЖЕНИЕ или READ(...)ИМЯ

и оканчиваться оператором

END

Здесь ИМЯ — имя функции; x_1,x_2,...,x_n — формальные аргументы, аналогичные оператору-функции; ТИП — описатель типа результата, выдаваемого подпрограммой; при его отсутствии тип REAL или INTEGER определяется по первой букве имени.

Обращение к оператору FUNCTION осуществляется так же, как к оператору-функции. Выполнение подпрограммы должно завершаться оператором RETURN. При этом в вызывающую программу на место обращения к функции по имени передается вычисленное значение. Можно оканчивать работу подпрограммы оператором STOP, тогда выполнение всей программы прекращается. Внутри подпрограммы FUNCTION не должно быть обращений к ней самой.

Помеченная подпрограмма DEFFN'*a* Бейсика по сравнению с подпрограммой FUNCTION обладает тем достоинством, что переменные, не являющиеся формальными аргументами (например, коэффициенты) — общие для основной программы и подпрограммы. В Фортране связь между всеми величинами, входящими в основную программу и подпрограмму, осуществляется только через список аргументов.

Подпрограмма SUBROUTINE, **оператор** CALL. Оператор-функция или подпрограмма FUNCTION могут иметь несколько аргументов, но всегда дают только один явный результат, возвращаемый в основную программу. Во многих случаях необходимо бывает передать в вызывающую программу ряд результатов.

Пример 2. Имеется несколько массивов A, B и т. д. В процессе вычислений требуется найти сумму SA,SB...первых KA,KB... элементов каждого массива, а также максимальный из этих элементов (его значение AMAX, BMAX и индекс NA,NB...).

Средством для решения этой задачи является подпрограмма SUBROUTINE, которая, так же как и FUNCTION, является самостоятельной программной единицей. Она должна начинаться с оператора

SUBROUTINE ИМЯ · [(x_1,x_2,...,x_n)]

и оканчиваться END.

ИМЯ подпрограммы — произвольное, оно не определяет типа результатов, т. е. подпрограмма может создавать результаты разного типа; x_1,x_2,...,x_n — формальные параметры (могут отсутствовать), среди которых аргументы и результаты не различаются и могут записываться в любом порядке. Как и FUNCTION, подпрограмма SUBROUTINE должна завершаться выполнением RETURN или STOP.

Обращение к SUBROUTINE из вызывающей программы осуществляется оператором

CALL ИМЯ (b_1,b_2,...,b_n)

где b_1,b_2,...,b_n — фактические параметры.

Фрагменты вызывающей программы и подпрограмма для приведенного выше примера могут выглядеть так:

```
      DIMENSION A(1∅∅), B(2∅∅)
      .
      CAL SUMAX(A,KA,SA,AMAX,NA)
      ...
      CALL SUMAX(B,KB,SB,BMAX,NB)
      END                (конец основной программы)
      SUBROUTINE SUMAX(X,K,S,XMAX,NX)
      DIMENSION X(K)
      S=∅.
      XMAX=-1.E72
      DO 1 I=1,K
      S=S+X(I)
      IF(X(I)-XMAX)1,1,2
    2 XMAX=X(I)
      NX=I
    1 CONTINUE
      RETURN
      END
```

В подпрограмме SUBROUTINE можно указывать размер массива переменным, причем граница индексов должна быть в списке формальных параметров и ее значение при обращении к подпрограмме должно быть не больше фактического массива. Эта особенность относится и к подпрограмме FUNCTION.

При пользовании переменными границами индексов в формальных массивах размерностью 2 и выше необходимо соблюдать осторожность, так как элементы фактического массива ставятся в соответствие элементам формального массива по порядку их расположения в памяти. Если, например, в основной программе описан массив A(8,8), а при обращении к некоторой подпрограмме указаны границы индексов 3 и 5, то в подпрограмме сформируется такая матрица:

$$\begin{matrix} A_{11} & A_{41} & A_{71} & A_{22} & A_{52} \\ A_{21} & A_{51} & A_{81} & A_{32} & A_{62} \\ A_{31} & A_{61} & A_{12} & A_{42} & A_{72} \end{matrix}$$

Тип, количество и порядок следования фактических параметров в вызывающей программе должны соответствовать типу, количеству и порядку формальных параметров в подпрограммах FUNCTION и SUBROUTINE.

Можно использовать параметры в следующих сочетаниях:

формальный параметр — переменная, фактический параметр — константа*, переменная, элемент массива*, выражение*;

формальный параметр — массив, фактический параметр — массив или элемент массива.

В последнем случае формальный массив в подпрограмме как бы начинается с указанного элемента фактического массива. Параметрам, отмеченным *, в подпрограмме не должны присваиваться значения.

Упражнение 17. В некоторой программе используются комплексные переменные для обозначения напряжений $\dot{U}$ и тока $\dot{I}$ различных нагрузок. Составьте подпрограмму для вычисления полной мощности $\dot{S}=\dot{U}*\dot{I}^*$ — сопряженный комплекс тока) и коэффициента мощности ($\cos\varphi=P/S$, где P — активная составляющая мощности).

Дополнительные возможности подпрограмм. В качестве формального и фактического аргументов можно употреблять имя внешней подпрограммы. При этом в вызывающей программе все такие фактические аргументы должны быть описаны (до первого исполнимого оператора и операторов-функций) с помощью описателя EXTERNAL имя$_1$,имя$_2$

Пример.

Вызывающая программа	Подпрограмма
EXTERNAL SIN, COS	FUNCTION SF (X, FUN)
...	
Y1=SF(A, SIN)	SF=SQRT(1.+(X*
	*FUN(X))**2)
	RETURN
Y2=SF(B, COS)	END

Будут вычислены $y_1=\sqrt{1+(a\sin a)^2}$ и $y_2=\sqrt{1+(b\cos b)^2}$

Обычно выполнение подпрограммы начинается с первого исполнимого оператора, следующего за FUNCTION или SUBROUTINE, называемого основным входом.

В подпрограммах может быть несколько входов со своими именами, тогда выполнение можно начинать с них. Дополнительный вход описывается оператором

ENTRY ИМЯ $(a_1,a_2,\ldots,a_n)$

Обращение выполняется по тем же правилам, как и по основному входу. Число и порядок формальных параметров в списке могут быть другими. Если в подпрограмме FUNCTION имеется несколько входов с именами ИМЯ$_1$, ИМЯ$_2$ и т. д., перед оператором RETURN должно быть выполнено присваивание любому из этих «имен».

Если оператор ENTRY встречается по ходу выполнения подпрограммы, он игнорируется.

Пример подпрограммы для нахождения проекции вектора PX на ось x или PY на ось y по его модулю A и углу между ним и осью x—FI:

```
      FUNCTION  PX  (A,FI)
      B=SIN (FI)
      GO TO 1
      ENTRY  PY (A,FI)
      B=COS (FI)
1     PX=A*B
      RETURN
      END
```

Обращение из основной программы X=PX(D,F) позволит определить проекцию вектора D (угол F) на ось x, а Y=PY(D,F) — его же проекцию на ось y. В операторе с меткой 1 можно было бы написать и PY=A*B.

В подпрограмме SUBROUTINE в качестве формальных параметров можно указать символы «*». Тогда в самой подпрограмме, помимо оператора RETURN, который передает управление вслед за оператором CALL... основной программы, должны быть операторы RETURN 1, RETURN 2...RETURNn, где n — число звездочек в списке формальных параметров. В качестве фактического параметра на месте каждой i-й звездочки надо употребить сочетание символа & с меткой основной программы, куда передается управление после выполнения RETURN i.

Формальным аргументом может быть имя, заключенное в символы «/». Тогда в подпрограмму передается не значение, а адрес фактической переменной, что экономит память при передаче длинных аргументов.

5.7. Управление распределением памяти, присвоение начальных значений

Распределение памяти для размещения переменных и массивов выполняется автоматически с помощью транслятора; при этом программист располагает некоторыми средствами воздействия на этот процесс.

Оператор EQUIVALENCE. С помощью этого неисполнимого оператора можно потребовать от транслятора назначения массивам или переменным с разными именами одних и тех же областей памяти. Конструкция оператора имеет вид

EQUIVALENCE(a_1,a_2...),(b_1,b_2...)...

Здесь a_i,b_i — имя переменной или элемента массива с индексами в виде констант, причем a_i,b_i... не могут быть формальными параметрами.

По этому оператору транслятор назначает одинаковый адрес первого байта всех переменных, перечисленных в одной паре скобок.

Например, пусть написано

```
COMPLEX  Z(3)
DIMENSION  X(6),A(1∅∅),B(2,5,1∅),C(5∅)
EQUIVALENCE  (Z(1),X(1)),(A(1),B(1,1,1)),(A(51),C(1))
```

тогда комплексный массив Z и вещественный X расположатся в одних и тех же шести последовательных словах по 4 байта. Это позволяет, например, обращаться к действительной части комплексных чисел Z(1), Z(2), Z(3), как к X(1), X(3), X(5), а к мнимой, как к X(2), X(4), X(6). Массивы A,B,C займут одну и ту же память, причем массив C — начиная с A(51), т. е. в программе можно, например, записать и Y=A(57), и Y=C(7). Такое совмещение может создавать дополнительные удобства при программировании. В данном примере, в частности, очистку всего массива B проще осуществить операторами

```
    DO 1 I=1,1∅∅
1   A(I)=∅.
```

чем писать три вложенных цикла.

Общие области данных (COMMON). Как неоднократно подчеркивалось, все имена переменных и массивов локализованы в отдельных программных единицах. Передачу значений между ними можно осуществлять через параметры подпрограмм. Фортран представляет возможность другого способа обмена данными между программными единицами — через общие области.

Общая область объявляется оператором

COMMON $a_1,a_2,...,a_n$

где a_1, a_2, ..., a_n — имена переменных или массивов, причем границы индексов массивов можно указывать, как в предшествующих операторах описания DIMENSION, так и в операторе COMMON только целыми константами.

Переменным $a_1 - a_n$ транслятор отведет место, начиная с некоторого фиксированного адреса, подряд по порядку перечисления их. Если в другой программной единице, выполняемой совместно с первой, также присутствует оператор COMMON$b_1,b_2,...,b_n$, переменные $b_1 - b_n$ транслятор поместит, начиная с того же фиксированного адреса. При этом списки $a_1 - a_n$ и $b_1 - b_n$ могут не соответствовать друг другу ни по типу, ни по количеству, ни по общей длине данных.

Пусть имеются две программные единицы

```
COMPLEX Z(1∅)
COMMON Z,X(8),A,B,K(2)
...
CALL SUB
...
END
SUBROUTINE SUB
```

```
COMMON Y(25),Z(10)
...
END
```

При работе подпрограммы SUB будут соблюдаться следующие эквивалентности (слева — переменная в SUB, справа — в основной программе): Y(1) ≡ REAL(Z(1)); Y(2) ≡ AIMAG(Z(1)); Y(3) ≡ ≡ REAL(Z(2)) ... Y(20) ≡ AIMAG(Z(10)); Y(21) ≡ X(1); Y(22) ≡ ≡ X(2) ... Y(25) ≡ X(5); Z(1) ≡ X(6); Z(2) ≡ X(7); Z(3) ≡ X(8); Z(4) ≡ A; Z(5) ≡ B; Z(6) ≡ K(1); Z(7) ≡ K(2).

Элементы Z(8) — Z(10) не будут иметь соответствия в основной программе. Обратите внимание на то, что элементы Y(1) — Y(20) по существу являются вещественными числами, так как комплексное число представлено в памяти парой вещественных. Если же в основной программе элементам K(1) и K(2) присвоены целочисленные значения, употребление в SUB Z(6) и Z(7) в арифметическом выражении приведет к ошибке, так как форма представления целых и действительных чисел разная.

Транслятор не проверяет соответствие типов в области COMMON, ответственность за это целиком лежит на программисте. При распределении памяти транслятор назначает начальные адреса переменных длиной в 2 байта — с границы полуслова, 4 байта — с границы слова, 8 байтов — с границы двойного слова и т. п. Если переменные в списках COMMON — разные по длине и количеству, возможны ошибки, поэтому всегда лучше сначала записать переменные более длинного типа, а потом — более короткие.

По приведенному оператору создается одна общая область. В ряде случаев, в особенности в больших программах, целесообразно организовать несколько областей COMMON. Для этого их можно снабдить именами, заключенными в символы «/». Конструкция оператора, объявляющего помеченные общие области, имеет вид:

COMMON/имя$_1$/$a_{11},a_{12},\ldots,a_{1n}$/имя$_2$/$a_{21},a_{22},\ldots,a_{2m}\ldots$

где имя$_1$, имя$_2$,... — имена помеченных общих областей, которые могут совпадать с именами переменных, но не должны совпадать с именами точек входов ENTRY и именами подпрограмм, участвующих в программе;

$a_{11},a_{12},\ldots,a_{1n};a_{21},a_{22},\ldots,a_{2m}$ — списки переменных и массивов в каждой области.

Переменные, объявленные в COMMON, могут участвовать в операторе EQUIVALENCE, при этом нельзя нарушать логику распределения памяти. Например, нельзя назначить на одно и то же место разные переменные из области COMMON или наложить массив, начало которого «вылезет» за начало области.

Присваивание начальных значений. В отличие от Бейсика, который обеспечивает обнуление всех переменных, Фортран не определяет их значений при запуске программы.

Фактически ячейки, отведенные переменным, могут содержать любые случайные коды.

Для того чтобы на этапе трансляции переменным сообщить определенные значения, используется неисполнимый оператор

DATA $a_1/c_1/,a_2/c_2/,\ldots,a_n/c_n/$,

где $a_1,a_2,\ldots,a_n$ — имена переменных или массивов;

$c_1,c_2,\ldots,c_n$ — константы или списки констант (для массивов), отделенных запятыми. Список констант может иметь вид $m*c$, если константа c должна быть повторена m раз.

Пример. По программе

```
      DIMENSION X(1ØØ), H(2)
     ,DATA A/—6./,K/2/,PI/3.14159/,H/'1—Й␣'/'Y=␣␣'/
      DATA X/3Ø.,1Ø.,98*Ø./
      Y=SIN(PI*X(1)/18Ø.)+A/X(K)
      PRINT 1Ø, H, Y
1Ø    FORMAT ('␣',2A4, F5.2)
      END
```

напечатается

```
 1—Й␣Y=␣␣—Ø.1Ø
```

Начальные значения можно еще задавать в операторах описания типа. В приведенном примере можно написать вместо первых трех операторов

```
REAL X(1ØØ)/3Ø.,1Ø.,98./,A/—6./
REAL PI/3.14159/,H(2)/'1—Й␣','Y=␣␣'/
INTEGER K/2/
```

С помощью DATA и операторов описания типа нельзя присваивать начальные значения переменным, находящимся в области COMMON. Для этого существует специальная стандартная подпрограмма BLOCK DATA, которая используется редко и подробнее здесь не рассматривается.

5.8. Обработка программ на Фортране в ОС ЕС ЭВМ

Операционная система ОС ЕС ориентирована на пакетный метод обработки заданий в многопрограммном режиме. Многопрограммный режим состоит в том, что в память ЭВМ одновременно могут быть загружены несколько программ разных пользователей, которые работают одновременно. На самом деле процессор в каждый момент выполняет одну за другой команды единственной программы. Но как только в выполняемой программе встречается, например, более медленная операция ввода-вывода, процессор не ждет, когда эта операция завершится, а переходит к выполнению программы другого пользователя, пока в ней не встретится ввод-вывод, тогда он вернется к первой программе с прерванного места и т. д. Программы и данные для обработки подготовлены заранее, объединены в

пакет, а операционная система сама распределяет для них место в памяти ЭВМ, места на дисках под временные файлы, организует очередь на выполнение программ, компонует вместе результаты, получаемые по каждой программе, и т. п.

Система ОС ЕС предоставляет пользователю разнообразные услуги. Полное ее освоение требует изучения довольно большого объема материала, например [28], и значительного опыта в области программирования. В данном пособии ставится цель дать общее представление о порядке обработки Фортран-программы на ЭВМ под управлением ОС и научить читателя составлять несложные задания. Программисту, начинающему работать в ОС, можно рекомендовать также [29].

Порядок решения задачи на ЭВМ под управлением ОС. Решение любой задачи на ЭВМ можно рассматривать как обработку файлов или наборов данных по терминологии ОС. Набором данных являются перфокарты с текстом программы на Фортране, перфокарты с исходными данными, текст, отпечатанный на АЦПУ, файл на МЛ или МД и т. п.

Пользователь ОС ЕС, решая задачу на ЭВМ, должен определить *задание*. Задание состоит из *пунктов задания*, выполняемых друг за другом. В каждом из них описываются используемая программа и наборы данных, с которыми программа работает.

Программа, написанная на Фортране (или другом алгоритмическом языке), называется *исходным модулем*. Операторы исходного модуля должны быть переведены в команды машинного языка, после чего возможен счет по программе. Этот процесс организуется несколькими этапами, каждому из которых соответствует пункт задания.

Первым пунктом осуществляется трансляция исходного модуля и получение *объектного модуля*. Этот модуль содержит последовательность машинных команд, но в него не включены стандартные подпрограммы Фортрана, например, необходимые пользователю функции, подпрограммы ввода-вывода и т. п. Подпрограммы FUNCTION и SUBROUTINE, написанные пользователем, на этом этапе также транслируются в объектные модули, но связь между ними и основной программой еще не налажена. Объектные модули включают в себя дополнительную информацию (словари внешних имен, точек входа и т. п.), необходимую для организации правильных связей между программными единицами. После трансляции объектные модули помещаются в специальные наборы данных — библиотеки, временные или постоянные, по указанию пользователя. При трансляции в другой набор данных (обычно на печать) выводятся сообщения транслятора: текст программы, обнаруженные ошибки, таблицы имен и другая информация.

Второй пункт называется редактированием, в результате которого вызываются из библиотеки Фортрана недостающие объектные модули, организуется передача параметров и формируется *выпол-*

нимый загрузочный модуль, который обычно также помещается в библиотеку.

Редактирование выполняется специальной системной программой — Редактором Связей, или просто Редактором.

Третьим пунктом задания является выполнение созданного загрузочного модуля, т. е. готовой программы пользователя в машинных кодах.

Таким образом, для выполнения Фортран-программы необходимо составить для ОС задание из трех пунктов.

1. Вызвать транслятор с Фортрана. Наборы данных: входной с исходным модулем; выходные — сообщения транслятора, с объектным модулем.

2. Вызвать Редактор связей. Наборы данных: входные — объектные модули пользователя, библиотека подпрограмм; выходные — сообщения Редактора, загрузочный модуль.

3. Вызвать созданный загрузочный модуль.

Наборы данных: входной — данные пользователя; выходной — результаты работы программы пользователя.

По существу, задание представляет собой некоторую программу работы ОС. Для описания заданий разработан специальный Язык Управления Заданиями (ЯУЗ) — английская аббревиатура JCL (Job Control Language).

Операторы ЯУЗ. Существуют восемь операторов ЯУЗ, имеющих форму:

```
//ИМЯ␣JOB␣ОПЕРАНД␣[КОММЕНТАРИЙ]
//[ИМЯ]␣EXEC␣ОПЕРАНД␣[КОММЕНТАРИЙ]
//[ИМЯ]␣DD␣ОПЕРАНД␣[КОММЕНТАРИЙ]
//ИМЯ␣PROC␣ОПЕРАНД␣[КОММЕНТАРИЙ]
//ИМЯ␣PEND␣ОПЕРАНД␣[КОММЕНТАРИЙ]
//   <пустой оператор>
/*  <ограничительный оператор>
//* КОММЕНТАРИЙ
```

Операторы ЯУЗ, так же как и другие данные, готовятся на перфокартах. Символы «//» или «/*» должны занимать колонки 1,2. Имя оператора, если оно необходимо, набивается, начиная с колонки 3, т. е. без пробела после // В местах, обозначенных ␣, обязательны один или несколько пробелов.

Имя оператора ЯУЗ содержит от 1 до 8 символов — латинских букв или цифр, начинается с буквы. К буквам приравнены также символы @ (*коммерческое «at»*), # («решетка»), ¤ (денежная единица). Операнд состоит из последовательности параметров, разделенных запятыми. Пробелы внутри операндов не разрешаются. Комментарий к оператору отделяется от операнда пробелами и может занимать поле до 71-й колонки включительно.

Если оператор не помещается на одной карте, его можно продолжить на других, также начинающихся с // Операнд на каждой

карте нужно закончить запятой, а на картах продолжения писать его, начиная от 4-й колонки по 16-ю. В каждой карте продолжения может быть свой комментарий. Для продолжения комментария на следующей карте можно поставить любой символ в колонке 72 или использовать отдельную карту-комментарий (//* в колонках 1 — 3). Колонки 73 — 80 могут содержать нумерацию или идентификацию карт и влияния на работу ОС не оказывают.

Параметры в операндах бывают *позиционные*, смысл которых определяется порядком записи их значений в операнде, и *ключевые*, форма записи которых: ключевое-слово = значение.

В отдельных параметрах могут использоваться подпараметры, также разделяемые запятыми и заключаемые в скобки. Если в параметре используется только один подпараметр, скобки можно не писать. Подпараметры также могут быть позиционные и ключевые. Порядок записи ключевых параметров и подпараметров может быть любым.

Начинающему программисту, работающему на Фортране, не приходится пользоваться операторами PROC и PEND, поэтому они далее не рассматриваются.

Использование процедур. Как отмечалось выше, процесс обработки Фортран-программы описан тремя пунктами задания. Очевидно, разные пользователи должны были бы составлять похожие задания на ЯУЗ для решения своих задач. ОС представляет возможность избежать повторения однообразной работы по составлению заданий. Для этого в ее составе имеется *библиотека процедур*, в которой хранятся типовые программы на ЯУЗ (процедуры).

Рассмотрим пример составления задания для трансляции, редактирования и выполнения Фортран-программы, подготовленной на перфокартах. Исходные данные также подготовлены на картах. Обращения к МЛ и МД в программе не предусмотрены.

```
//EXAMPLE1 JOB 'K—11—1','КУЛИКОВ КАФ.ЭНС',
//  MSGLEVEL=2,TIME=5
//  EXEC FORTGCLG,PARM.LKED='NOMAP',REGION.GO=60K
//FORT.SYSIN DD *
```

карты исходных модулей основной программы и подпрограмм пользователя

```
//GO.SYSIN DD *
```

карты с данными для программы пользователя

```
//
```

Первым оператором задания всегда является оператор JOB, который не входит в библиотечную процедуру. Имя оператора (EXAMPLE1) выбирается произвольно, никакого влияния на выполнение задания оно не оказывает, а используется для распознавания данных, выводимых на печать, и для контроля за ходом выполнения заданий со стороны персонала ЭВМ.

Наиболее употребительные параметры оператора следующие:

учетная информация ('К—11—1') — первый позиционный параметр. Вид этого параметра и его конкретное содержание устанавливаются особо в каждом ВЦ;

идентификация программиста — второй позиционный параметр, который может содержать до 20 символов, как правило, заключается в апострофы, чтобы можно было применять любые символы. Остальные параметры оператора JOB — ключевые;

MSGLEVEL=... — указывает режим распечатки управляющих операторов и сообщений системы о распределении устройств под наборы данных (сокращение от level of messages — уровень сообщений). Запись MSGLEVEL=2 означает, что обеспечивается печать всех операторов ЯУЗ, содержащихся в задании. Этот параметр можно опустить, тогда устанавливается режим распечатки, принятый в данном ВЦ «по умолчанию»;

TIME=m, где m — предельное время обработки задания процессором в минутах. Если m=1440, — время не ограничено. Ограничение времени целесообразно устанавливать для того, чтобы не расходовать много машинного времени при зацикливании программы. Если параметр TIME= опустить, устанавливается принятое на данном ВЦ ограничение «по умолчанию». При решении больших задач, когда известно заранее, что нормальная работа программы требует больше времени, предусмотренного по умолчанию, TIME= надо указывать обязательно. Для небольших программ обычно достаточно 2 — 5 мин.

Остальные параметры, которые можно употреблять в JOB, для начинающего программиста не представляют интереса. Вид оператора JOB надо хотя бы раз согласовать с персоналом ВЦ, так как могут требоваться некоторые обязательные параметры.

Второй оператор задания обеспечивает выполнение (EXEC — сокращение от execute — выполнять) стандартной процедуры FORTGCLG.

Имя оператора, являющееся именем пункта задания, в примере опущено, оно необязательно.

Первым позиционным параметром оператора EXEC является имя процедуры (или программы, тогда записывается PGM=имя-программы). Имя процедуры надо выяснить в ВЦ, так как могут использоваться разные модификации процедур для работы с Фортраном.

Процедура FORTGCLG содержит три пункта (шага), имеющие имена: FORT — трансляция исходного модуля, LKED — редактирование, GO — выполнение. В ключевых параметрах оператора EXEC можно изменить стандартные режимы (опции) и другие параметры, записанные в операторах EXEC каждого шага процедуры. Опции — это режимы работы системных программ (транслятора, Редактора), которые задаются ключевым параметром PARM=... В данном примере PARM.LKED='NOMAP' означает, что при редактировании (шаг LKED указывается через точку после параметра PARM) требуется не печатать таблицу расположения

объектных модулей, включенных в загрузочный. Полный перечень опций системных программ приводится в специальной литературе [27,28,29].

Начинающий программист может не менять стандартные опции, заложенные в процедуре.

Параметр REGION.GO=6∅K указывает, что для выполнения шага GO, т. е. готовой Фортран-программы, необходимо 60 Кбайт оперативной памяти. Этого объема обычно достаточно для программ, содержащих до 100 операторов Фортрана и массивы до 2 — 4 тыс. чисел. Параметр REGION для шага GO чаще всего указывают постольку, поскольку запрос объема памяти по умолчанию может быть чрезмерным, чем замедляется обработка заданий в ОС.

Оператор, стоящий третьим в задании, описывает набор входных данных для шага FORT. Наборы обязательно описываются операторами DD (от data definition — определение данных). Имя оператора — составное, указывает, что описание относится к набору с dd-именем SYSIN в пункте процедуры FORT.

Имя оператора DD (dd-имя) — это имя, по которому к набору обращается программа. В операнде оператора DD должны быть описаны физическое устройство, а также физические характеристики набора данных. Этим средством ОС позволяет по одной и той же программе обрабатывать данные, размещенные на разнообразных носителях. В данном примере операнд «*» означает, что набор данных расположен на картах, следующих за картой DD.

Конец набора распознается по следующему оператору ЯУЗ, т. е. среди данных типа «*» не должно быть карт с «//» или «/*» в колонках 1,2.

Оператор //GO.SYSIN DD* описывает входной набор данных для выполнения готовой программы в виде карт, следующих за этим оператором. Пустой оператор//указывает на конец задания.

Приведенный пример задания может служить образцом для разового счета по программе, составленной на Фортране. Каталогизация отлаженных программ, т. е. запись загрузочных модулей в библиотеку, позволяет проводить массовые расчеты без затрат времени на трансляцию и редактирование. В данном пособии приемы каталогизации не рассматриваются.

Наборы данных на МЛ и МД. Для того чтобы обрабатывать с помощью Фортран-программ данные на МЛ и МД, необходимо научиться описывать наборы данных (НД) с помощью оператора DD.

Например dd-имя, к которому обращаются программы, транслированные в ОС с Фортрана, имеет вид FT*kk*F*nn*. Здесь *kk*—число от ∅1 до 99, равное номеру файла *k*, употребляемому пользователем в операторах READ/WRITE; *nnn* — трехзначное число от 001 до 999 — порядковый номер набора, он применяется для обращения к разным НД по одному номеру файла (при обработке магнитных лент).

Например, dd-имена следующих стандартных файлов:

FT∅5F∅∅1 — системный ввод;
FT∅6F∅∅1 — вывод на печать;
FT∅7F∅∅1 — вывод на перфорацию.

В ОС используются четыре способа организации НД, из которых программисту на Фортране достаточно иметь представление о последовательном и библиотечном наборах. Последовательные НД используются как для фортрановских файлов последовательного доступа на МЛ и МД, так и для файлов прямого доступа на МД. Записи в последовательном НД организованные по схеме рис. 5.2, называют неблокированными.

В операторе DD должно содержаться описание логической записи НД. Под логической записью понимается совокупность данных, обрабатываемых за одно обращение по оператору READ (WRITE) бесформатного ввода-вывода или между разделителями «/» формата. Иными словами, логическая запись — это запись «с точки зрения» программы пользователя. В ОС различают следующие типы (форматы) записей: фиксированной F, переменной V и неопределенной U длины. Формат записей (record format) описывается ключевым подпараметром RECFM=... параметра DCB=... (data control block — блок управления данными). Длина логических записей (logical record length) указывается в байтах подпараметром LRECL=... , причем для записей формата V указывается максимальная длина +4 байта вспомогательной информации, необходимой для ОС. Например, для набора данных на перфокартах параметр DCB должен выглядеть так: DCB=(RECFM=F,LRECL=8∅, BLKSIZE=8∅).

В параметре указан также подпараметр BLKSIZE=...(block-size — размер блока). ОС не требует, чтобы физическая запись, т. е. длина записи на носителе между промежутками, была равна логической записи, как это имеет место, например, на перфокартах. При использовании МЛ и МД записи выгодно блокировать, т. е. в одной физической записи размещать несколько логических. Блокирование указывается добавлением буквы B к значению RECFM=... Если, например, для МД записать

DCB=(RECFM=FB,LRECL=8∅,BLKSIZE=728∅)

то на 1-й дорожке диска ЕС-5061 будет размещаться информация с 91 перфокарты; если же записи определить неблокированными, на 1 дорожке размещаются только 40 перфокарт, так как промежутки между записями тоже занимают место.

С точки зрения программы пользователя совершенно безразлично, блокированы или не блокированы записи: операционная система сама «поставляет» по одной логической записи. Фактически вся физическая запись вначале считывается в специально отведенную область памяти (буфер), а по каждому оператору READ операционная система передает очередную логическую запись. Обратно при

записи буфер заполняется постепенно каждым оператором WRITE, а затем целиком записывается. Последний блок может быть неполным.

Записи переменной длины также позволяют экономить магнитные носители. Если, например, в программе есть фрагменты, которые повторяются в цикле 1000 раз

```
      DIMENSION X(5∅∅)
      . .
1∅    WRITE(9)A,B,C,D
      . .
4∅    WRITE(9) X
```

то при использовании записей типа F нужно было указать LRECL = 2000, чтобы там мог поместиться массив X. Но при выполнении оператора 1∅ полезной информацией заполнились бы только первые 16 байтов. Во время работы программы около 2 Мбайтов носителя пропало бы впустую (примерно 1/10 емкости пакета дисков).

Записи переменной длины тоже можно блокировать (RECEM = VB), а можно, наоборот, сегментировать (RECFM=VS), т. е. одну логическую запись размещать в нескольких физических (например, если логическая запись не размещается на одной дорожке диска) Файлы последовательного доступа Фортрана при бесформатной передаче должны описываться как блокированные сегментированные. При этом DCB=(RECFM=VSB,LRECL=n,BLKSIZE=m). Здесь $m \leqslant 32\,760$ для магнитных лент и не более емкости одной дорожки для дисков ($m \leqslant 7296$ у EC=5061); $n = m - 4$ обеспечит наилучшее использование пространства носителя.

Чрезмерно большие блоки требуют увеличения памяти под буфер и, кроме того, последний блок может заполниться частично, но займет много места на носителе.

Записи неопределенной длины бывают только неблокированными, при этом LRECL не задается: DCB=(RECFM=U, BLKSIZE=m), где m — определяется типом физического устройства. Записи типа U используются только при форматном вводе-выводе.

Для файлов прямого доступа НД на диске описывается следующим образом:

$$\text{DCB}=(\text{RECFM}=\text{F},\text{LRECL}=l,\text{BLKSIZE}=l)$$

Здесь l— длина записи в операторе DEFINE FILE (в байтах).

Во всех приведенных выше примерах опущено описание организации НД— подпараметр DSORG=...(data set organization) параметра DCB. Этот подпараметр по умолчанию предполагается DSORG=PS— физически последовательный (physically sequential).

Библиотечная организация НД возможна только на дисках и состоит в том, что каждый из ряда последовательных наборов (разделов) снабжается самостоятельным именем. Ряд объединяется под общим именем библиотеки. Имя набора (data set name) указывается

параметром DSN=BIBL(MEM), где BIBL — имя библиотеки, MEM — имя раздела.

Библиотечная организация описывается подпараметром DSORG= =PO (partitioned—поделенный на части). Фортран-программа обрабатывает раздел библиотечного набора, как файл последовательного доступа, но при этом в операнде оператора DD должен присутствовать параметр LABEL=(,,,IN) для вводных файлов и LABEL= (,,,OUT) для выводных.

Наборы на магнитных носителях обязательно снабжаются именами. Имена НД никак не связаны с dd-именами, а являются как бы этикетками физической совокупности данных. Это позволяет по одной и той же программе обрабатывать разные наборы, заменяя только карты DD в задании, а не переписывая заново операторы READ (WRITE) и не перетранслируя Фортран-программу.

Имена наборов, создаваемых самой системой, назначаются ею, и программист может их не знать. Имя набора в простейшем случае — от 1 до 8 символов, букв или цифр, начинающихся с буквы. Имя может быть составным, т. е. составленным из нескольких имен, разделенных точками. Если пользователь хочет создать НД на МЛ или МД, он должен согласовать с администрацией ВЦ серийный номер тома, на котором он будет размещать набор, и имя набора. ВЦ, обслуживающие много пользователей, обычно устанавливают особую систему составных имен, чтобы легко было определить, какой группе пользователей (лаборатории, кафедре и т. п.) данный набор принадлежит. Составное имя вместе с точками не может содержать более 44 символов.

Если впереди имени набора поставить два амперсанда, система рассматривает такой набор как временный и удаляет его после выполнения задания или пункта задания.

Серийный номер тома содержит шесть букв или цифр и присваивается каждому конкретному пакету дисков или катушке магнитной ленты (тому — volume). Он описывается подпараметром SER=... параметра VOL=... Физический тип устройства описывается параметром UNIT=...

Наконец, для НД обязательно описывается его диспозиция (DISP=...), т. е. состояние набора в начале пункта задания (NEW—набор создается, OLD—существует и др.), и указание, что сделать с ним после выполнения (DELETE—удалить, KEEP—сохранить и др.).

Операторы DD могут добавляться к любому шагу процедуры, а также замещать стандартные описания НД в процедурах.

Рассмотрим пример задания на трансляцию, редактирование и выполнение Фортран-программы, использующей, помимо системного ввода (файл № 5 или SYSIN) и вывода на печать, также файлы № 8, 9, 10 на магнитных носителях. Файл № 8— прямого доступа, в программе описан оператором

```
DEFINE FILE 8 (1000, 30, U, KEY)
```

Он используется для хранения промежуточных результатов самой программой. Файл № 9 на МД содержит ранее созданные данные. В файл № 10 программа выводит новые результаты на МЛ

```
//EXAMPLE2␣JOB␣'K-11-1', 'КУЛИКОВ', TIME=10
//␣EXEC␣FORTGCLG, REGION. GO=1000 K
//FORT.SYSIN␣DD␣UNIT=SYSDA, VOL=SER=ENS001,
//␣DISP=OLD, DSN=PROGRAM
//GO. SYSIN␣DD␣*
```

<карты с исходными данными>

```
//GO. FT0␣F001␣DD␣DSN=PROM. NABOR, UNIT=SYSDA,
//␣DISP=(NEW, DELETE), DCB=(RECFM=F, LRECL=120,
//␣BLKSIZE=1200), SPACE=(120, (1000))
//GO. FT09F001␣DD␣UNIT=SYSDA, VOL=SER=ENS001,
//␣DISP=OLD, DSN=DANN(VAR4), LABEL=(,,,IN)
//GO.FT10F001␣DD␣UNIT=5010, VOL=SER=K11EE3,
//␣DISP=(NEW, KEEP), DSN=RESULT3, LABEL=5,
//␣DCB=(RECFM=VSB, LRECL=796, BLKSIZE=800)
//
```

Здесь предполагается, что для выполнения программы потребуется 100 Кбайт памяти. Исходный модуль подготовлен не на картах, а заранее записан в последовательный набор PROGRAM на пакете дисков (SYSDA означает накопители на МД) с серийным номером ENS001. Поскольку НД создан ранее (DISP=OLD), параметр DCB для него не указывается, а ОС берет всю необходимую информацию из самого набора.

Исходные данные подготовлены на картах вместе с заданием. Для файла № 8 набор будет создан при выполнении программы и уничтожен после. Серийный номер тома не указан, ОС сама найдет свободное место для НД. Параметр SPACE=... (пространство) обязательно указывается при создании новых НД на дисках. В данном случае он предписывает отвести 1000 блоков по 120 байтов.

В файл № 9 считываются данные из раздела VAR4 ранее созданного библиотечного НД DANN, расположенного на томе ENS001. Вывод в файл № 10 создает новый набор на томе K11EE3 МЛ (тип устройства — ЕС-5010); НД снабжается именем RESULT 3, он размещается пятым на томе (LABEL=5). После окончания счета НД RESULT 3 будет сохранен.

Отметим, что описатель DISP=OLD вовсе не запрещает производить запись в такой НД. При создании НД выполняются следующие операции: отводится место на томе; записываются метки набора — его имя, параметры DCB и другая информация. После этого набор может быть как выходным, так и входным.

5.9. Отладка программ

При разработке программ возможны различные ошибки, которые можно подразделить на следующие основные группы: а) синтаксические ошибки в написании операторов, неверное употребление меток и имен (например, имени массива без индексов в арифметическом выражении), а также ошибки при перфорации программы; б) ошибки программирования, реализующие разработанный алгоритм неадекватно; в) ошибки при составлении алгоритма, не обеспечивающего решение поставленной задачи.

Приемы отладки программ на Бейсике в общих чертах применимы к программам на Фортране.

Большинство ошибок группы «а» легко обнаруживается на стадии трансляции программы. Все синтаксические ошибки, неверные метки, имена и т. д. транслятор отмечает на распечатке (листинге) программы и указывает характер ошибки (на англ. языке). В [27] и документации, имеющейся на ВЦ, дается расшифровка этих сообщений по номеру, которым снабжается каждый вид сообщения. Ошибки при перфорации, не вызывающие нарушения синтаксиса (например, пробивка другого символа, создающая новое имя), транслятором не отмечаются, но легко обнаруживаются при внимательном просмотре листинга.

Для обнаружения ошибок групп «б» и «в» необходимо бывает проследить ход вычислительного процесса по шагам или отдельным блокам. Операционные системы ДОС и ОС не имеют стандартных средств для пошагового визуального контроля работы программ, поэтому здесь приходится на этапе отладки заранее предусматривать печать промежуточных результатов.

Одним из наиболее неприятных следствий ошибок в алгоритме или его реализации является «зацикливание» программы, т. е. бесконечное повторение какого-либо цикла. Зацикливание может наступить, например, при выполнении следующего фрагмента для расчета таблицы функции $y=ax^2+bx+c$

```
      X=0.
   15 Y=A*X**2+B*X+C
      PRINT 100, X, Y
100 FORMAT ('␣X=', F6.2,'␣␣Y=', E12.5)
      X=X+0.05
      IF(X.NE.10.) GOTO 15
```

Казалось бы, после 200 повторений цикла переменная X примет значение X=10, условие оператора IF будет не выполнено, и цикл должен закончиться. Однако из-за неточного представления вещественных чисел в ЭВМ после 200 витков на самом деле может оказаться X=9.99999 или 10.00001, и в этом случае продолжается повторение цикла.

При зацикливании программы ее выполнение в ОС прекращается благодаря ограничению времени на выполнение задания. Однако, если внутри цикла предусмотрена печать, вывод результатов потребует большого количества бумаги и времени на вывод. Для предотвращения этого можно в задании на трансляцию, редактирование и выполнение программы с помощью процедуры FORTGCLG (см. параграф 5.8) перед картой // GOSYSIN␣DD... поставить карту //GO.FT∅6F∅∅1␣DD␣SYSOUT=A,OUTLIM=*m*. Здесь *m* — максимальное число строк, которое допускается напечатать.

Некоторые ошибки в программах приводят к недопустимым операциям, например, делению на нуль, извлечению квадратного корня из отрицательного числа и т. п. При возникновении таких ситуаций также печатается нумерованное сообщение, которое можно расшифровать, используя [27] или документацию ВЦ. Часть этих ошибок вообще не приводит к прекращению, пока ошибка не повторится несколько (обычно 10) раз. Количество допустимых повторений ошибок устанавливается при постановке (генерации) ОС в конкретном ВЦ.

Некоторые ошибки (чаще всего нарушения границ индексов или обращение к несуществующим подпрограммам) вызывают немедленное прекращение шага задания.

Для выявления места в программе, где происходят ошибки, проще всего вставить операторы печати в узловых точках алгоритма с выводом значений наиболее важных переменных и поясняющего текста.

В Фортране ЕС ЭВМ предусмотрены также специальные операторы отладки, которые дают возможность получить более полную информацию о ходе выполнения программы.

Операторы отладки. Эти операторы вместе с другими исполнимыми операторами Фортрана, помещенными среди них, образуют пакеты отладки, которые действуют в пределах одной программной единицы. К операторам отладки относятся: DEBUG («выявить дефекты»), AT («у», «около»), TRACE ON («включить трассировку»), TRACE OFF (выключить трассировку»), DISPLAY («вывести содержимое»).

Неисполнимый оператор DEBUG устанавливает режимы отладки и записывается один раз перед первым пакетом отладки. Режимы указываются в произвольном порядке, записываются вслед за словом DEBUG и отделяются друг от друга запятыми. Могут указываться следующие режимы:

UNIT (*k*)— указывает, что результаты отладки должны выводиться в файл № *k*. Если параметр опущен, вывод идет на системную печать (файл № 6 в ОС);

TRACE— выполнять трассировку, т. е. печатать метки операторов в процессе их выполнения. Участок программы, подлежащий трассировке, указывается операторами TRACE ON, TRACE OFF При их отсутствии трассировка не выполняется;

SUBTRACE— печатать момент входа в подпрограмму, содержащую DEBUG SUBTRACE в виде «SUBTRACE ИМЯ», где ИМЯ— имя подпрограммы. В момент выхода печатается «SUBTRACE RETURN»;

SUBCHK (x_1, x_2, ..., x_n — контроль индексов («subscript check») массивов с именами x_1, x_2, ..., x_n. Если при обращении к элементу какого-либо массива x_j порядковый номер элемента в памяти i выходит за его границы, указанные в DIMENSION или при описании типа, печатается «SUBCHK $x_j(i)$». Если список имен опущен, проверяются все массивы данной программной единицы;

INIT (x_1, x_2, ..., x_n) — контролировать присвоение («initiation») переменным или элементам массивов x_1, x_2, ..., x_n новых значений. При каждой операции присваивания печатается «ИМЯ = значение» или «ИМЯ (индексы) = значение» — для элемента массива. Если список опущен, контролируются все переменные и массивы программной единицы.

Неисполнимый оператор AT m определяет начало пакета отладки, который будет выполняться перед выполнением оператора с меткой m основной программы. Следом за AT m можно записать как отладочные операторы TRACE ON, TRACE OFF, DISPLAY, так и прочие операторы Фортрана, кроме операторов описания. Конец пакета отладки определяется следующим оператором AT ... или END.

Операторы TRACE ON, TRACE OFF обеспечивают трассировку, начиная с метки m_1, указанной в AT m_1 того пакета, где записан TRACE ON. Трассировка выполняется до тех пор, пока программа не дойдет до метки m_2, указанной в другом пакете AT m_2, содержащем TRACE OFF. При проходе каждого помеченного оператора с меткой m_j печатается «TRACE m_j». Трассировка работает, если в операторе DEBUG указан режим TRACE. Оператор DISPLAY x_1, x_2, x_n эквивалентен действию операторов

```
NAMELIST /n/x1, x2, ..., xn
WRITE (k, n)
```

т. е. обеспечивает печать переменных и массивов x_1, x_2, x_n при выполнении пакета отладки, в котором он записан.

Структура программной единицы на Фортране. С учетом использования пакетов отладки последовательность операторов в программной единице должна быть следующей:

оператор определения подпрограммы FUNCTION, SUBROUTINE или BLOCK DATA;
оператор IMPLICIT;
операторы описания, отличные от IMPLICIT (например, REAL, INTEGER, DIMENSION, COMMON и т. д.);
операторы-функции;
исполнимые операторы;
оператор DEBUG;
операторы пакетов отладки;
оператор END.

Операторы FORMAT, DATA, NAMELIST, DEFINE FILE могут находиться в любом месте после оператора IMPLICIT. Операторы NAMELIST и DEFINE FILE должны логически предшествовать соответствующим операторам ввода-вывода.

В пакетах отладки метки не должны совпадать с метками основной программы; циклы DO должны заканчиваться в пределах одного пакета и не допускаются операторы описания. Нельзя передавать управление из основной программы в пакет отладки и из одного пакета в другой, но из пакета отладки можно осуществлять переход к любому исполнимому оператору основной программы.

По окончании отладки все пакеты и оператор DEBUG надо удалить, так как их наличие значительно удлиняет и замедляет программу.

Ответы к упражнениям 1—17

1. б, д, е. **2.** а, б, е, и. **3.** целые: б, д, з, л; вещественные: а, в, е, к; недопустимые: г, ж, и. **4.** д— недопустимы комплексные операнды в операции отношения; з— логическая переменная D не может использоваться в арифметическом выражении; нельзя ставить подряд два знака арифметической операции, надо A*(—B). 5. 81.0, 5.0, 1.0, —12.0,—3.0.

6. Один из возможных вариантов:

```
REAL L, I1, I2, I3

TOKB=(I1*X1+I2*H2+I3*X3)/L
TOKA=I1+I2+I3-TOKB
U1=TOKA*X1*R
U2=U1+(TOKA-I1)*(X2-X1)*R
U3=TOKB*(L-X3)*R
```

7. Возможные варианты:

```
а) IF (TOK) 1, 2, 2
1 U=-A*TOK**2
GOTO 3
2 U=A*TOK**2
3    (продолжение программы)
б) U=A*TOK**2
IF(TOK.LT.0.)U=-U

в) U=SIGN(A*TOK**2, TOK)
```

8. Возможные варианты:

```
а) TOK=0.
   IF(U.GT.0.)TOK=U/R
б) TOK=DIM(U,0.)/R
```

9. Один из вариантов:

```
X=AMOD(T,100.)  (отбрасывается целое число периодов)
U=0.
IF (X.GT.20..AND.X.LT.30..OR.X.GT.60..AND.X.LT.70.)U=50.
```

10.

```
DIMENSION N(2), T(2, 10)

    F=0.
    DO 1 J=1,2
    K=N(J)
    DO 1 I=1,K
1   F=F+T(J, I)
```

11.
```
      DIMENSION F(2Ø), V(2Ø)
С ОПРЕДЕЛЕНИЕ БЛИЖАЙШИХ УЗЛОВ ТАБЛИЦЫ
      DO 1Ø I=1,19
      IF (VØ.LT.V(I+1)) GOTO 11
   1Ø CONTINUE
(печать сообщения «VØ =... БОЛЬШЕ МАКСИМАЛЬНОЙ»)
      STOP
11 FØ=F(I)+(F(I+1)-F(I))*(VØ-V(I))/(V(I+1)-V(I))
```

12. Карты занумерованы 1), 2) и т. д.

Вариант а):

```
1)  ␣␣␣␣3␣␣␣11␣␣␣12␣␣␣13␣␣␣14␣␣␣151␣␣2␣␣
                                        3␣␣4␣␣
2) ␣␣5␣␣␣␣6␣␣␣␣7␣␣8␣␣
3) ␣␣11␣␣12␣␣13␣␣14␣␣15␣␣16
```

Вариант б):

```
1) 3 (в 1-й колонке)
2) 1112131415
3) ␣␣1␣␣␣2␣␣␣3␣␣␣4␣
4) ␣␣5␣␣␣6␣␣␣7␣␣␣8␣
5) ␣11␣␣12␣␣13␣␣14␣
6) ␣15␣␣16
```

Остальные символы на картах могут быть любыми.

13.
```
      PRINT 9Ø, VØ
   9Ø FORMAT'(␣VØ =',F5.1,'␣БОЛЬШЕ МАКСИМАЛЬНОЙ')
```

14.
```
      7Ø PRINT 71, V, K
   71 FORMAT('␣V=',F4.Ø,␣␣КОЭЭФИЦИЕНТ СЦЕПЛЕНИЯ=',F6.4)
```

15.
```
      COMPLEX ZKP1, ZTK, ZKT
      AL (X)=Ø.Ø628*ALOG (X)
      (ввод RK, RT, DKP, DKT, RRK, RRT)
      ZKP1=CMRLX(RK, AL(1.28*DKP/RRK))
      ZTK=CMPLX(RT, AL(1.28*DKT/RRT))
      ZKT=CMPLX (RK, AL(1.28*DKT/RRK))
```

16.
```
      COMPLEX ZKP1, ZTK, ZKT, Z
      Z(R, D, RR)=CMPLX(R, Ø.Ø628*ALOG(1.28*D/RR))
(ввод RK, RT, DKP, DKT, RRK, RRT)
      ZKP1=Z(RK, DKP, RRK)
      ZTK=Z(RT, DKT, RRT)
      ZKT=Z (RK, DKT, RRK)
```

17. Один из вариантов:

```
      SUBROUTINE SFI (U, I, S, CFI)
      COMPLEX U, I, S
      S=U*CONJG (I)
CFI=REAL(S)/CABS(S)
      RETURN
      END
```

Контрольные вопросы

1. Какие типы данных используются в Фортране и какими способами описываются они в программе?
2. Какие различают виды выражений и каков приоритет операций в них?
3. Перечислите операторы управления программой и объясните их назначение.
4. Какими способами можно организовать циклические программы на Фортране?

5. Как организуются данные в файлах последовательного доступа?

6. Приведите конструкции операторов ввода-вывода последовательного доступа по формату и без него.

7. Объясните назначение и правила записи оператора FORMAT.

8. Какими форматными кодами можно воспользоваться для ввода-вывода данных вещественного типа стандартной и двойной точности?

9. Как осуществляется взаимодействие операторов ввода-вывода с оператором FORMAT?

10. Какими достоинствами обладает ввод-вывод прямого доступа и для каких носителей информации он применим?

11. В чем заключается различие в употреблении подпрограмм FUNCTION и SUBROUTINE?

12. Что такое формальные и фактические параметры подпрограмм, в каких сочетаниях их можно употреблять?

13. Как используются общие области данных COMMON при программировании задач с подпрограммами?

14. Приведите структуру задания на выполнение программы, написанной на Фортране, в ОС ЕС ЭВМ.

Глава 6

ИМИТАЦИОННЫЕ МОДЕЛИ ТЯГОВОГО ЭЛЕКТРОСНАБЖЕНИЯ И ЭЛЕКТРОПОДВИЖНОГО СОСТАВА

6.1. Структуризация объекта моделирования

Построение аналитической модели такого сложного объекта, как электрифицированный участок железной дороги, как отмечалось в гл. 1, практически невозможно. Это определяется сложностью его структуры и функционирования. В этих случаях сложная система, в данном случае система электрифицированной железной дороги, должна быть разбита на ряд подсистем с сохранением существующих между ними связей. Каждая из полученных таким образом подсистем может оказаться достаточно сложной системой. В таких случаях разбиение продолжается до тех пор, пока не будут получены простые подсистемы, функционирование которых может быть описано аналитическими зависимостями. Такие конечные подсистемы называются элементами сложной системы [1] В процессе расчленения системы должны сохраняться все внешние и внутренние связи между подсистемами всех уровней.

Структура электрифицированного участка содержит входящие в нее подсистемы *4, 5, 6* (рис. 6.1), которые являются сложными системами. Так, система отдельной единицы электроподвижного состава *5* может быть разбита на более мелкие, но все же сложные подсистемы (рис. 6.2).

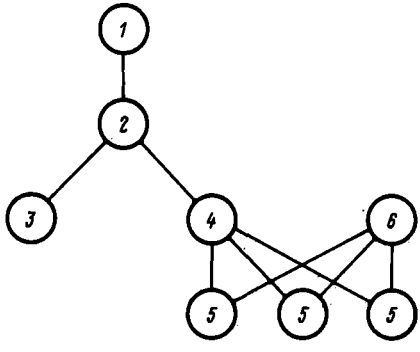


Рис. 6.1. Структурная схема электрифицированного участка:
1— внешняя часть электроснабжения; *2*— система электроснабжения электрифицированного участка; *3* — нетяговая часть системы электроснабжения; *4*— тяговая часть электроснабжения; *5*— тяговые потребители (локомотивы); *6*— система организации движения

В настоящее время более полно разработана структура системы тягового электроснабжения (рис. 6.3).

Структура, положенная в основу имитационной модели тягового электроснабжения, была создана в МИИТе в 50-х годах и в связи со слабым в то время развитием универсальных ЭВМ была реализована только на специализированном устройстве.

В дальнейшем во ВЗИИТе и других организациях она стала реализовываться как на универсальных ЭВМ, так и на специальных гибридных комплексах.

Структура имитационной модели включает следующие основные блоки: блок формирования графика движения поездов, блок перемещения и формирования нагрузок, блоки тяговых подстанций и тяговой сети. В наиболее полной структуре (принятой, в частности, в 50-е годы) включаются блоки внешней части электроснабжения.

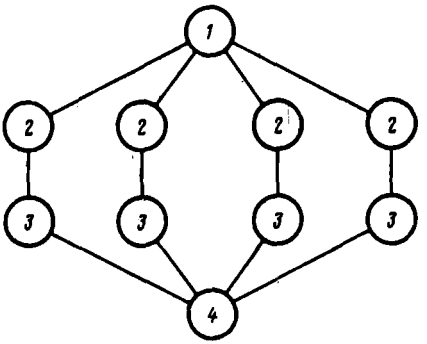


Рис. 6.2. Структурная схема электровоза: *1*— система преобразования энергии; *2*— система тяговых двигателей; *3*— системы колесных пар; *4*— система взаимодействия колес с рельсами

Принятая структура отражает все основные функциональные связи (рис. 6.4), имеющие место на реальном электрифицированном участке.

Наиболее полно, особенно для дорог переменного тока, воспроизводятся они при реализации имитационной модели на аналого-цифровом комплексе. Практически только на таком комплексе возможно адекватное воспроизведение режимов работы преобразователей электровозов.

Использование имитационной модели для целей проектирования в условиях эксплуатации снимает вопрос о точности расчетов, так как с ее помощью можно учесть практически все влияющие на выбор параметров устройств электроснабжения факторы и получить результаты, точность которых будет зависеть только от точности задания исходных данных. К исходным данным, в частности, относятся размеры движения и массы поездов.

Большая часть экономически обоснованных параметров системы электроснабжения является функционалами множества реализаций случайных нагрузок устройств электроснабжения, в основном зависящих от режимов движения поездов.

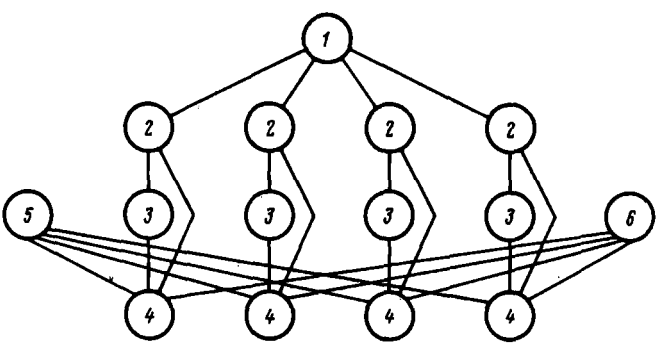


Рис. 6.3. Структурная схема системы электроснабжения: *1*— система внешнего электроснабжения; *2*— системы тяговых подстанций; *3*— системы тяговых сетей; *4*— системы э. п. с.; *5*— система управления движением; *6*— система «план и профиль пути»

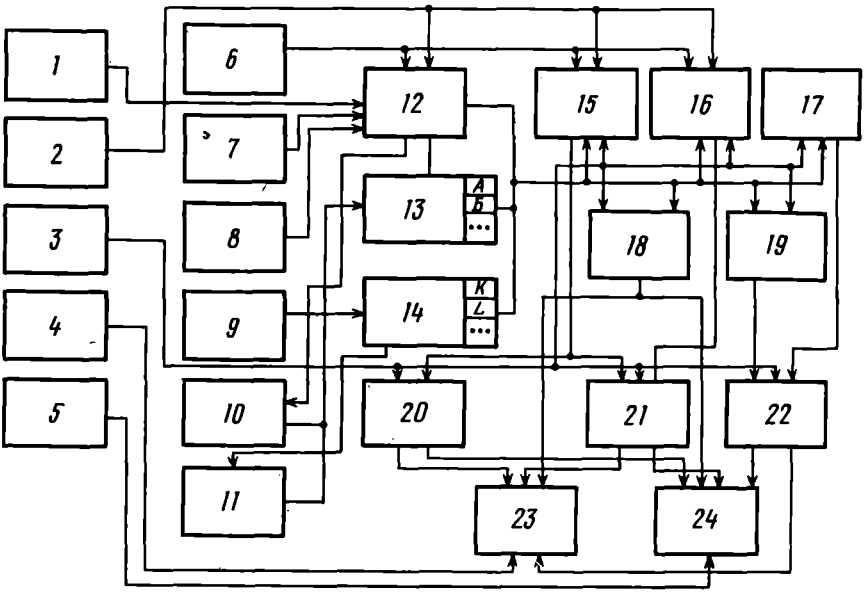


Рис. 6.4. Структура информационной программной реализации имитационной модели: *1*— механические параметры и характеристика поезда; *2*— климатологические сведения; *3*— параметры оборудования; *4*— укрупненные стоимостные показатели; *5*— нормы; *6*— район проектирования; *7*— тип локомотива, масса поезда; *8*— топографические данные; *9*— данные по организации движения; *10*—результаты тяговых расчетов; *11*—графики движения; *12*—тяговый расчет; *13*—формирование и решение мгновенной схемы; *14*—составление расчетного графика движения; *15*— расчет старения изоляции трансформатора; *16*— расчет температуры *p-n*-переходов выпрямителей; *17*— расчет потерь энергии в тяговой сети; *18*— расчет критерия качества напряжения; *19*— проверка проводов контактной сети по нагреву; *20*—определение мощности трансформаторов; *21*—определение мощности выпрямителей; *22*—определение сечения контактной сети; *23*—определение критериев экономической эффективности; *24*—проверка технических норм качества функционирования

Таким образом, показатели и работоспособность большинства устройств электроснабжения зависят не от отдельных экстремальных режимов, а от характера изменения их нагрузки.

6.2. Агрегатирование системы тягового электроснабжения

Рассмотрим более подробно имитационную модель тягового электроснабжения. Начнем с агрегатирования этой сложной системы [1] Следует отметить, что разбиение системы на агрегаты неоднозначно. Поэтому от принятых на этапе агрегатирования решений зависит как сложность имитационной модели, так и адекватность ее модулируемой системе. В связи с этим чрезвычайно важно четко определить, какие свойства системы являются основными для воспроизведения изучаемых процессов и какие второсте-

пенными. От такого разделения будут зависеть как выделение тех или иных агрегатов, так и требования к преобразованию информации о них. Это вытекает из указанного ранее положения, что структура всякой, в том числе и имитационной модели, в значительной мере определяется теми задачами, для решения которых она предназначена.

В этой связи следует прежде всего учесть, что как при эксплуатационных, так и при проектных расчетах необходимо выполнять расчеты ряда показателей, по которым проверяются или выбираются параметры устройств электроснабжения. В данном случае важно, что в своем большинстве каждый показатель является результатом преобразования (часто очень сложного) случайной реализации процесса изменения нагрузки устройств.

Из этого следует, что при проектировании системы необходимо предусмотреть агрегат (агрегаты), формирующий процесс изменения нагрузки с учетом влияния на него случайных факторов. Этот агрегат и соответствующий ему модуль модели будут обеспечивать последовательность в системном времени значений и расположений нагрузки в соответствии с рассматриваемой реализацией графика движения поездов. Далее правильность решения задач будет определяться учетом тех параметров и характеристик, которые существенно влияют на токораспределение, а следовательно, на потери мощности, потери напряжения, нагрузки трансформаторов, преобразовательных установок, устройств поперечной компенсации и др.

Параметры тяговой сети переменного тока весьма существенно сказываются на форме кривых токов отдельных локомотивов и их влиянии друг на друга. В конечном итоге это влияние определяет коэффициент мощности электрической энергии на вводах подстанции и у районных потребителей. Из сказанного следует, что агрегаты, на которые будет расчленена собственно система электроснабжения, должны отражать влияние на токораспределение и форму токов параметров тяговых подстанций и тяговой сети, а агрегаты, формирующие тяговую нагрузку, обеспечивать такие функциональные преобразования, которые воспроизведут на выходе реальную форму кривой тока, если на их входах будут напряжение тяговой сети и выпрямленный ток.

В полной мере это можно выполнить практически только при реализации модели на гибридной ЭВМ. Моделируемую систему целесообразно разбить на статическую подсистему устройств электроснабжения и динамическую подсистему перемещающихся по участку и изменяющихся во времени по заданному закону тяговых нагрузок. Статическая подсистема должна быть представлена агрегатами, обеспечивающими функциональные преобразования входной информации внешней части электроснабжения, тяговыми подстанциями, тяговой сетью, а подвижная подсистема — агрегатами, выполняющими функции перемещения и изменения нагрузок от ЭПС.

В соответствии с [1] каждый агрегат в момент времени t $(0, T)$ находится в одном из возможных состояний, которое при изменении аргумента t в интервале $(0, T)$ изменяется.

В отличие от структурной схемы схема электроснабжения, представленная как агрегативная система (рис. 6.5), определяет функциональные связи, обусловливающие взаимную зависимость между подпроцессами и назначение каждого агрегата как преобразователя входной информации.

При использовании для моделирования первой — статической части системы аналоговых устройств многие функциональные преобразования входной информации происходят, естественно, по тем же законам, что и в натуре.

Агрегативная система включает следующие агрегаты (см. рис. 6.5): *АВЭ* — агрегат, функционирование которого отражает состояние внешней системы электроснабжения; *АТП1; АТП2; АТП3* ... — агрегаты, отражающие состояние тяговых подстанций; *АТС* — агрегаты, отражающие состояние тяговой сети; *АН1, АН2, АН3*... — агрегаты, состояние которых определяет нагрузку тяговой сети; *АI*(t)— агрегат, формирующий токи локомотива; *АU* — агрегат, формирующий координаты локомотивов; *АГД*— агрегат, отображающий расположение поездов в соответствии с графиком движения. Агрегативная система может быть разбита на две подсистемы: на агрегативную подсистему электроснабжения и на агрегативную подсистему формирования значений и координат нагрузок. К первой части относятся агрегаты *АВЭ, АТП, АТС;* ко второй — *АГД*.

Агрегаты *АВЭ, АТП, АГД, АI* (t) по терминологии, введенной Н. П. Бусленко [1], являются входными полюсами, получающими информацию $x_{вэ}$, $x_{тп}$, $x_{гд}$, x_I. Символами $x_{вэ}$ обозначена информация о напряжениях источников питания в моделируемой части системы внешнего электроснабжения; $x_{тп}$— информация о нагрузках нетяговых потребителей и о температуре окружающей среды; $x_{гд}$ — информация о размерах движения, расположении станций, минимальных межпоездных интервалах и плотности вероятности их.

Входной информацией для агрегата $AS(t)$ являются x_I либо зависимости тока поезда и пройденного им пути, либо (при учете влияния напряжения в тяговой сети на режим работы локомотива) данные о плане и профиле пути. Во втором случае агрегат $AS(t)$ преобразует под действием управляющих сообщений внешнюю и внутреннюю информацию (пунктир от агрегата AU к агрегату $AS(t)$) поочередно в координаты токов поездов в данный момент. В рассматриваемом случае учитывается влияние напряжения на график движения, что показано пунктирной линией между агрегатами $AS(t)$ и *АГД*.

Управляющее воздействие g_1 позволяет в каждый момент времени поочередно выбирать из агрегата $AS(t)$ координаты локомотивов и

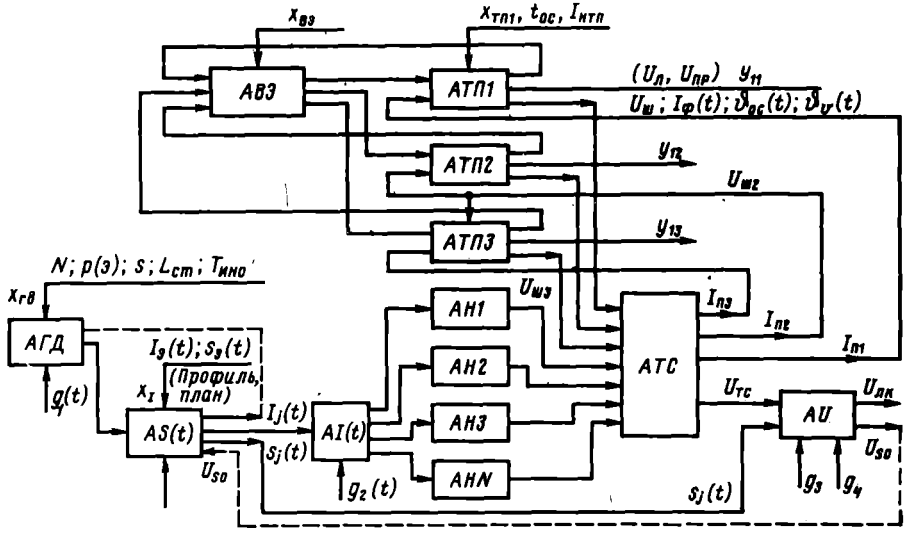


Рис. 6.5. Агрегативная система схемы электроснабжения

потребляемые ими токи. Воздействие $g_2(t)$ обусловливает передачу по каналам, соответствующим координатам поезда, информации о токах локомотивов блокам *АН*.

Агрегаты *АН* передают информацию об этих нагрузках агрегату *АТС* (нагружают тяговую сеть). Кроме этого, на вход агрегата *АТС* даются сообщения о напряжениях на шинах тяговых подстанций ($U_{ш1}$, $U_{ш2}$, $U_{ш3}$, ...). Для дорог переменного тока от каждой подстанции подается сообщение о двух напряжениях (два плеча).

Функцией агрегата *АТС* является преобразование входной информации в информацию, передаваемую агрегатам *АТП* (распределение нагрузок между подстанциями) и агрегату *AU* (сообщение о распределении напряжения вдоль тяговой сети).

Следует обратить внимание на то, что выходная информация блоков *АТП* — напряжения на шинах подстанций является результатом преобразования как информации, подступающей от агрегата *АВЭ*, так и от агрегата *АТС* (нагрузок подстанций). Таким образом, агрегаты *АТП* и *АТС* охвачены обратной связью.

Управляющее воздействие g_3 обеспечивает выходную информацию о напряжении при движении рассматриваемого локомотива, а воздействие g_4 — поочередную передачу сообщений о напряжении у каждого локомотива агрегату $AS\ (t)$ в случае, если он выполняет функции преобразования входной информации в ток и координату его с учетом напряжения в тяговой сети.

Агрегат *AU*, так же как и агрегаты *АТП*, является выходным полюсом. Выходной информацией агрегатов *АТП* (y_{11}, y_{12}, ...) являются нагрузки обмоток трансформаторов, выпрямителей (для под-

станций постоянного тока), превышение температуры обмоток тягового трансформатора над окружающей средой $\vartheta_{oc}(t)$, температура р-n-переходов вентилей тяговых преобразователей $\vartheta_{v}(t)$, токи тяговых фидеров $I_{ф}(t)$.

Схема агрегатирования (см. рис. 6.5) охватывает наиболее полно реальный объект, включая электроподвижной состав. Моделирующий алгоритм для такой схемы может быть составлен довольно просто. Однако программная реализация его очень сложна и пока реализована только для упрощенных исходных данных.

Для решения большинства практических задач агрегативную систему можно существенно упростить, если не учитывать влияние изменения напряжения тяговой сети в местах расположения локомотивов на режимы их работы.

При таком допущении, как сказано выше, для агрегата $AS(t)$ входной информацией будут только данные о типах и времени отправления поездов, получаемые от агрегата *АГД*.

Для построения математической модели всей системы необходимо произвести моделирование процесса функционирования каждого агрегата, т. е. получить значения, определяющие состояние агрегата, а следовательно, и сигналов на его выходе в зависимости от системного времени на интервале исследования $(0, T)$ и последовательности сигналов, поступающих на его вход.

Заметим, что функциональные преобразования, выполняемые всеми агрегатами, кроме агрегата *АГД*, известны из курса «Электрические железные дороги». По существу — это расчеты токораспределения между элементами системы тягового электроснабжения нагрузок в последовательно формируемых мгновенных схемах. Обработка таких расчетов позволяет определить нагрузки фидеров, подстанций, потери мощности в элементах системы, изменение температуры проводов и обмоток и т. д.

6.3. Общая схема моделирующего алгоритма

На имитационной модели непосредственно воспроизводятся все основные связи между отдельными устройствами реальной системы. Поэтому проведение расчета на такой модели эквивалентно эксперименту на реальном объекте. Имитационное моделирование позволяет воспроизводить работу системы электроснабжения как при различных случайных реализациях графика движения, так и при жестких заранее заданных графиках.

Как правило, выбор параметров устройств по экономическим критериям производится исходя из работы при всех возможных графиках, а проверки по техническим ограничениям ведутся по специальным графикам. В нашей стране накоплен опыт разработки имитационных моделей для электрифицированных железных дорог [2, 30, 31, 32, 33].

В Метрогипротрансе и на Московском метрополитене уже в течение длительного времени электрические расчеты электроснабжения метрополитена производятся при помощи имитационной модели [33].

Представление имитационной модели тяговой нагрузки как случайного или неслучайного процесса позволяет выполнить любые расчеты устройств электроснабжения, так как моделирует нагрузку всех элементов во времени.

Имитационное моделирование имеет ряд следующих преимуществ перед другими методами расчета:

оно позволяет легко реализовать модульную структуру программного обеспечения;

добавление новых задач не вызывает перестройки основных ее элементов, реализующих имитационную модель;

сама модель может строиться на разных уровнях точности (а следовательно, и сложности), поэтому дальнейшее ее развитие не сказывается на программном обеспечении частных задач.

Компоненты автоматизированной системы проектирования для решения задачи выбора оптимальных параметров системы электроснабжения определяются технологией выполнения электрических расчетов.

В качестве исходных данных в САПР-ЭЛ принимаются: размеры движения, классифицированные по типам поездов; профиль и план участка; характеристики локомотивов; веса поездов и типы локомотивов; места обязательных остановок и длительность стоянок поездов; возможные места расположения тяговых подстанций; стоимость внешнего электроснабжения и подъездного пути для каждого места расположения подстанции; данные о нагрузках нетяговых потребителей.

На основе этих исходных данных в первую очередь должны быть намечены варианты расположения подстанций, подлежащие рассмотрению.

На первом этапе эту работу целесообразно возложить на специалиста-проектировщика.

В дальнейшем формирование расчетных вариантов можно производить на ЭВМ.

Таким образом, технология выполнения электрических расчетов содержит следующие этапы:

формирование и ввод исходных данных (профиль и план, площадки для подстанции, стоимость внешних устройств для различных мест их расположения, веса поездов, размеры движения, расчетные режимы, требования к точности расчетов, параметры тяговых нагрузок и т. д.);

формирование расположения тяговых подстанций (в первой очереди САПР выполняется проектировщиком);

выполнение тяговых расчетов для всех режимов движения поездов;

формирование очередного расчетного режима движения в соответствии с действующими нормами и правилами;

формирование графика движения;

формирование и расчет мгновенных схем;

обработка результатов расчета мгновенных схем с целью получения значений величин, необходимых для выбора или проверки параметров устройств электроснабжения, и проверка выполнения технических норм;

проверка окончания периода (сутки или другой период, установленный дополнительно), рассматриваемого расчетного режима движения;

осуществляется проверка окончания просмотра очередного расчетного режима. Если очередной расчетный режим связан с вероятностными графиками движения, то расчеты повторяются до получения заданной точности при заданной надежности результатов. Для режимов движения по заданному жесткому графику расчет заканчивается одновременно с окончанием просмотра периода этого режима;

осуществляется проверка просмотра всех предписанных режимов движения;

расчет параметров устройств электроснабжения для рассмотренного варианта расположения тяговых подстанций, расчет защиты от т. к. з.;

фиксация основных показателей рассмотренного варианта;

проверка просмотра всех намеченных вариантов расположения тяговых подстанций;

окончание расчета и вывод на печать показателей по всем вариантам.

При реализации имитационной модели на ЭВМ необходимо учитывать ряд общих требований, предъявляемых к подобным системам:

структура и язык должны быть достаточно универсальными, дающими возможность совершенствования методов расчета некоторых параметров, а также максимальную независимость от модели ЭВМ;

должна быть обеспечена высокая живучесть, т. е. изменение типового оборудования, технических норм и требований не должно приводить к отказу системы;

реализованная модель должна обеспечивать возможность ее совершенствования и наращивания путем добавления новых задач и корректировки существующих;

должна быть налажена единая система хранения и обработки информации по типовому оборудованию, характеристикам и параметрам устройств, исходным данным для проектирования;

должна обеспечиваться выдача минимального количества промежуточных результатов, необходимых для принятия решений о дальнейшем ходе проектирования;

выдача результатов расчета должна производиться в виде легко читаемых результатов.

Решение отдельных задач реализуется набором программных модулей на общей информационной базе.

На первом этапе целесообразно автоматизировать расчеты основных параметров системы электроснабжения, определяющих технико-экономическую эффективность ее функционирования: выбор оптимального расположения и мощности подстанций, сечения проводов контактной сети. Эти задачи ориентированы на использование человеко-машинной системы, что и определяет структуру информационно-программного обеспечения первой очереди реализации имитационной модели.

Низшим уровнем программных модулей являются процедуры имитационной модели: тяговый расчет, составление расчетного графика движения, формирование и решение мгновенных схем. Последние две процедуры разрабатываются в различных вариантах для разных родов токов, схем организации движения и могут зависеть от решаемой задачи, поскольку требования к точности и быстродействию различны.

Информация для работы имитационной модели поступает из справочного и текущего фондов. На основе информации, получаемой с имитационной модели в процессе ее работы, решаются подзадачи, связанные с определением технико-экономических показателей отдельных устройств электроснабжения и системы в целом.

6.4. Моделирование нагрузок электроподвижного состава

Важнейшими подсистемами, определяющими нагрузку всех элементов тягового электроснабжения, являются электровозы и электропоезда.

Каждая единица э. п. с., как уже говорилось, представляет сложную динамическую систему.

При моделировании системы тягового электроснабжения можно не моделировать внутренние состояния отдельных модулей системы э. п. с., а воспроизводить только внешние характеристики его, определяющие взаимодействие с системой электроснабжения. Воздействие э. п. с. на систему электроснабжения определяется токами отдельных его единиц в любой момент времени и скоростями их движения. Взаимодействие э. п. с. и тягового электроснабжения обусловливается напряжением в контактной сети в местах расположения локомотивов.

Токи и скорости движения поездов, как известно, определяются тяговыми расчетами, которые в условиях эксплуатации уточняются при опытных поездках.

В принципе для электрических расчетов могут быть использованы методы выполнения тяговых расчетов, применяемые для

других целей. Однако существующие программы для использования их в электрических расчетах тягового электроснабжения требуют проведения ряда дополнительных преобразований. Особенно большие затруднения возникают при учете взаимодействия э. п. с. и системы электроснабжения, при котором желательно иметь непосредственную зависимость тока и скорости локомотива от подводимого к нему напряжения.

В связи с этим для модуля «тяговые расчеты» была разработана специальная методика и моделирующий алгоритм.

Методологическая база моделирующего алгоритма «тяговые расчеты» для э. п. с. переменного тока. При работе тяговых двигателей на расчетной (ходовой) характеристике зависимость силы тяги от тока может быть хорошо аппроксимирована уравнением

$$F = C_F \Phi I_d = a I_d - b, \tag{6.1}$$

где I_d — выпрямленный ток локомотива, приведенный к первичной стороне трансформатора;

Φ — магнитный поток тягового двигателя;

C_F — коэффициент, который, в частности, учитывает приведение тока к первичной стороне трансформатора;

a и b — коэффициенты аппроксимации, которые будут определены ниже.

Из выражения (6.1) найдем

$$\Phi = \frac{a I_d - b}{C_F I_d} \tag{6.2}$$

Электродвижущая сила тягового двигателя, приведенная к первичной стороне трансформатора, может быть представлена выражением

$$E = C_E \Phi v, \tag{6.3}$$

где значение коэффициента C_E дается с учетом приведения E к стороне высшего напряжения.

С учетом формул (6.1) и (6.3) можно написать

$$v = \frac{E I_d}{C(a I_d - b)}, \tag{6.4}$$

где $C = C_E / C_F$.

Формула (6.4) позволяет непосредственно связать приращение скорости с приращением тока

$$\Delta v = \frac{dv}{dI_d} \Delta I_d$$

или, учитывая выражения (6.3) и (6.4),

$$\Delta v = \left[\frac{-Eb}{(a I_d - b)^2} + \frac{I_d}{a I_d - b} \frac{dE}{dI_d} \right] \Delta I_d / C.$$

Если приведенное к выпрямленному напряжение в контактной сети равно U_-, то

$$E = U_- - z_{э-} I_d, \tag{6.5}$$

где $z_{э-}$ — приведенное к выпрямленному напряжению на первичной стороне трансформатора электровоза сопротивление силовой цепи.

Учитывая, что $\frac{dE}{dI_d} = -z_{э\_}$, получим

$$\Delta v \approx \frac{1}{C}\left[-\frac{bU_- - z_{э\_} I_d}{(aI_d - b)^2} - \frac{z_{э\_} I_d}{aI_d - b}\right]\Delta I_d$$

или

$$\Delta v = \frac{-1}{C(aI_d - b)^2}(bU_- + az_{э\_} I_d^2 - 2bz_{э\_} I_d)\,\Delta I_d \qquad (6.6)$$

При проходе поездом отрезка пути ΔS, км, совершается работа, Вт·с:

$$\Delta A = 9{,}81\, m(w + i_к)\,\Delta S \cdot 10^3 + (1+\gamma)\frac{v\Delta v}{3{,}6^2}\cdot m \cdot 10^3,$$

где m— масса поезда в т; w — основное удельное сопротивление движению, кгс/т; $i_к$— приведенный подъем, ‰; $1+\gamma$— коэффициент инерции вращающихся масс.

Первый член этой формулы равен работе, затраченной на преодоление сил сопротивления, а второй учитывает изменение кинетической энергии.

Если эта работа совершена за время Δt, то электрическая энергия будет равна $EI_d \Delta t$.

Из баланса механической и электрической энергии при $1+\gamma = 1{,}06$ получим:

$$\eta EI_d \Delta t = [9{,}81(w + i_к)\,\Delta S \cdot 10^3 + 81{,}8 v \Delta v]\, m,$$

где η— коэффициент полезного действия локомотива без учета электрических потерь.

Из последнего выражения можно найти

$$\left[\frac{\eta EI_d}{vm} - 2{,}725(w + i_к)\right]\Delta t = 81{,}8\,\Delta v. \qquad (6.7)$$

Заметим, что равенство нулю левой части уравнения (6.7) соответствует установившейся скорости, когда энергия, получаемая из сети, полностью расходуется на преодоление сил сопротивления движению. Следовательно, при

$$\frac{\eta EI_d}{vm} = 2{,}725(\omega + i_к) \qquad \Delta v = 0.$$

При подстановке в формулу (6.7) значений v из формулы (6.4) и Δv из выражения (6.6) после преобразований получим:

$$\Delta t = \frac{81{,}8(bU_- + az_{э\_} I_d^2 - 2bz_{э\_} I_d)\,\Delta I_d}{\left[2{,}725(w + i_к) - \frac{C\eta}{m}(aI_d - b)\right](aI_d - b)^2 C}. \qquad (6.8)$$

Для определенной позиции контроллера приведенную э. д. с. E можно представить в виде:

$$E = 0{,}9\,U_- - z_{э\_} I_d, \qquad (6.9)$$

где U_- — действующее напряжение холостого хода, приравниваемое к напряжению в контактной сети; $z_{э\_}$ — может быть найдено по внешней характеристике выпрямителя.

Из формул (6.4) и (6.8) вытекает, что равенство

$$2{,}725(w + i_к) = C\eta\frac{aI_d - b}{m} \qquad (6.10)$$

соответствует установившейся скорости, при которой

$$\Delta v = 0 \text{ и } \Delta I_d = 0.$$

Алгоритмы выполнения тяговых расчетов при различных режимах ведения поезда. Тяговый расчет после выхода электродвигателей на естественную характеристику можно проводить в следующем порядке:

по заданному ΔI_d и току в середине предыдущего шага находится ток в середине шага j, т. е. $I_{dj}=I_{dj-1}+\Delta I_d$;

по формуле (6.9) находится E_j;

по формуле (6.8) определяется Δt_j;

по формуле (6.4) определяется v_j;

находится приращение пути на шаге j:

$$\Delta S_j=v_j\Delta t_j/3600; \tag{6.11}$$

определяется путь, пройденный в конце шага j,

$$S_j=S_{j-1}+\Delta S_j; \tag{6.12}$$

делается проверка:

$$C\eta\frac{a\left(I_{dj}+\frac{\Delta I_d}{2}\right)-b}{m}>2{,}725\left(w_j+i_{kj}\right)>C\eta\frac{a\left(I_{dj}-\frac{\Delta I_d}{2}\right)-b}{m}$$

или

$$\Delta I_d>2{,}725\frac{m}{C\eta}\left(w_j+i_{kj}\right)+\frac{b}{a}-I_{dj}+\frac{\Delta I_d}{2}>0, \tag{6.13}$$

где i_{kj}— текущее значение подъема.

Если неравенства (6.13) выполняются, то до конца текущего подъема i_{kj} ток и скорость принимаются постоянными:

$$\left.\begin{aligned} I_{j+1}&=I_j;\\ v_{j+1}&=v_j\\ &\text{и}\\ \Delta S_{j+1}&=S_{ikm}-S_j;\\ \Delta t_{j+1}&=\frac{\Delta S_{j+1}}{v_j}3600, \end{aligned}\right\} \tag{6.14}$$

где S_{ikm} — координата текущего значения подъема i_{km}.

По этому алгоритму может выполняться тяговый расчет по ходовой характеристике.

В режиме пуска будем считать, что трогание и разгон поезда производятся при постоянной силе тяги и полном возбуждении тяговых двигателей.

Пусть сила тяги при разгоне будет равна $F_п$. Обозначим ток тягового двигателя через $I_д$. Положим, что в течение разгона сопротивление движению не меняется и равно среднему. Профиль также не меняется. Тогда ускорение поезда во время разгона будет постоянным

$$\frac{dv}{dt}=\frac{9{,}81\left[F_п-m\left(w_п+i_{кп}\right)\right]}{(1+\gamma)\,m\cdot10^3}$$

Последнее выражение можно записать в виде

$$\frac{dv}{dt}=\frac{9{,}81}{1+\gamma}\left(f_п-w_п-i_{кп}\right)10^{-3}\ \text{м/с}^2, \tag{6.15}$$

где $f_п=F_п/m$— удельная сила тяги при пуске, кгс/т.

В соответствии с формулой (6.15) скорость поезда при пуске при $1+\gamma=1{,}06$ составит:

$$v=33{,}32\left(f_п-w_п-i_{кп}\right)t\quad10^{-3}\ \text{км/ч}. \tag{6.16}$$

Так как ранее было принято, что ток тяговых двигателей не меняется, то без существенной погрешности можно положить также неизменными потери мощности в локомотиве ΔP. Тогда полезно используемая часть получаемой из сети мощности будет

$$P_{\text{пол}} = UI' - \Delta P,$$

где I'— активная составляющая тока, приведенная к напряжению контактной сети.

С другой стороны, ту же мощность можно найти как произведение силы тяги и скорости движения

$$P_{\text{пол}} = 9{,}81\, m f_{\text{п}} \frac{v}{3{,}6} = \frac{m f_{\text{п}} v}{0{,}367}.$$

Сравнивая два последних выражения, получим

$$UI' - \Delta P = \frac{m f_{\text{п}} v}{0{,}367}. \tag{6.17}$$

Подставляя сюда v из формулы (6.16), получим

$$t = 11 \frac{UI' - \Delta P}{m f_{\text{п}} (f_{\text{п}} - w_{\text{п}} - i_{\text{кп}})}. \tag{6.18}$$

Из рассмотрения тяговых характеристик электровозов переменного тока непосредственно вытекает, что в период пуска как активный ток, так и выпрямленный находятся в линейной зависимости от скорости. Следовательно, можно записать:

$$\begin{aligned} I' &= a_1 + b_1 v; \\ I_d &= a_2 + b_2 v. \end{aligned} \tag{6.19}$$

Из этих выражений можно получить

$$I' = a_1 - \frac{b_1 a_2}{b_2} + \frac{b_1}{b_2} I_d \tag{6.20}$$

Подставляя I' из формулы (6.20) в выражение (6.18), получим

$$t = 11 \frac{U\left(a_1 - \dfrac{b_1 a_2}{b_2} + \dfrac{b_1}{b_2} I_d\right) - \Delta P}{m f_{\text{п}} (f_{\text{п}} - w_{\text{п}} - i_{\text{кп}})}. \tag{6.21}$$

Отсюда, принимая $\Delta t = \dfrac{dt}{dI} \Delta I_d$,

$$\Delta t_i \approx 11 \frac{U \dfrac{b_1}{b_2}}{m f_{\text{п}} (f_{\text{п}} - w_{\text{п}} - i_{\text{кп}})} \Delta I_d. \tag{6.22}$$

Далее имеем

$$t_j = t_{j-1} + \Delta t_j; \tag{6.23}$$

$$v_j = 33{,}32 (f_{\text{п}} - w_{\text{п}} - i_{\text{кп}}) 10^{-3} t_j. \tag{6.24}$$

Алгоритм расчета при пуске состоит в последовательном использовании формул, начиная с формулы (6.22).

Рассмотрим режимы выбега и торможения. В общем случае режиму торможения до остановки может предшествовать режим выбега. Тогда скорость начала торможения должна быть задана исходя из тех или иных соображений. Время торможения и путь, пройденный поездом за это время, определяются обычными способами. Для этого необходимо выполнить следующие операции:

выбрать интервал изменения скорости Δv, в пределах которого замедляющее усилие принимается постоянным. Этот интервал должен быть кратен $v_т$;

определить число шагов при расчете времени торможения и пути, пройденного поездом за это время,

$$n_т = v_т/\Delta v_т; \tag{6.25}$$

определить для всех шагов от первого до $n_т$-го интервал времени, в течение которого скорость поезда изменяется на $\Delta v_т$,

$$\left.\begin{aligned} \Delta t_{jт} &= \frac{81{,}8\,\Delta v_т}{2{,}725\left(b_{эj} + w_j + i_{кj}\right)} \\ \text{или} \qquad \Delta t_j &= \frac{30\,\Delta v_т}{b_{эj}}, \end{aligned}\right\} \tag{6.26}$$

где $b_{эj} = b_{тj} + w_j + i_{кj}$ — удельное замедляющее усилие на шаге j.

определить для каждого шага средние скорости

$$v_{jт} = \left(j - \frac{1}{2}\right)\Delta v_т; \tag{6.27}$$

найти приращение пути на каждом шаге

$$\Delta s_{jт} = \frac{v_{jт}\,\Delta t_{jт}}{3600}; \tag{6.28}$$

найти общее время торможения

$$t_т = \sum_{1}^{n_т} \Delta t_{jт}; \tag{6.29}$$

найти общий путь, пройденный поездом при торможении

$$S_т = \sum_{1}^{n_т} \Delta s_{jт}. \tag{6.30}$$

Путь, проходимый поездом на выбеге, будем отсчитывать от точки начала торможения в сторону, противоположную направлению движения.

Этот путь и время выбега можно находить в следующем порядке, задавшись по соображениям точности расчетов интервалом изменения скорости $\Delta v_в$:

определяется средняя скорость движения поезда на шаге j:

$$v_{вj} = v_т + \left(j - \frac{1}{2}\right)\Delta v_в; \tag{6.31}$$

определяется интервал времени на шаге j:

$$\Delta t_{вj} = \frac{30\,\Delta v_в}{w_j + i_{кj}}; \tag{6.32}$$

определяется отрезок пути, пройденный поездом за время

$$\Delta s_{вj} = \frac{v_{вj}\,\Delta t_j}{3600}; \tag{6.33}$$

определяется путь, пройденный поездом на выбеге до начала торможения, и время, затраченное на прохождение этого пути,

$$S_{вk} = \sum_{1}^{k} \Delta s_{вj}; \tag{6.34}$$

$$t_{вk} = \sum_{1}^{k} \Delta t_{вj}. \tag{6.35}$$

Расчеты ведутся до достижения поездом максимально допустимой скорости (по ограничениям или конструкционной скорости локомотива).

Для дальнейших расчетов зависимости $S_в(v_в)$ и $t_в(v_в)$ аппроксимируются полиномами второй степени:

$$S_в = \alpha_{в0} + \alpha_{в1} v_в + \alpha_{в2} v^2_в \tag{6.36}$$

и

$$t_в = \beta_{в0} + \beta_{в1} v_в + \beta_{в2} v^2_в, \tag{6.37}$$

где α и β с соответствующими индексами — коэффициенты.

Обозначим наибольшие значения пути, пройденного поездом на выбеге при скорости начала выбега $v_в = v_{в\max}$, через $S_{в\max}$. Общий максимальный путь за время выбега и торможения

$$S_{вт} = S_т + S_{в\max}.$$

При проведении тягового расчета на каждом его шаге делается проверка

$$S_j < L - S_{вт}, \tag{6.38}$$

где L — длина участка.

Если это соотношение не выполняется, то переходят к проверке на всех последующих шагах тождества

$$S_j = L - S_т - (\alpha_{в0} + \alpha_{в1} v_в + \alpha_{в2} v^2_в). \tag{6.39}$$

После того как это тождество будет удовлетворено, расчет заканчивается.

Если выбег не предусматривается, то расчет ведется несколько в другом порядке: интервал изменения скоростей $\Delta v_т$ выбирается по условиям точности расчета; по формулам (6.26), (6.27), (6.28) ведется расчет $\Delta t_{jт}$, $v_{jт}$ и $\Delta S_{jт}$; на каждом шаге k находится время торможения и пройденный при торможении путь:

$$\left.\begin{aligned} t_{kт} &= \sum_{j=1}^{k} \Delta t_j; \\ S_{kт} &= \sum_{j=1}^{k} \Delta S_j; \end{aligned}\right\} \tag{6.40}$$

производится аппроксимация зависимостей $S_{кт}(v_т)$ и $t_{кт}(v_т)$ полиномами второй степени:

$$S_{кт} = \alpha_{т1} v_т + \alpha_{т2} v^2_т; \tag{6.41}$$

$$t_{кт} = \beta_{т1} v_т + \beta_{т2} v^2_т. \tag{6.42}$$

Тяговый расчет, после того как неравенство (6.38) перестает выполняться, ведется с проверкой тождества

$$S_j = L - \alpha_{т1} v_т + \alpha_{т2} v^2_т. \tag{6.43}$$

Шаг, на котором это тождество будет удовлетворено, является последним.

Здесь рассмотрены только основные принципы построения алгоритмов приведения тяговых расчетов при отдельных режимах движения поезда. Не рассмотрен режим движения по невредным спускам, на которых приходится чередовать режимы выбега и тяги.

При составлении полного алгоритма тяговых расчетов необходимо предусмотреть много логических операций, обеспечивающих соблюдение ограничений скорости движения на отдельных участках и сопряжение расчетных зависимостей на границах элементов профиля. Полный алгоритм и тем более программа тягового расчета в настоящем учебнике не рассматриваются.

6.5. Моделирование графика движения

Иногда при проектировании и особенно в эксплуатации требуется проверить заданные графики движения поездов. В таких случаях никаких особых исследований для создания модели графика движения не требуется. Остаются, однако, чисто технические и весьма существенные трудности программной реализации графика движения на ЭВМ.

Расчеты системы тягового электроснабжения базируются в основном на вероятностной основе. Обоснованный выбор или проверка параметров большинства устройств тягового электроснабжения возможны только путем реализаций случайных процессов нагрузки отдельных элементов системы. Это возможно только при моделировании ряда реализаций графиков движения поездов для конкретных условий работы рассматриваемого участка.

При известных (в результате выполнения тяговых расчетов или проведения опытных поездок) зависимостях пройденного пути от времени график движения поездов определяется порядком следования поездов и интервалами времени между их отправлениями.

В качестве исходных данных для моделирования графика движения должно быть задано число типов поездов по каждому пути, станции, на которых имеются стоянки поездов того или иного направления, заданное время стоянок, число поездов каждого типа. Время хода поездок различных типов по перегонам определяется по тяговым расчетам. Кроме того, должны быть особо отмечены поезда, для которых время стоянок нельзя менять. Предполагается, что для остальных поездов в случае необходимости время стоянок можно менять, но только в сторону увеличения.

Составление реализации графика движения производится в следующем порядке:

определение порядка отправления поездов разного типа с начальной станции;

определение моментов отправления поездов;

корректировка длительности стоянок некоторых поездов на последующих станциях для пропуска других поездов и определение моментов отправления поездов на этих станциях.

Будем в дальнейшем считать, что стоянка поезда «утверждена», если заданную длительность стоянки изменить нельзя. К «утвержденным» будем относить также длительности стоянок поездов после их окончательной корректировки.

Рассмотрим последовательно этапы составления алгоритма, моделирующего случайную реализацию графика движения.

При статистическом моделировании, в частности при моделировании случайных графиков движения поездов, приходится иметь дело со случайными числами.

Имеются специальные программы, которые позволяют генерировать случайные[1] значения числа, равномерно распределенные в интервале 0—1. Эта программа используется в первую очередь для определения порядка следования поездов разного типа.

Пусть общее число поездов, отправляемых за период T,

$$N = \sum_{g=1}^{V} N_g,$$

где g — номер типа поезда; N_g — число поездов типа g; V — число типов поездов.

Вероятность того, что первым отправляемым поездом будет поезд типа g, равна $P_{g1} = N_g/N$

Для M-го по порядку следования поезда вероятность принадлежности его к типу g будет:

$$P_{gM} = \frac{N_g - N_{gM}}{N - M}, \qquad (6.44)$$

где N_{aM}— число поездов типа g среди первых M поездов в реализации графика движения.

Для определения порядка следования поездов поступают следующим образом. Отрезок числовой сети от 0 до 1 делят на V частей, пропорциональных числу поездов каждого типа, с помощью программы генерирования случайных чисел и в зависимости от того, на какую часть числовой оси оно попало, назначают тип поезда. После этого числовую ось вновь делят на части в соответствии с оставшимся числом поездов, вновь используют программу для получения следующего случайного числа и определяют тип второго поезда. Эта процедура повторяется до тех пор, пока не будет установлен порядок отправления всех N поездов в рассматриваемой реализации графика движения. В результате этого создается массив очередности типов поездов, отправляемых с начальной станции.

После того как установлены типы поездов в порядке их отправления, можно найти минимальные межпоездные интервалы ТЕТА (рис. 6.6 и 6.7). Минимальные интервалы между поездами зависят от их типа.

За расчетный период T необходимо отправить N поездов. В начальный момент резервное время, оставшееся до отправления первого поезда, $T0 = T$ Это время после каждого отправления поезда уменьшается на межпоездной интервал ТЕТА.

---

[1] В действительности программы генерируют так называемые псевдослучайные числа, которые по своим числовым характеристикам хорошо имитируют случайное число, равномерно распределенное в интервале 0—1.

Отправление первого поезда совмещают с моментом начала времени моделирования. Организуются циклы по очередности отправления поездов. В цикле рассматриваются два поезда. Первый — уже условно «отправленный» в результате выполнения предыдущего цикла, и второй — «отправляемый». Определяются их типы по массиву очередности типов поездов, созданному в результате работы предыдущего блока. Далее находится станция, где есть остановка одного из рассматриваемых поездов, и определяется время хода каждого поезда до этой станции. После этого находится разность времени хода двух поездов Z— предыдущего и отправляемого. Если $Z \leqslant 0$— т. е. время хода отправляемого поезда больше времени хода предыдущего (см. рис. 6.6), то TETA = DM (DM — заданный минимальный интервал по отправлению).

Если $Z > 0$ (см. рис. 6.7), то TETA = DM + Z.

До расчета случайного интервала проверяется значение NG-- признака, определяющего тип графика. Если NG = 1, то расчет случайного интервала пропускается и время отправления поезда определяется по расчетному интервалу. В противном случае рассчитывается

$$\text{TETA} = [1 - (1 - Y)^{1/N_{\text{ост}}}] \qquad (6.45)$$

где TETA— случайный интервал; $N_{\text{ост}}$— число оставшихся неотправленными поездов; Y— равномерно распределенное случайное число.

Далее сравниваются TETA и DM (расчетный минимальный интервал), за фактический интервал выбирается больший. После этого резервное время уменьшается на фактический интервал, а число оставшихся поездов уменьшается на 1, и определяется время отправления очередного поезда путем суммирования фактического интервала и времени отправления предыдущего поезда. Циклы кончаются просмотром всех поездов. Создается массив моментов отправления поездов с начальной станции.

Далее сравниваются TETA и DM (расчетный минимальный интервал), за фактический интервал выбирается большой. После этого резервное время поездов определяют их типы. По типу поезда находят его время хода по первому перегону. Это время прибавляется к времени отправления поезда с начальной станции и результат заносится в массив времени прибытия этого поезда на вторую станцию. Создается массив характера стоянок. При утвержденной стоянке поезда элемент массива NUT (I, IC) приравнивается нулю, если на станции у поезда есть стоянка, и единице — если остановки нет. Стоянка поезда считается «утвержденной», если у поезда нет остановки или длительность ее не требует корректировки.

По времени прибытия и заданной длительности стоянки поезда заполняется массив заданного времени отправления поезда OTP (I, IC).

Заметим, что члены массива будут всегда возрастать с ростом индекса I только на начальной станции. На остальных станциях такое положение может нарушиться вследствие неодинакового времени стоянок поездов разных номеров.

Для корректировки первоначально заданных стоянок, а следовательно, и значений членов массива OTP организуется циклический перебор поездов в порядке их прибытия с целью отыскания очередного поезда с «неутвержденной» стоянкой.

В начале цикла задается заведомо большое значение времени прибытия TP некоторого фиктивного поезда. На первом шаге с этим временем сравнивается время прибытия поездов по массиву OTP. Если стоянка очередного поезда «утверждена», то просмотр продолжается. Так как начальное значение TP очень велико, то при начальном просмотре будет найден некоторый поезд с «неутвержденной» стоянкой: Далее TP присваивается значение времени прибытия этого поезда и если время прибытия очередного поезда оказалось больше TP или стоянка этого поезда на этой станции «утверждена», то этот поезд исключается из рассмотрения. Если у поезда есть стоянка и его время прибытия меньше TP, то очередному поезду № I, стоянка которого требует корректировки, присваивается номер шага цикла, соответствующий номеру поезда, присвоенного на начальной станции, а TP присваивается время прибытия этого поезда. По окончании цикла определяется номер поезда L = I, прибывшего раньше других поездов с «неутвержденной» стоянкой.

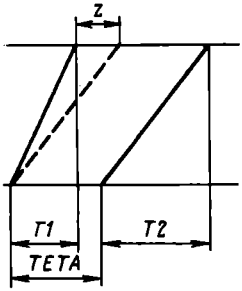


Рис. 6.6. Определение межпоездного интервала в случае большей скорости 1-го поезда

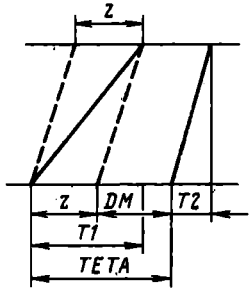


Рис. 6.7. Определение межпоездного интервала в случае меньшей скорости 1-го поезда

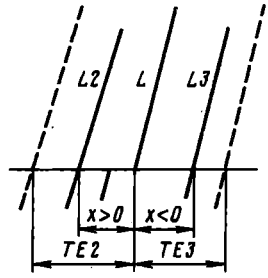


Рис. 6.8. Поиск ближайших по времени отправления поездов

Необходимость корректировки времени стоянки этого поезда определяется путем рассмотрения соседних поездов с «утвержденными» стоянками. Для этого отыскивается поезд L2 (рис. 6.8), предыдущий по отношению к рассматриваемому, и последующий L3. До начала цикла интервалам между рассматриваемым поездом, предыдущим TE2 и последующим TE3 присваиваются большие числа.

В циклах рассматриваются только поезда с «утвержденной» стоянкой. Вновь организуется цикл для определения интервалов между рассматриваемым и другими поездами. Далее на каждом шаге определяются интервалы x по отправлению между поездом L и другими поездами с «утвержденными» стоянками. Если $x<0$, т. е. поезд пришел позже поезда L, то x сравнивается с интервалом TE3, определенным на предыдущих шагах. Если $x<TE3$, то за последующий поезд L3 принимается текущий номер поезда, а интервал TE3 заменяется интервалом TE2. В противном случае рассматриваемый поезд пропускается, а TE3 и L3 не переопределяются. Такая же процедура выполняется и при $x>0$ (поезд текущего номера пришел позже), когда x сравнивается с интервалом по отправлению поездов L и ближайшего предыдущего из рассмотренных поездов. После перебора всех поездов с «утвержденной» стоянкой остаются два соседних поезда с номерами L2 и L3, ближайшие по отправлению.

При проверке отправления поезда L по отношению к L2 корректируется время стоянки поезда по отношению к предыдущему поезду. Если поезд L пришел первым на станцию (что соответствует условию $L2=0$), то его время по отношению к L2 не корректируется.

Сначала корректируется время отправления L по интервалу отправления TE2, чтобы выполнялось условие $TE2 \geqslant DM$ (рис. 6.9). Затем определяется тип обоих поездов. По типу поездов определяется ближайшая станция, где у одного из рассматриваемых поездов есть остановка, определяется время хода до этой станции и рассчитывается отклонение от минимально допустимого интервала прибытия этих поездов на данную станцию DELTA1:

$$DELTA1=(OT1+T1)-(OT2+T2)-DM.$$

Если $DELTA1 \geqslant 0$, что означает, что скорость поезда L меньше, интервал по прибытии больше, чем DM, и дополнительная корректировка времени отправления L не требуется. Если $DELTA1<0$ (рис. 6.10), значит, скорость поезда L больше и интервал по прибытии на следующую станцию, где у одного из поездов есть остановка, недопустимо мал или более того — возможна встреча поездов на перегоне, то необходимо увеличить время отправления поезда L на DM—T1, чтобы итервал по прибытии был равен DM. Поеле этого проверяется необходимость корректировки времени отправления рассматриваемого поезда по отношению к последующему поезду L3.

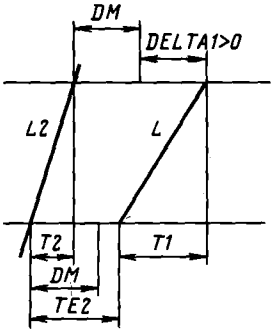


Рис. 6.9. Корректировка времени отправления поездов в случае большей скорости предыдущего поезда

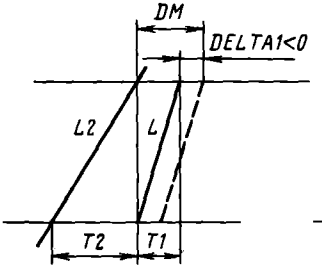


Рис. 6.10. Корректировка времени отправления поездов в случае меньшей скорости предыдущего поезда

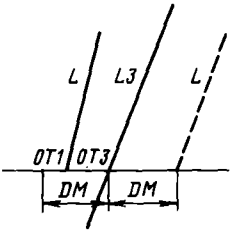


Рис. 6.11. К расчету межпоездного интервала при стоянке одного из поездов

Если отсутствует поезд L3, то стоянка рассматриваемого поезда сразу «утверждается». В противном случае проверяется интервал по отправлению с рассматриваемой станции между поездами L и L3 (рис. 6.11). Если этот интервал меньше минимально допустимого, то время отправления будет OT1 = OT3 + DM. В этом случае производится возврат к отысканию соседних поездов без дальнейшей корректировки, так как нарушилась очередность отправления поездов.

Если же интервал по отправлению допустим, то, как и в блоке корректировки по отношению к L2, определяется следующая ближайшая станция, где у одного из рассматриваемых поездов есть остановка, определяется время хода поездов до этой станции и рассчитывается отклонение DELTA2 (рис. 6.12) от минимального интервала DM. DELTA2 = (OT3 + T3) − (OT1 + T1) − DM.

Если DELTA2 < 0, корректируется время отправления рассматриваемого поезда (рис. 6.13) OT = OT3 + DM и возвращаются к операции отыскивания соседних поездов из-за нарушения очередности отправления.

Если DELTA2 ⩾ 0, то окончательно определяется время стоянки поезда и члену массива TST(L, I, C, K) присваивается значение фактического времени стоянки поезда номера L на станции IC, пути K:

$$TST(L, IC, K) = OT1 - PB(L, IC)$$

где PB — массив времени прибытия поездов.

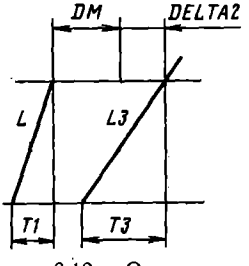


Рис. 6.12. Определение времени отправления поезда в случае меньшей скорости следующего поезда

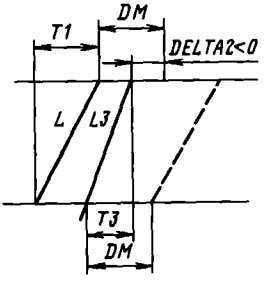


Рис. 6.13. Определение времени отправления поезда в случае большей скорости следующего поезда

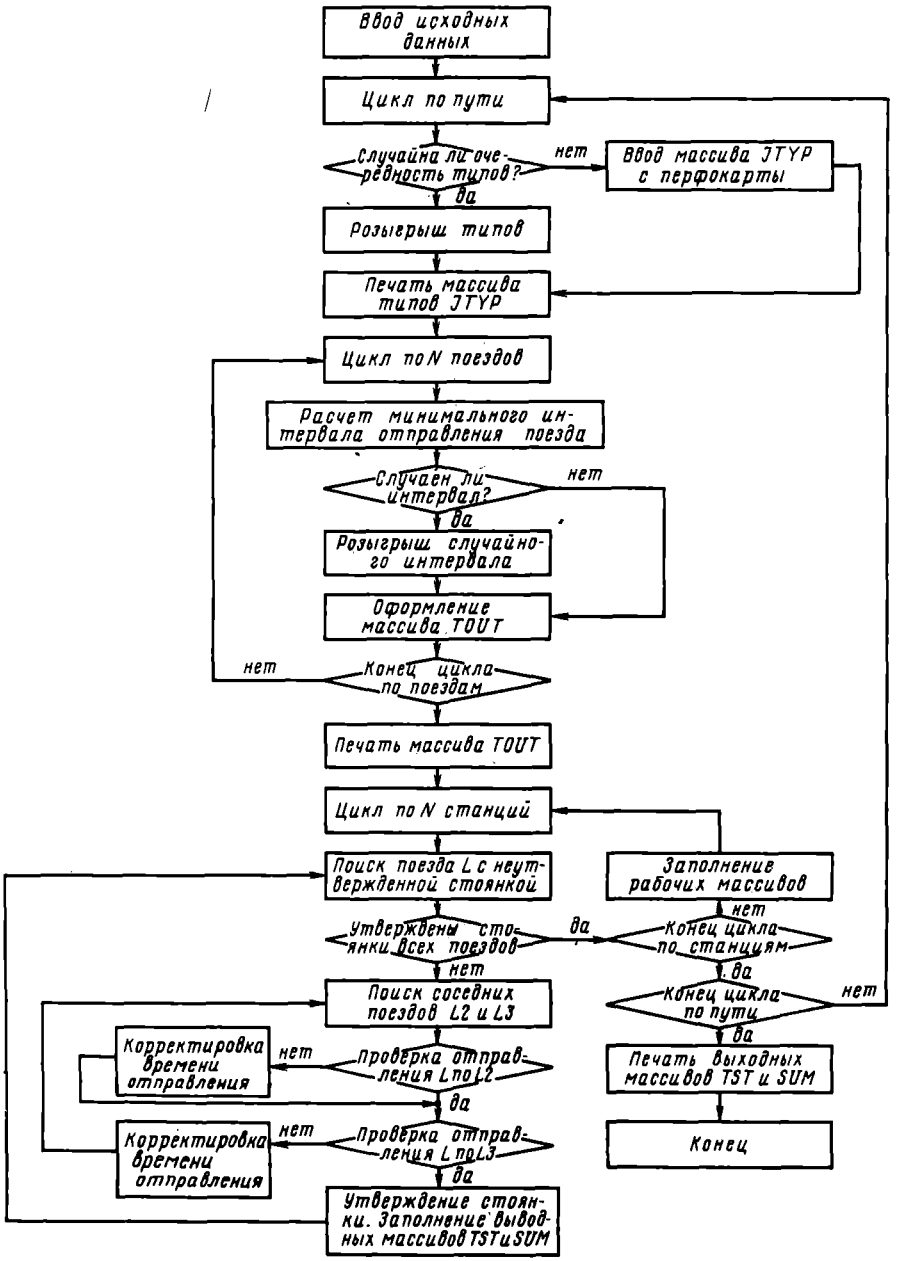


Рис. 6.14. Схема программы GRAF

Время стоянки поезда «утверждается», элементу массива NUT (LTC) присваивается значение 0 и осуществляется переход к отысканию следующего по времени прибытия поезда с «неутвержденной» стоянкой.

После того как будут откорректированы стоянки всех поездов, т. е. не будет поездов с «неутвержденной» стоянкой на данной станции, произойдет переход из блока отыскания очередного рассматриваемого поезда в блок «Конец расчета».

В этом блоке происходит заполнение рабочих массивов PB, OT(I) и NUT(I, C) для следующей станции.

После просмотра всех станций производится вывод на печать выходных массивов подпрограммы:

SUM (I, K) — суммарного времени стоянки на всех промежуточных станциях поезда I и TST(I, IC, K) — фактического времени стоянки поезда I, на промежуточных станциях IC пути K.

Выходной массив: JTIP (I, K) — очередность типов отправляемых поездов с начальной станции и TUT (I, K) — массив времени отправления поездов с начальной станции выводятся на печать в конце соответствующих расчетов.

На основе рассмотренного алгоритма разработана программа GRAF (рис. 6.14).

6.6. Моделирование тяговой сети

В большинстве случаев в расчетах рассматриваются однопутные или двухпутные участки с однородной тяговой сетью. Для моделирования такой тяговой сети можно использовать известные аналитические выражения для расчета мгновенных схем [24]

В этом случае, однако, приходится на каждом шаге системного времени определять количество нагрузок и их месторасположение. Кроме того, встречаются более сложные участки тяговой сети. К ним относятся станции большой протяженности, участки с двухпутными вставками, пересечениями и ответвлениями путей, многопутные участки с раположением некоторых путей на отдельном полотне.

В таких случаях для расчета мгновенных схем удобнее использовать матричные методы.

Матричные методы для расчета тяговой сети рассмотрены в [32]. В них определение дополнительных узлов схемы связано с отрезками пути, проходимыми поездами за шаг дискретизации времени. Такой метод усложняет составление матриц, делает их зависимыми от специализации путей и требует их модификации в случаях ее изменения или введения новых типов поездов. Поэтому далее излагается подход, ранее реализованный в специализированных аналоговых устройствах [36].

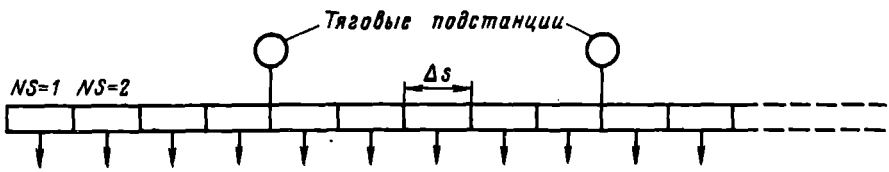


Рис. 6.15. Квантование тяговой сети

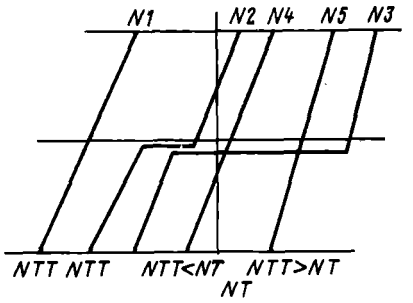


Рис. 6.16. Сечение графика движения

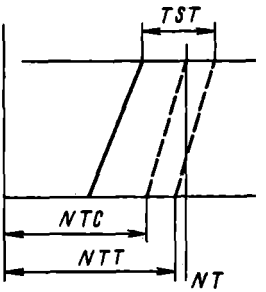


Рис. 6.17 Определение точки пересечения «нити» поезда в графике с его сечением

Тяговая сеть при этом подходе квантуется на отрезки Δs (рис. 6.15). Длина отрезка определяется требуемой точностью расчета. В программе для расчета участков переменного тока принято $\Delta s = 0{,}5$ км.

Каждой ячейке тяговой сети присваивается номер (1, 2, 3, ..., J, ...). После этого обычными методами могут быть составлены матрицы сопротивлений и проводимостей и найдены коэффициенты токораспределения.

Алгоритм формирования мгновенных схем[1] Для формирования мгновенных схем организуются циклы просмотра номеров поездов. На каждом его шаге сравнивается время отправления NTT поезда с текущим моментом времени NT (рис. 6.16). Если NTT$>$ NT, то происходит выход из цикла, так как продолжать сравнение нет смысла, поскольку все поезда будут правее сечения NT.

Если NTT$<$NT, то происходит сравнение времени прибытия на конечную станцию с временем NT. Если время прибытия меньше времени NT, то переходят к следующему шагу перебора номеров поезда до тех пор, пока время отправления рассматриваемого поезда не окажется меньше NT, а время его прибытия на конечную станцию — больше NT.

В этом случае организуется цикл для определения перегона, на котором произойдет пересечение «нити» поезда с сечением NT (рис. 6.17).

В результате этого находится перегон IC, для которого справедливо соотношение:

$$\mathrm{OT}(\mathrm{I},\ \mathrm{IC}) \leqslant \mathrm{NT} \leqslant \mathrm{PB}(\mathrm{I},\ \mathrm{IC}+1).$$

Далее определяется момент времени L от начала тягового расчета для поезда рассматриваемого типа:

$$\mathrm{L} = \mathrm{NT} - \mathrm{NTT} - \sum_{j=1}^{IC} \mathrm{TST}\ (\mathrm{I},\ \mathrm{IC},\ \mathrm{K}).$$

[1] Разработан З. А. Фоминой.

Для определения места нахождения поезда в момент времени NT организуется цикл просмотра массива с результатом тягового расчета, на основании которого находятся нагруженные ячейки тяговой сети и значения нагрузок.

На этом рассмотрение отдельных элементов имитационной модели можно закончить, хотя для ее практического использования необходимо иметь еще ряд моделирующих алгоритмов. Все они относятся к электрическим расчетам мгновенных схем и их обработке с целью определения значений параметров, по которым осуществляется выбор или проверка устройств электроснабжения.

6.7. Алгоритм учета связей между подсистемами при имитационном моделировании системы электроснабжения

В предыдущем параграфе рассмотрены моделирующие алгоритмы для отдельных агрегатов, входящих в имитационную модель. Для создания всей модели необходимо иметь алгоритм и разработанную на его основе программу, которые обеспечат учет взаимодействия отдельных агрегатов (программных модулей) и функционирование имитационной модели.

Имитационные модели, в которых учитывается электрическое взаимодействие системы тягового электроснабжения и электроподвижного состава (т. е. учитывается изменение напряжения на э. п. с. при выполнении тяговых расчетов), программно реализованы пока лишь для упрощенных моделей дорог постоянного тока. Поэтому рассмотрим вначале общий моделирующий алгоритм имитационной модели, не учитывающей взаимного влияния системы тягового электроснабжения и электроподвижного состава.

В такой модели подсистемы э. п. с. и графика движения не охватываются обратными связями с системой электроснабжения. Они моделируются изолированно и только их выходные массивы используются как исходные данные для расчета системы электроснабжения.

Укрупненная схема алгоритма предполагает, что отдельные ее блоки реализуются сами при помощи сложных алгоритмов.

Схема алгоритма (рис. 6.18) благодаря укрупнению проста и не требует пояснений, однако практическая реализация алгоритма на ЭВМ связана с созданием очень сложного и большого программного комплекса.

Теперь рассмотрим общий моделирующий алгоритм, в котором учитывается взаимодействие системы электроснабжения и э. п. с.

В приведенной ранее методике проведения тяговых расчетов предложены формулы, в которые непосредственно входит напряжение в контактной сети U и связанная с ним э. д. с. E, что существенно облегчает проведение тяговых расчетов с учетом фактического напряжения на токоприемнике каждого поезда. Понятно,

что такие тяговые расчеты могут выполняться только совместно с электрическими расчетами системы электроснабжения.

В качестве аргумента при выполнении тяговых расчетов для периодов потребления локомотивом энергии был принят ток. Такой подход дал возможность определять нагрузки с заранее заданной точностью.

Результаты выполненных таким образом расчетов передаются в программы электрических расчетов. Входными данными для программ электрических расчетов служили ток I (t), время $t(I)$, в течение которого он изменяется не более чем на $\pm\Delta I/2$ (где $\Delta I/2$— принятая точность представления тока).

В случае совместного проведения электрических и тяговых расчетов необходимо учитывать изменение напряжения за время $t(I)$. Это обстоятельство легко может быть учтено, если при проведении тяговых расчетов принять (как это делается во многих случаях) в качестве аргумента время. При одинаковой дискретизации времени в тяговых и электрических расчетах фактическое напряжение легко может быть учтено. На каждом шаге тяговых и электрических расчетов производится взаимный обмен информацией.

Существенным недостатком этого простого, на первый взгляд, подхода, является резкое увеличение времени проведения тягового расчета. Следует учитывать, что на каждом шаге электрических расчетов при таком подходе надо выполнять шаг тяговых расчетов для всех поездов, расположенных на участке.

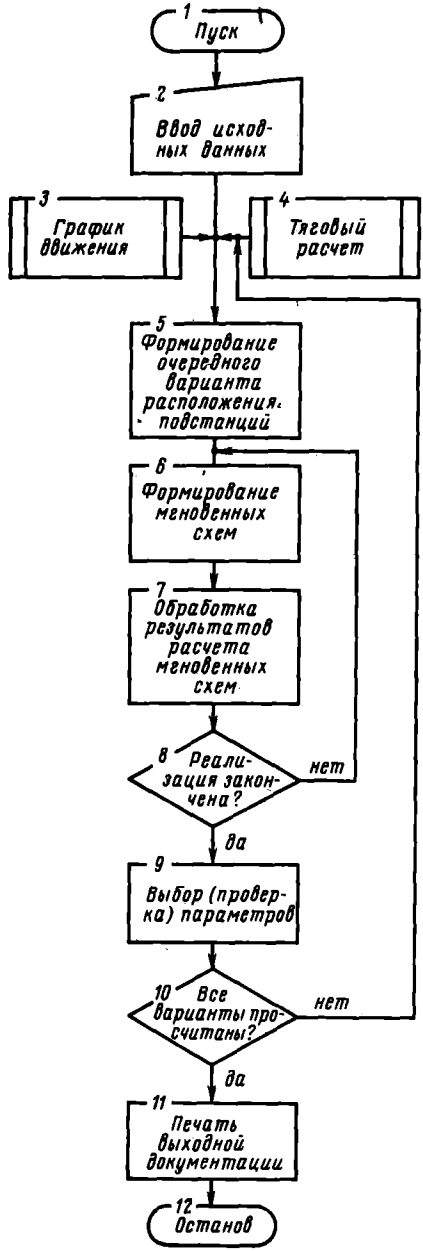


Рис. 6.18. Схема моделирующего алгоритма при раздельном выполнении тяговых и электрических расчетов

Чтобы избежать такого увеличения времени счета, разработан алгоритм совместного проведения тяговых и электрических расчетов, в котором порядок тяговых расчетов в значительной степени остается таким же, как и при выполнении их для неизменного напряжения в контактной сети. В основу этого алгоритма положена корректировка информации из программы тяговых расчетов, полученной на каждом шаге электрических расчетов, и возврат в нее откорректированной информации после использования.

Общий порядок расчета рассмотрим на шаге электрических расчетов. Пусть в момент, соответствующий этому шагу на расчетном участке, расположено N поездов и пусть для $m1$ из них к моменту выполнения шага k информация о их состоянии исчерпана. В этом случае в программу тяговых расчетов из программы электрических расчетов передаются откорректированные значения времени t_i пройденного пути s_i и скорости v_i. Кроме того, передаются значения напряжения для каждого поезда. Эта информация используется при выполнении следующего шага тяговых расчетов для рассматриваемых $m1$ поездов. На время выполнения этого шага работа программы электрических расчетов приостанавливается. Полученные в результате выполнения шага тяговых расчетов значения токов, скорости и пути передаются в программу электрических расчетов. В этой программе выполняются расчеты, начиная с шага i до шага, на котором исчерпывается информация некоторого числа поездов $m2$. Далее все повторяется. Информация, полученная в результате работы программы тяговых расчетов, корректируется на каждом шаге выполнения программы электрических расчетов.

Рассмотрим способы корректировки результатов тяговых расчетов.

Режим движения по ходовой характеристике. В разработанном алгоритме проведения тяговых расчетов, изложенном ранее, формула (6.4) дает зависимость между скоростью поезда, током его и э. д. с.

Так как $E = U - z\_I_d$, то можно написать:

$$v = \frac{(U - z\_I_d) I_d}{C(aI_d - b)}. \tag{6.46}$$

Изменение скорости при изменении напряжения и тока

$$\Delta v = \frac{\partial v}{\partial I_d}\Delta I + \frac{\partial v}{\partial U}\Delta U \tag{6.47}$$

С учетом формулы (6.46) после преобразований найдем

$$\Delta v_i = \frac{1}{C}\left[\frac{I_{di}}{aI_{di} - b}\Delta U_i - \left(\frac{z\_I_{di}}{aI_{di} - b} + \frac{bE_i}{(aI_{di} - b)^2}\right)\Delta I\right], \tag{6.48}$$

где E_i— значение э. д. с., переданное в программу тягового расчета для выполнения очередного его шага.

Далее воспользуемся формулой (6.7) для шага i электрических расчетов (шаг тягового расчета будет иметь для каждого поезда свой номер j):

$$\left[\frac{\eta E_i I_{di}}{v_i m} - 2{,}725(w + i_w)\right] t_i = 81{,}8\,\Delta v_i$$

После подстановки в эту формулу v_i и Δv_i из формулы (6.46) и (6.48) получим:

$$t_i = \frac{81{,}8\left[bE_i \Delta I - (aI_{di} - b) I_{di} \Delta E_i\right]}{\left[2{,}725(w_i + i_w) - \frac{C\eta}{m}(aI_{di} - b)\right](aI_{di} - b)^2 C}, \tag{6.49}$$

где $U_i - z\_\Delta I = E_i$.

Пусть t_{0i}— время, рассчитанное при неизменных U_i и E_i. Тогда можно записать

$$t_i = \alpha_{1i} t_{0i}, \tag{6.50}$$

$$\text{где } \alpha_{1i} = \left(1 - \frac{a_k I_{dik} - b_k}{b_k E_{ik} \Delta I} I_{dik} \Delta E_{ik}\right). \tag{6.51}$$

Здесь в обозначение величин введен дополнительный индекс k, указывающий номер расчетного поезда.

Введем также коэффициент α_{1jk}, который можно определить по формуле (6.51) при подстановке в нее I_{dik}, E_{jk} и ΔE_{jk} для шага j электрических расчетов.

Если на каком-либо шаге электрических расчетов α_{ij} получается отрицательным, то общее изменение ΔE_{jk} разбивается на части δE_{jk} так, чтобы α_{ijk} стало равным нулю. Очевидно,

$$\delta E_{jk} = \frac{b_k E_{jk} \Delta I}{(a_k I_{djk} - b_k) I_{djk}}. \tag{6.52}$$

При этом получаем $t_j = 0$; $I_{dj1k} = I_{djk} + \Delta I$; $E_{j1k} = E_{jk} + \delta_{jk}$. После этого вновь определяется δE_k по формуле (6.52), находятся новые значения I_{djk} и δE_{jk}. Такой расчет повторяется до тех пор, пока удовлетворяется неравенство

$$\Delta E_{jk} - \sum \delta E_{jk} > 0. \tag{6.53}$$

При нарушении его в программе электрических расчетов фиксируется возрастание тока от начального значения I_{djk} до конечного $I_{djk} + \Delta In$ при неизменном времени, где n— количество обращений к формуле (6.52) до нарушения соотношения (6.53). По завершении расчета такого приращения тока из программы электрических расчетов в программу тяговых расчетов передаются данные для очередного шага тяговых расчетов, в том числе конечное значение I_{djk}.

Как видно из формулы (6.52), значение коэффициента α_{1ik} будет изменяться на каждом шаге электрических расчетов, если будет меняться напряжение U_i, а следовательно, значение E_i и δE_i.

При корректировке результатов тяговых расчетов этот коэффициент используется следующим образом. Пусть в начале очередного шага тяговых расчетов для поезда k из программы электрических расчетов на шаге $i-1$ работы передаются значения $U_{i-1,\ k}$, $t_{i-1,\ k}$, $s_{i-1,\ k}$. После выполнения шага тяговых расчетов в программу электрических расчетов передаются значения t_{0ik}, S_{0ik}, I_{di}. В программе электрических расчетов по этим данным выполняется шаг расчета i, определяется значение напряжения у всех поездов и, в частности, поезда k. Определяется путь, пройденный поездом k за интервал дискретизации времени Δt. На основании результатов расчета определяется значение α_{1ik}. Очевидно, при $\alpha_{1ik} \neq 1$ время Δt в программе электрических расчетов будет соответствовать времени $\Delta t_{ik} = \frac{\Delta t}{\alpha_{1jc}}$ в тяговых расчетах. Поэтому результаты, полученные на очередном шаге тяговых расчетов, будут полностью использованы, когда сумма Δt_{ik} станет равной t_{0ik}. При проведении электрических расчетов на каждом его шаге h для всех поездов производится проверка соотношения:

$$\Delta t \sum_{j=1}^{h} \frac{1}{\alpha_{1jk}} < t_{0ik}, \tag{6.54}$$

где $\alpha_{1jk}=1$ (на шаге $i=1$ электрических расчетов непосредственно после получения данных из программы тяговых расчетов корректировка не производится).

Если это соотношение нарушается, то производится очередное обращение к программе тяговых расчетов. Общее время работы по данным очередного шага тяговых расчетов в зависимости от режима напряжения может быть как больше, так и меньше t_{0ik}.

На каждом шаге электрических расчетов корректируются также значения v_j и s_j по формулам:

$$v_j = \frac{E_j}{E_i} v_{0i} \quad \text{и} \quad \Delta s_j = v_j \Delta t. \tag{6.55}$$

Подсчитывается сумма

$$s_h = \sum_{j=i-1}^{h} \Delta s_j \tag{6.56}$$

Для шага h_1, при котором нарушается соотношение (6.54), определяются значения v_h и s_h по формулам (6.55) и (6.56), а также E_{h1}:

$$t_{hi} = \Delta t \sum_{j=i-1}^{h_1} \alpha_{1j} \tag{6.57}$$

Значения всех этих величин передаются в программу тяговых расчетов и используются для выполнения очередного их шага.

При корректировке результатов тяговых расчетов в программе электрических расчетов могут быть случаи выхода скорости за пределы допустимой и пути за пределы рассматриваемого элемента профиля. Это может иметь место, если на некоторых шагах электрического расчета напряжение U_j будет больше напряжения U_i, по которому производились тяговые расчеты на текущем шаге. Для предотвращения этого на каждом шаге электрических расчетов следует, кроме проверки формулы (6.54), выполнять для условия $U_j > U_i$ проверку неравенств:

$$v_j \leqslant v_{\text{доп}} \text{ и } s_j \leqslant s_{0k} \tag{6.58}$$

где $v_{\text{доп}}$— максимально допустимая скорость на текущем участке пути; s_{0k}— расстояние до конца этого участка.

В случае нарушения хотя бы одного из неравенств (6.58) вызывается программа тяговых расчетов, в которую передаются текущие значения начальных данных.

Предложенная корректировка результатов шага тяговых расчетов основана на приближенной формуле (6.47). Она дает хорошие результаты при небольших приращениях $\triangle I$ и $\triangle U$ ($\triangle E$). Приращение тока ограничено принятым квантованием, а $\triangle E$ не может быть больше получаемой по формуле (6.52), т. е. она не может быть причиной изменения тока больше, чем на $\triangle I$. Значение δU, ограниченное этим условием, дает возможность использовать формулу (6.47), не внося в расчеты существенной погрешности.

Режим пуска. Если в формуле (6.19) коэффициенты a_1 и b_1 определены для номинального напряжения $U_{\text{ном}}$, то при напряжении U_j, очевидно, можно написать:

$$I_U = a_1 \frac{U_j}{U_{\text{ном}}} + b_1 \frac{U_j}{U_{\text{ном}}} v.$$

Из этого следует, что при напряжении U_j в формулу (6.22) надо вместо b_1 и b_2 подставить $b_1 \frac{U_j}{U_{\text{ном}}}$ и $b_2 \frac{U_j}{U_{\text{ном}}}$, при этом отношение этих величин, входящих в формулу, не изменится.

Таком образом, расчет при пуске не меняется при изменении напряжения до выхода на ходовую характеристику. Однако выход на нее будет осуществлен при другой скорости. При проверке выхода на ходовую характеристику по току никаких

изменений в алгоритме тягового расчета делать не требуется. В случае выполнения такой проверки по скорости скорость выхода на ходовую характеристику при напряжении U_j надо корректировать по формуле:

$$v_{nj} = v_{\text{пном}} \frac{U_j}{U_{\text{ном}}}. \tag{6.59}$$

Эту скорость следует использовать в программе тяговых расчетов при вызове ее для выполнения шага в режиме пуска.

Следует заметить, что, кроме указанных ранее проверок при выполнении электрических расчетов, необходимо на каждом их шаге производить проверку

$$v_j > v_{nj}. \tag{6.60}$$

В случае использования данных для режима движения по ходовой характеристике и нарушении неравенства (6.60) должен последовать вызов программы тяговых расчетов. При работе с результатами тяговых расчетов для режима пуска вызов программы тяговых расчетов производится, если неравенство (6.60) соблюдено.

Общий порядок расчета[1] До начала всех расчетов должен быть произведен предварительный тяговый расчет для всех типов поездов при неизменном напряжении на токоприемниках. Этот предварительный расчет необходим для приближенного определения времени хода поездов по перегонам участка и составления графика движения поездов.

Корректировка тягового расчета для каждого поезда начинается с первого шага электрических расчетов. До того считается, что поезда идут при неизменном напряжении. На первом шаге в подпрограмму тяговых расчетов передается лишь время хода с начала движения каждого поезда.

В подпрограмме формирования мгновенных схем FORM при совместном расчете следует ввести учет числа и нумерации поездов, находящихся на участке на данном шаге электрического расчета. Это изменение программы FORM вводится для учета того, что характер движения поездов одного и того же типа будет зависеть от фактического напряжения на токоприемнике. Возможны случаи, когда шаг по времени электрического расчета оказывается больше шага по времени в тяговом расчете. Поэтому в подпрограмме тягового расчета следует просчитать все шаги до того момента, когда суммарное время шагов тягового расчета окажется больше времени дискретизации электрического расчета. В этом случае наблюдается скачок тока

$$I_{\text{скjk}} = h\Delta I,$$

где ΔI — принятое в тяговом расчете приращение тока на одном шаге; h — число шагов.

Работа подпрограммы стыкования электрического расчета с тяговым происходит в следующем порядке (рис. 6.19).

По первому сечению графика движения определяются число поездов, их типы и время хода от начальной станции. Эти данные

[1] Разработан З. А. Фоминой.

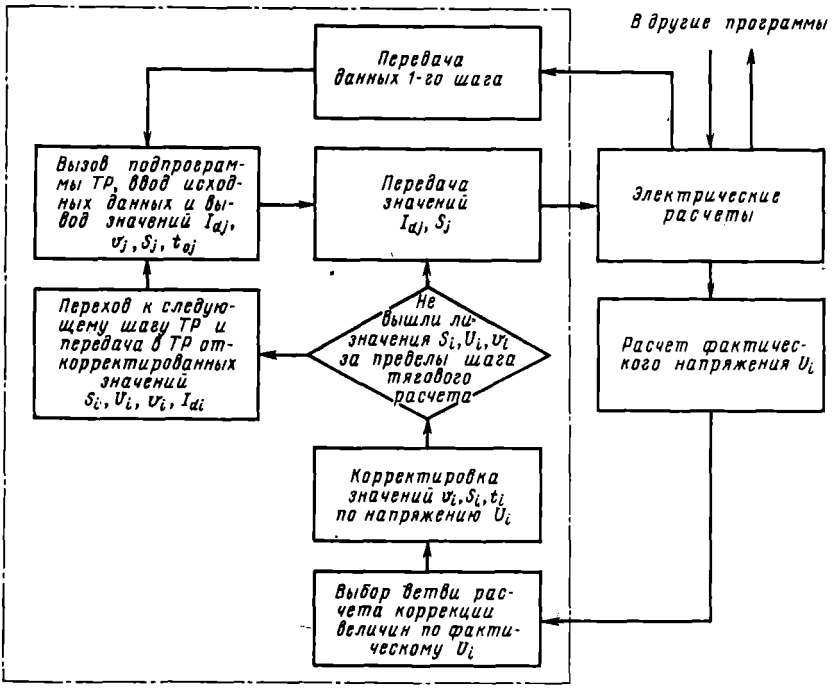


Рис. 6.19. Схема программы стыковки тяговых и электрических расчетов

передаются в специальную подпрограмму, которая вызывает подпрограмму тягового расчета TR. Подпрограмма TR по исходным данным (в первый раз по времени, потом по откорректированным значениям времени предыдущего шага, напряжению и пути) выдает значения одного шага тягового расчета: ток на шаге I_{dj}, скорость v_i, интервал по времени Δt_{0j} и путь s_j. Далее подпрограмма STYK по этим значениям шага находит ток поезда, точное местонахождение поезда для данного момента электрических расчетов и передает их в блок электрических расчетов. Одновременно подсчитываются и реальные напряжения на токоприемниках поездов, которые передаются обратно в подпрограмму STYK. Блок по фактическому напряжению корректирует переданные ранее значения шага тягового расчета v_i, s_i и Δt_i. Далее эти откорректированные значения сравниваются со значениями величин, полученных на шаге тягового расчета при неизменном напряжении. При выходе какого-либо проверяемого значения за пределы значений, полученных на данном шаге тягового расчета, вызывается подпрограмма TR и передаются откорректированные значения v_i, s_i и Δt_i, t_i.

Если же значения величин, полученных на шаге тягового расчета еще не исчерпаны, в блок электрического расчета из блока STYK передаются откорректированные по фактическому напряжению значения для следующего момента. На этом завершается один цикл расчетов.

Контрольные вопросы

1. Что такое структуризация объекта исследования?
2. Для чего сложная система разделяется на подсистемы?
3. Назовите основные подсистемы, на которые может быть разбита система «Электрифицированная железная дорога».
4. Как учитываются связи между подсистемами?
5. Какие основные блоки включает структура имитационной модели электрифицированного участка железной дороги?
6. Составьте укрупненный алгоритм совместного проведения тягового и электрического расчетов.

Глава 7

АНАЛОГОВЫЕ И АНАЛОГО-ЦИФРОВЫЕ ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ

7.1. Общая характеристика аналоговых вычислительных машин

Аналоговой вычислительной машиной (АВМ) в широком смысле слова является любая материальная система (например, электрическая цепь), которая устанавливает определенные заранее заданные соотношения между непрерывно изменяющимися физическими величинами (обычно называемыми машинными переменными).

Этими величинами могут быть любые поддающиеся количественной оценке параметры физического явления. Но наиболее удобно в качестве этих величин использовать такие, которые легко могут быть измерены (электрические токи, напряжения, углы поворота валов в механических устройствах). Машинные переменные должны удовлетворять математическим соотношениям, описывающим рассматриваемую задачу в соответствующих физических терминах или обозначениях.

Для этого между машинными переменными и переменными рассматриваемой задачи устанавливают определенные соотношения, которые выражаются коэффициентами пропорциональности, называемыми *масштабами*. Подбор этих коэффициентов называют *масштабированием*.

Решение математических и инженерных задач на АВМ сводится к эксперименту с некоторой физической системой, которая сознательно построена таким образом, чтобы математическое описание было одинаково с задачей, подлежащей решению. В этом смысле решение задачи на АВМ называют *моделированием*, а саму АВМ называют *моделью*.

Сущность моделирования заключается в замене исходных явлений или систем (оригиналов) другими явлениями или системами, называемыми моделями. Этот переход облегчает проведение исследований с целью проверки гипотез, нахождения оптимальных параметров системы на стадии проектирования, а также получения информации о поведении системы в любых интересующих исследователя режимах вплоть до аварийных, не осуществимых на реальном объекте.

Существуют два основных способа моделирования с использованием технических средств для проведения модельного эксперимента — машинное и физическое моделирование. При физическом моделировании природа оригинала и модели, как правило, одинакова, при машинном — различна. *Физические модели* реализуются на основе теории подобия и их основной недостаток — малая универсальность.

Машинные модели реализуются на вычислительных машинах на основе математического описания оригинала — математической модели.

В зависимости от используемых вычислительных машин различают машинное моделирование на АВМ и ЦВМ. В последнее время эти вычислительные машины сопрягают и получают вычислительную систему, которая называется *гибридной*. При машинном моделировании вычислительная система может сопрягаться с реальной аппаратурой. Такое моделирование называют *полунатурным*. Оно позволяет воспроизводить процессы, которые не имеют математического описания. Моделирование на АВМ во многих случаях является предпочтительным по сравнению с моделированием на ЦВМ. Это определяется по крайней мере двумя обстоятельствами.

Во-первых, основным математическим аппаратом для описания многих физических явлений в устройствах электрических железных дорог являются дифференциальные уравнения, для решения которых предназначены АВМ, поскольку протекающие в них при моделировании процессы сами описываются аналогичными уравнениями. Последнее обстоятельство определяет большую скорость решения этих уравнений, так как процессы в элементах АВМ происходят одновременно.

При построении АВМ использован операционный принцип, который подразумевает наличие операционных блоков, выполняющих элементарные математические операции над переменными, используемыми в задаче (например, умножение на постоянную величину, сложение, вычитание, интегрирование и т. п.). Это позволило отойти от прямой аналогии между параметрами изучаемой физической системы и параметрами модели, принципа, который широко использовался на заре аналоговой вычислительной техники при построении настольных электрических моделей электрических цепей, сетей, энергосистем.

Во-вторых, устройства электрических железных дорог являются «грубыми» системами, параметры которых известны с большой степенью приближения, а иногда требуют подбора, поэтому ограниченная точность АВМ как вычислительного прибора позволяет получать удовлетворительную оценку качества исследуемой системы без большой затраты труда и времени.

Несмотря на значительные успехи в области моделирования систем на ЦВМ, аналоговое моделирование продолжает оставаться мощным средством изучения динамических, переходных режимов в устройствах электрических железных дорог. Особенно удобно использовать аналоговые модели для исследования и отладки микропроцессорных систем автоматического управления объектами с большой инерцией и нелинейными звеньями.

Отечественная промышленность выпускала аналоговые машины типов ИПТ5, «Аналог-1, ЭМУ-10, МН-7 (ламповые), МН-10М (полупроводниковые), в последнее время — АВК-31, АВК-32.

Аналоговая вычислительная машина АВК-31 наиболее подходит для применения в учебном процессе. В ее комплект входят: три блока интеграторов-сумматоров, состоящих из двух сумматоров и двух интеграторов, блок нелинейной функции, блок перемножения, блок набираемого оператора и блок логических переменных. Номинальный диапазон изменения аналоговых величин на входах и выходах операционных блоков составляет от 0 до ± 10 В. Блок набираемого оператора предназначен для установки дополнительных радиоэлементов, необходимых для решения задач и выполнения некоторых логических операций (имеются два триггера со счетным и установочным входами, три логических элемента И-НЕ с четырьмя входами, логический элемент ИЛИ-НЕ с четырьмя входами, два индикаторных элемента). Блок логических элементов содержит два компаратора, схему для реализации зоны нечувствительности, четыре диодных элемента и четыре потенциометра, четыре реле с магнитоуправляемыми контактами, два логических инвертора. Аналоговая вычислительная машина АВК-32 предназначена для исследовательских целей и обладает возможностями стыковки с цифровой вычислительной машиной.

В последние годы получили развитие новые вычислительные системы — так называемые гибридные вычислительные системы (ГВС). Примером такой системы является ГВС-100, которая разработана как единое целое, содержит цифровую и аналоговую части, имеет математическое обеспечение, позволяющее решать гибридные задачи на ЦВМ и АВМ в реальном масштабе времени, выполнять работу цифровой части в мультипрограммном режиме, автоматизировать подготовку задач для АВМ.

Система программирования предусматривает использование языков Фортран, а также языка Автокод, дополненного операторами управления устройствами ГВС-100 и операторами управления АВМ. В систему программирования входят специализированные языки, объединенные в систему автоматизации аналогового программирования (СААП) и позволяющие вводить в ЦВМ уравнения, подлежащие решению на АВМ. Схемы для аналогового набора, определение масштабов, расчет коэффициентов передач блоков, напряжения для статического контроля задачи можно выдать на печатающее устройство. Таким образом обеспечивается полная автоматизация подготовки и решения задач.

7.2. Операционный усилитель. Операционные блоки

Операционный принцип построения АВМ позволил создать вычислительную машину, обладающую большой гибкостью при решении широкого класса математических задач. Для решения линейных обыкновенных дифференциальных уравнений и дифференциальных уравнений с нелинейными коэффициентами в состав АВМ должны входить функциональные блоки, выполняющие математические опе-

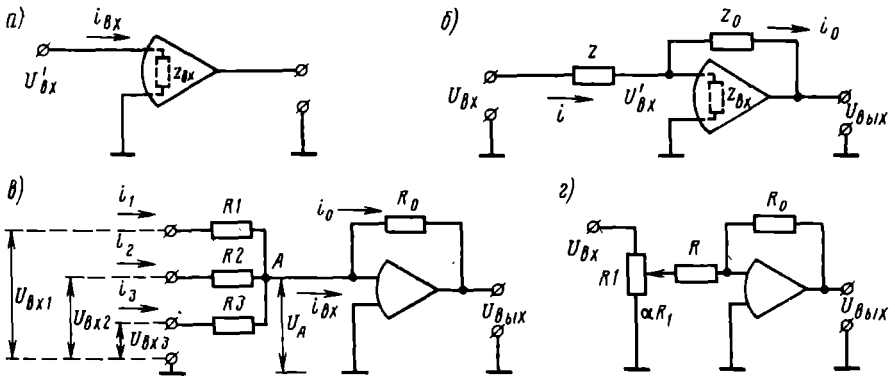


Рис. 7.1. Операционный усилитель (*а*) и операционные блоки: операционный блок (масштабный усилитель) (*б*), блок суммирования трех переменных (сумматор) (*в*), операционный блок с потенциометром на входе (*г*)

рации суммирования, интегрирования, дифференцирования, умножения на постоянный и переменный коэффициенты, перемножения и деления, а также блоки для нелинейных функциональных преобразований. Кроме перечисленных основных блоков, в состав АВМ входят устройства обслуживания: блоки для электропитания функциональных блоков, управления процессом решения, контроля и настройки коэффициентов, индикации результатов решения задачи, наборное поле для надежного соединения функциональных блоков в соответствии с решаемой задачей.

Основным элементом АВМ является электронный усилитель постоянного тока (рис. 7.1, *а*), на базе которого строятся все функциональные блоки АВМ. В настоящее время с появлением усилителей постоянного тока в интегральном исполнении этот усилитель стали называть операционным усилителем (ОУ). Он широко используется в устройствах автоматики при выполнении различных математических операций и преобразований.

По конструктивному исполнению АВМ различны, но каждая в своей основе содержит набор усилителей постоянного тока, также имеющих разные исполнения как по схеме, так и по примененной элементной базе (ламповые, полупроводниковые, интегральные).

Усилитель постоянного тока выполняется так, чтобы можно было обеспечить его отрицательную обратную связь по напряжению. Он должен иметь большой коэффициент усиления напряжения по постоянному току, большое входное и малое выходное сопротивления, а также нулевой выходной сигнал при отсутствии входного сигнала.

В современных аналоговых машинах используют операционные усилители с коэффициентом усиления около 10^5—10^7 Частотный

диапазон усилителей различен. В малых АВМ полоса пропускания усилителей составляет сотни герц, в средних и больших АВМ (число операционных усилителей 50 и 100) усилители имеют полосу пропускания примерно несколько тысяч герц. Широкая полоса пропускания при работе в реальном масштабе времени обеспечивает воспроизведение без искажений всех динамических свойств системы с учетом нелинейных звеньев и сигналов сложной формы, которые содержат высшие гармонические составляющие. Широкая полоса пропускания обеспечивает высокое быстродействие, что позволяет моделировать системы в ускоренном масштабе времени.

Из-за явления насыщения ОУ имеют ограниченную зону линейности, определяемую максимальным диапазоном изменения выходного напряжения, в котором можно получить без искажения выходной сигнал. Диапазон изменения напряжений обычно составляет ± 100 В (ламповые усилители); ± 50 В; ± 25 В и ± 10В (полупроводниковые). Его называют диапазоном линейности. Он определяет выбор масштабов машинных переменных. Масштабы должны быть выбраны так, чтобы ни одна машинная переменная в процессе решения задачи не выходила за пределы линейной зоны усилителей.

ОУ обладают напряжением дрейфа нуля выходного сигнала, которое проявляется в медленном изменении выходного напряжения усилителя при нулевом сигнале на входе. Основными причинами дрейфа нуля являются нестабильность источников питания, изменение температурного режима работы схемы и нестабильность характеристик элементов схемы (резисторов, транзисторов и т. п.). Дрейф нуля — это источник погрешностей операционных блоков. Для его уменьшения применяют стабилизированные источники питания, резисторы и конденсаторы со стабильными характеристиками, специальные схемные способы компенсации дрейфа [34, 35].

Вход усилителя постоянного тока «потенциально заземлен». Это означает, что напряжение на входе усилителя пренебрежимо мало и чем больше коэффициент усиления, тем меньше напряжение на входе усилителя. Часто вход усилителя называют «суммирующей точкой», так как в этой точке происходит суммирование токов от всех сигналов. Чем меньше входной ток, тем выше точность суммирования.

Из краткого рассмотрения характеристик ОУ следует, что коэффициент усиления является важнейшей характеристикой, влияющей на точность выполнения математических операций.

Определим коэффициент передачи операционного блока, производящего умножение напряжения на его входе на постоянный коэффициент (рис. 7.1, *б*).

Так как усилитель постоянного тока имеет большое входное сопротивление $z_{вх}$, то входным током можно пренебречь, и тогда токи во входной цепи операционного блока (цепь входных сопротивлений z) и в цепи обратной связи равны $i=i_0$. Выразим токи через соответствующие напряжения и, учитывая, что $U_{вых}=$

$= -kU'_{вх}$ (где k — коэффициент усиления усилителя постоянного тока), получим:

$$U_{вых} = -\frac{U_{вх}/z}{1/z_0 + (1/z_0 + 1/z)/k}. \quad (7.1)$$

Знак «минус» определяется конструкцией операционного усилителя как инвертора. Так как коэффициент усиления k велик ($k \to \infty$), то вторым членом в знаменателе выражения (7.1) можно пренебречь. Тогда

$$U_{вых} = -U_{вх}z_0/z. \quad (7.2)$$

Отношение $U_{вых}/U_{вх}$ или z_0/z называют *коэффициентом передачи операционного блока*. Последнее выражение справедливо только для масштабного блока и суммирующего блока по каждому из входов.

Из этого выражения видно, что коэффициент передачи операционного блока не зависит от коэффициента усиления усилителя постоянного тока, но необходимо, чтобы этот коэффициент был достаточно большим.

Как следует из рис. 7.1, *б*, входное сопротивление операционного блока $z_{вх} = U_{вх}/i = (zi + U'_{вх})/i$, но $i = (U'_{вх} - U_{вых})/z_0 = U'_{вх}(1 + k)/z_0$, следовательно,

$$z_{вх} = z + z_0/(1 + k). \quad (7.3)$$

Таким образом, входное сопротивление операционного блока определяется в основном сопротивлением входной цепи z. Выходное сопротивление операционного блока при $k \to \infty$ стремится к нулю.

В зависимости от того, какие элементы включены на входе и в цепи обратной связи (резистор, конденсатор), операционный усилитель может выполнять различные математические операции.

Для суммирующего блока (сумматора) в соответствии с первым законом Кирхгофа для точки A (рис. 7.1, *в*) справедливо следующее уравнение:

$$i_1 + i_2 + i_3 = i_0 + i_{вх}. \quad (7.4)$$

Так как входное сопротивление усилителя велико, то можно считать, что ток $i_{вх} \approx 0$. Остальные токи можно записать по закону Ома и, подставив их выражение в (7.4), получить

$$(U_{вх1} - U_A)/R_1 + (U_{вх2} - U_A)/R_2 + (U_{вх3} - U_A)/R_3 = (U_A - U_{вых})/R_0. \quad (7.5)$$

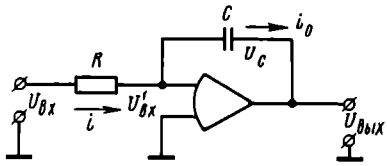
Решая уравнение относительно $U_{вых}$ с учетом того, что $U_A = -U_{вых}/k$, получим:

$$U_{вых} = \frac{U_{вх1}/R_1 + U_{вх2}/R_2 + U_{вх3}/R_3}{-(1/R_0) + (1/R_0 + 1/R_1 + 1/R_2 + 1/R_3)/k} \quad (7.6)$$

Принимая во внимание, что $k \to \infty$, выражение (7.6) можно записать

$$U_{вых} = -\left(\frac{R_0}{R_1}U_{вх1} + \frac{R_0}{R_2}U_{вх2} + \frac{R_0}{R_3}U_{вх3}\right), \quad (7.7)$$

где R_0/R_i — коэффициенты передачи сумматора по i-му входу.



△ Рис. 7.2. Интегрирующий блок

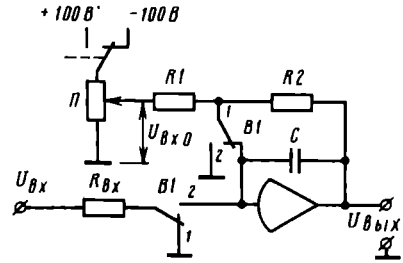


Рис. 7.3. Электрическая схема задания начальных условий на интеграторе

Если в суммирующем блоке использовать один вход, то такой операционный усилитель выполняет умножение на постоянный коэффициент $\alpha = R_0/R$ и такой блок принято называть *масштабным*.

Изменение коэффициента передачи масштабного усилителя (рис. 7.1, *г*) можно производить дискретно путем изменения соотношения R_0 и R и плавно, используя потенциометр $R1$ на входе.

При этом $U_{вых} = -\alpha U_{вх} R_0/R$, где α— коэффициент деления делителя, который плавно изменяется от 0 до 1. Если резисторы R_0 и R в масштабном блоке равны, то $U_{вых} = -U_{вх}$. Такой усилитель называют *инвертором*, так как он изменяет только знак переменной, поданной на его вход.

У интегрирующего блока (рис. 7.2) входное сопротивление усилителя велико, поэтому $i \approx i_0$. Выразим токи через соответствующие напряжения и, учитывая, что $i_0 = C dU_c/dt$, получим:

$$(U_{вх} - U'_{вх})/R = Cd(U'_{вх} - U_{вых})/dt. \qquad (7.8)$$

Учитывая, что $U'_{вх} \to 0$ (потенциально заземленная точка), преобразуем выражение (7.8):

$$dU_{вых}/dt = -(1/RC)U_{вх}. \qquad (7.9)$$

Интегрируя обе части уравнения по времени, получим

$$U_{вых}(t) = -1/RC \int_0^t U_{вх}(t)dt + U_0, \qquad (7.10)$$

где U_0— постоянная интегрирования.

Таким образом операционный усилитель с емкостью в обратной связи выполняет интегрирование напряжения, поданного на его вход с коэффициентом $1/RC$.

Если интегрирующий блок выполнить с несколькими входами, то получим операционный усилитель, интегрирующий сумму входных напряжений:

$$U_{вых} = -1/C \int_0^t \sum_{i=1}^{n} U_{вхi}/R_i\, dt + U_0. \qquad (7.11)$$

Коэффициент передачи для каждого входа задается соответствующим выбором входным резисторов или коэффициентов деления делителей α.

При решении дифференциальных уравнений на АВМ с использованием интегрирующих блоков необходимо задавать начальные условия задачи, которые реализуются путем установки на конденсаторе начального напряжения.

Возможны различные способы задания начальных условий. Рассмотрим наиболее простой из них (рис. 7.3).

При установке тумблера *B1* в положение *1* (режим «исходное положение») образуется так называемое инерционное звено, когда в цепи обратной связи включается *RC*-цепь. Напряжение на выходе такого звена в установившемся режиме ($t \to \infty$):

$$U_0 = -U_{вх0} R_2 / R_1, \tag{7.12}$$

где $U_{вх0}$— напряжение, снимаемое с движка потенциометра *П*.

При установке тумблера в положение *2* (режим интегрирования) напряжение, пропорциональное интегралу от входного напряжения, будет суммироваться с начальным напряжением на конденсаторе. Для дифференцирующего блока по аналогии с интегрирующим блоком можно показать, что

$$U_{вых}(t) \approx -RC dU_{вх}(t)/dt. \tag{7.13}$$

Дифференцирующий усилитель обладает повышенной чувствительностью к помехам, поэтому применяется редко, но в случае необходимости применяют фильтрацию сигнала от высокочастотных помех совместно с операцией дифференцирования.

7.3. Программирование для АВМ

Вычислительные особенности АВМ наиболее приспособлены для решения обыкновенных дифференциальных уравнений, заданных в форме задачи Коши. Поэтому решение на АВМ других задач, не связанных в первоначальной постановке с обыкновенными дифференциальными уравнениями, возможно лишь после того, как с помощью специальных приемов и методов исходные задачи сведены к эквивалентным задачам Коши.

Решению задач с помощью АВМ предшествует подготовительная работа, называемая программированием. В процессе программирования выполняется ряд работ, которые можно условно назвать этапами. Первый этап связан с анализом исходной задачи и разработкой структурной схемы. Затем производится выбор масштабных множителей (масштабов) и определяются коэффициенты передачи всех операционных блоков по каждому входу. Заканчивается процесс программирования разработкой коммутационной схемы соединений операционных блоков АВМ. Следует отметить, что результат про-

граммирования оформляется в виде коммутационной схемы операционных блоков. При программировании могут использоваться четыре уровня схем, отличающихся степенью детализации информации об организации вычислительного процесса: функциональные, структурные, коммутационные и электрические схемы.

Функциональная схема задачи содержит заданное уравнение в общем виде, а также заданные внешние воздействия, которые необходимо воспроизвести при организации решения исходного уравнения.

Структурная схема составляется для математических переменных и детализируется для отдельных операционных блоков, но схема не ориентирована на какую-либо конкретную АВМ.

Коммутационная схема составляется с ориентацией на конкретную АВМ. На коммутационных схемах проставляются номера используемых операционных блоков и использованных входов операционных блоков в соответствии с их изображением на наборном поле АВМ. В этих схемах переменные задачи изображаются или подразумеваются как машинные переменные (электрические напряжения).

Электрическая схема отличается от коммутационной еще большей детализацией. Она доводится до уровня операционных усилителей и радиодеталей, образующих входные цепи и цепи обратных связей операционного усилителя.

При соответствующем опыте иногда составляется только коммутационная схема и частично дополняется электрическими схемами отдельных блоков. В табл. 7.1 приведены наиболее распространенные изображения блоков и элементов аналоговых схем.

Для программирования однородных дифференциальных уравнений и неоднородных, внешнее воздействие в которых не содержит производных, применяется общий метод программирования, который заключается в последовательном понижении порядка высшей производной, входящей в дифференциальное уравнение, и называется методом непосредственного преобразования.

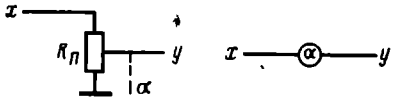
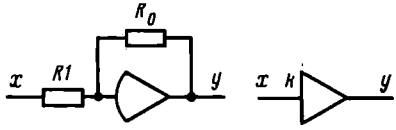
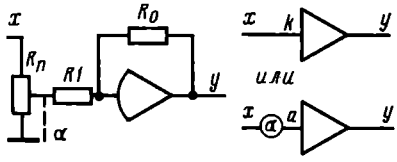
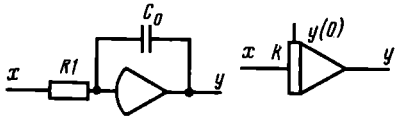
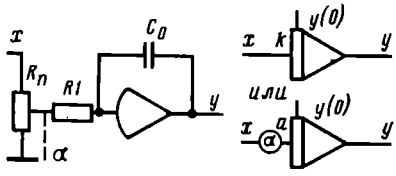
Рассмотрим этот метод на примере программирования линейного неоднородного дифференциального уравнения второго порядка с постоянными коэффициентами, которыми описываются колебания простого одноосного экипажа при движении по неровному железнодорожному пути (рис. 7.4) с амплитудой неровности η_0 при синусоидальной ее форме:

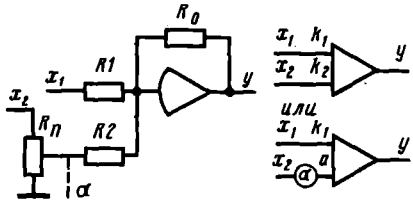
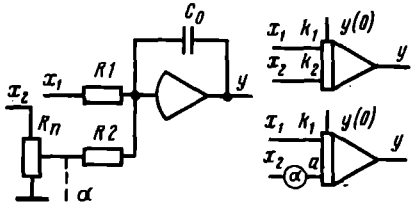
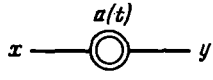
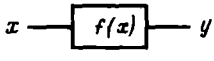
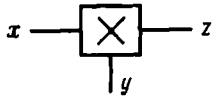
$$m\ddot{z} + c\dot{z} + kz = \eta_0 \sin \omega t + \eta_0 \omega \cos \omega t. \qquad (7.14)$$

Начальное условие: $\dot{z}(0) = 0$, $z(0) = 0$; $\eta\omega\cos\omega t$— производная по времени от функции $\eta_0 \sin\omega t$.

Составление структурных схем будем выполнять без учета масштабов переменных, поэтому входные и выходные переменные напряжения операционных блоков (рис. 7.5) обозначаются теми же символами, что и входящие в исходные уравнения переменные.

Таблица 7.1

Условное обозначение		Выполняемая функция
Вариант 1	Вариант 2	
Операционный усилитель 		$y=-kx$
Потенциометр 		$y=\alpha x$
Масштабный усилитель: нерегулируемый вход 		$y=-kx;\quad k=\frac{R_0}{R_1}$
регулируемый вход 		$y=-kx;\quad k=\frac{R_0}{R_1}\alpha=\alpha a$
Интегратор: нерегулируемый вход 		$y=-\int_0^t kx\,d\tau+y(0);\quad k=\frac{1}{R_1C_0}$
регулируемый вход 		$y=-\int_0^t kx\,d\tau+y(0);$ $k=\alpha\frac{1}{R_1C_0}$ или $k=\alpha a;\ a=\frac{1}{R_1C_0}$

Условное обозначение		Выполняемая функция
Вариант 1	Вариант 2	
Сумматор 		$y=-(k_1x_1+k_2x_2)$; $k_1=\frac{R_0}{R_1}$; $k_2=\alpha\frac{R_0}{R_2}$ или $k_1=\frac{R_0}{R_1}$; $k_2=\alpha a$; $a=\frac{R_0}{R_2}$
Интегратор-сумматор 		$y=-\int_0^t(k_1x_1+k_2x_2)\cdot d\tau+y(0)$; $k_1=\frac{1}{R_1C_0}$; $k_2\alpha=\frac{1}{R_2C_0}$ или $k_1=\frac{1}{R_1C_0}$; $k_2=\alpha a$; $a=\frac{1}{R_2C_0}$
Блок переменного коэффициента 		$y=a(t)x$
Блок нелинейной функции 		$y=(x)$
Блок перемножения 		$z=kxy$

Для составления структурной схемы используется следующая методика.

1. Уравнение (7.14) разрешается относительно старшей производной и все коэффициенты делятся на коэффициент при старшей производной:

$$\ddot{z} = -\frac{c}{m}\dot{z} - \frac{k}{m}z + \frac{\eta_0}{m}\sin\omega t + \frac{\eta_0\omega}{m}\cos\omega t; \qquad (7.15)$$

$$\ddot{z} = -\alpha_1\dot{z} - \alpha_2 z + \alpha_3\sin\omega t + \alpha_4\cos\omega t. \qquad (7.16)$$

2. Вычерчивается цепочка из последовательно соединенных интеграторов, число которых равно порядку дифференциального уравнения (см. рис. 7.5).

3. Входная переменная цепочка интеграторов обозначается через $\ddot{z}$, и по цепочке слева направо отмечаются выходные переменные интеграторов с учетом того, что каждый интегратор является одновременно инвертором знака любой переменной, поданной на его вход.

4. К цепочке интеграторов подсоединяется сумматор, на выходе которого образуется сумма $-\alpha_1\dot{z} - \alpha_2 z + \alpha_3\sin\omega t + \alpha_4\cos\omega t$. Для этого каждая переменная на вход сумматора подается с обратным знаком, так как любой операционный блок инвертирует переменные, поданные на его вход.

5. Замыкаются обратные связи, по которым с выходов интеграторов соответствующие переменные подаются на вход сумматора. В случае необходимости при этом переменные пропускаются через инверторы.

Так, переменная $\dot{z}$ подается на сумматор через инвертор, потому что на выходе первого интегратора она получается со знаком «минус».

Если поведение переменной $\ddot{z}$ при решении задачи не представляет интереса, то функции сумматора и интегратора можно совместить в одном операционном блоке и схема преобразуется к схеме, показанной на рис. 7.6. Число операционных блоков уменьшается на один. Как правило, такой прием используют при решении дифференциальных уравнений, описывающих переходные процессы в электрических цепях, когда скорости изменения токов не представляют интереса.

Если дифференциальное уравнение содержит в правой части производные входного воздействия, то применяется *метод вспомогательной переменной*, который основан на преобразовании исходного уравнения в систему уравнений с вспомогательной переменной.

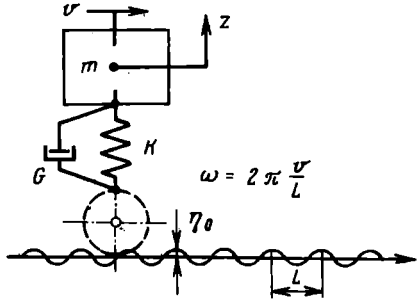


Рис. 7.4. Модель одноосного экипажа

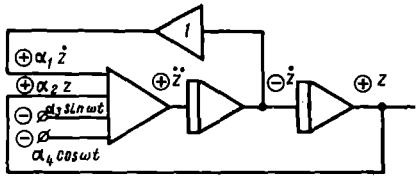


Рис. 7.5. Структурная схема соединения операционных блоков для решения дифференциального уравнения (7.14)

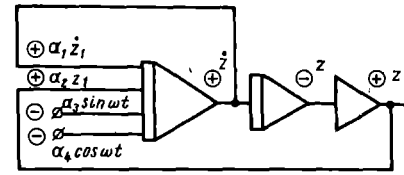


Рис. 7.6. Структурная схема соединения операционных блоков для решения уравнения (7.14) без получения z

Рассмотрим этот метод на примере программирования линейного неоднородного дифференциального уравнения второго порядка с постоянными коэффициентами, которым описываются колебания простого одноосного экипажа при движении по неровному железнодорожному пути (см. рис. 7.4) со случайными неровностями:

$$m\ddot{z}+c\dot{z}+kz=c\dot{\eta}+k\eta. \tag{7.17}$$

Начальные условия: $\dot{z}(0)=0$, $z(0)=0$, η— неровность пути.

В общем случае при ненулевых начальных условиях для их расчета составляется система алгебраических уравнений, которую необходимо предварительно решить. Использовать этот метод эффективно можно только при нулевых начальных условиях [2]

Запишем исходное уравнение в форме

$$\ddot{z}+a_1\dot{z}+a_0z=b_1\dot{\eta}+b_0\eta. \tag{7.18}$$

Составим вспомогательные уравнения

$$\ddot{y}+a_1\dot{y}+a_0y=\eta;\ b_1\dot{y}+b_0y=z. \tag{7.19}$$

Эти уравнения получаются путем замены переменных дифференциального уравнения одной вспомогательной переменной y и приравнивания правых и левых частей исходного дифференциального уравнения левой и правой переменным соответственно [32].

Разрешая первое из вспомогательных уравнений относительно старшей производной и поступая аналогично описанному выше, составим структурную схему (рис. 7.7), на выходах интеграторов которой имеем переменные y и $\dot{y}$, необходимые для получения исходной переменной z на выходе сумматора.

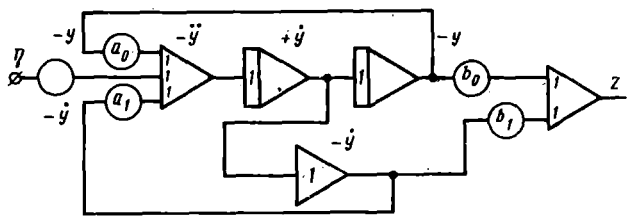


Рис. 7.7. Структурная схема соединения операционных блоков для решения дифференциального уравнения (7.17)

Недостатком метода является потеря информации о значениях первой и второй производных от переменной z, которые иногда, например при решении задач о колебаниях механических систем, желательно иметь.

Рассмотренные методы являются наиболее распространенными. Однако имеются другие, которые отражают характеристики, применяемые в ручном методе расчета, например метод *структурного моделирования*, широко применяемый для моделирования систем автоматического регулирования различными объектами [35].

7.4. Выбор масштабов. Определение коэффициентов передач входов операционных блоков и начальных условий

При решении дифференциальных уравнений на АВМ переменные этого уравнения преобразуются в соответствующие электрические напряжения на выходах и входах операционных блоков и называются машинными переменными. Для того чтобы машинные переменные изменялись по тем же законам, что и переменные в исходном уравнении, они должны быть связаны коэффициентами пропорциональности, которые называются масштабами.

Масштабом называют отношение машинной переменной U к переменной исходного уравнения, например, x, т. е.

$$m_x = U/x. \tag{7.20}$$

Для получения наименьшей ошибки решения из-за влияния дрейфа нуля операционного усилителя, нестабильности его источника питания и других факторов U принимают равным максимально возможному напряжению на выходе усилителя, которое зависит от типа операционного усилителя. Выполнение этого требования означает, что масштаб должен определяться из условия

$$m_x \leqslant U_{max}/|x_{max}|. \tag{7.21}$$

Тогда максимальное значение исходной переменной будет близко совпадать с допустимым значением машинной переменной, не превышая его. Как правило, на АВМ имеются индикаторы, сигнализирующие о превышении машинной переменной допустимого диапазона ее изменения, что означает неправильный выбор масштаба для данной переменной.

Если никаких сведений о максимальных значениях зависимых переменных нет, то масштаб определяется путем пробных решений задачи с соответствующей корректировкой выбранных ранее масштабов.

При решении уравнений на АВМ одной машинной переменной (электрическому напряжению) соответствуют исходные переменные,

определяемые условиями задачи и связанные с напряжениями через масштабы.

Что касается независимой переменной, то в АВМ — это единственная переменная — время. Поэтому какую бы величину и размерность ни имела независимая переменная исследуемой задачи, ей в АВМ всегда будет поставлено в соответствие время. Поскольку продолжительность решения задачи на АВМ можно изменять произвольно, то целесообразно вводить масштаб времени — m_τ. По существу введение масштаба времени вызывается необходимостью воспроизведения исходных процессов без искажения в частотном диапазоне работы АВМ и регистрирующей аппаратуры. Например, широко распространенная АВМ МН-7 имеет частотный диапазон от 0 до 5 Гц и таким образом не может без масштабирования моделировать реальные процессы частоты свыше 5 Гц. Масштаб независимой переменной представляет безразмерную величину, характеризующую соотношение скорости протекания динамических процессов в модели и исходной системе, и равен $m_\tau = \tau/t$, где τ — время протекания процесса в АВМ (машинное время), а t — независимая переменная (время протекания процессов в исходной моделируемой системе). При $m_\tau = 1$ принято говорить, что процесс протекает в натуральном (реальном масштабе) времени, при $m_\tau > 1$ — в замедленном, при $m_\tau < 1$ — в ускоренном времени. При введении масштаба времени изменяется масштаб оси времени полученных результатов решений в m_τ раз.

Максимальное время решения задачи ограничивается допустимым временем интегрирования, определяемым величиной допустимых ошибок интеграторов из-за дрейфа и конечности коэффициента усиления.

Допустимое время интегрирования зависит от типа АВМ и является его характеристикой. Во многих случаях выбираемое время решения из условия удобства наблюдения за процессом может быть принято равным 10—50 с, но не больше допустимого времени интегрирования. При включении в состав модели реальной аппаратуры моделирование может выполняться только в реальном времени.

После выбора масштабов всех переменных, участвующих в решении задачи, переходят к расчету коэффициентов передач операционных блоков. Известны два метода расчета коэффициентов передач: метод перехода к машинным уравнениям [34] и метод почленного масштабирования.

Первый метод громоздок и в настоящее время практически не используется. Второй метод основан на расчете масштабных соотношений и коэффициентов передач для каждого из операционных блоков структурной схемы решения задачи и сводится к приравниванию коэффициентов исходного математического выражения коэффициентам математического выражения, описывающего работу решающих блоков АВМ.

Рассмотрим в качестве примера вывод расчетной формулы для коэффициентов передач одного из операционных блоков: интегросумматора.

Операция интегрирования — суммирования описывается математическим выражением

$$y = \int_0^t \sum_{i=1}^{n} a_i x_i \, dt. \tag{7.22}$$

На АВМ при масштабах переменных m_y и m_{xi} эта операция выполняется интегратором-сумматором и для машинных переменных определяется следующим выражением:

$$U_y = -\frac{1}{m_\tau} \int_0^{\tau_м} \sum_{i=1}^{n} k_i U_{xi} \, d\tau_м, \tag{7.23}$$

где $k_i = \dfrac{1}{R_i C}$ — коэффициент передачи интегратора; m_τ — масштаб времени.

Подставив в формулу (7.22) масштабы, выразим из него U_y и сравним полученное выражение с выражением (7.23). Их равенство возможно при условии

$$k_i = a_i \frac{m_y}{m_{xi} m_\tau}. \tag{7.24}$$

Коэффициент передачи по i-му входу интегратора равен коэффициенту исходного моделируемого уравнения a_i, умноженному на отношение масштаба переменной на выходе блока m_y к масштабу переменной на i-м входе блока и деленному на масштаб времени.

Эту формулу можно преобразовать для расчета коэффициентов передач по входам сумматора или масштабного блока, если отбросить m_τ. При этом необходимо учесть, что коэффициент передачи сумматора по i-му входу равен $k_i = R_0/R_i$ или $k_i = \alpha_i R_0/R_i$, где R_0 — резистор в обратной связи усилителя, R_i — входной резистор по i-му входу, α_i — коэффициент деления делителя по i-му входу. Таким образом получаются простые соотношения, связывающие масштабы, коэффициенты решаемого дифференциального уравнения и параметры элементов (резисторы, емкости) операционных блоков.

Расчет коэффициентов передачи приходится производить несколько раз, постепенно согласовывая масштабы и параметры операционных блоков. Этот процесс обычно называют настройкой модели, а этап — подготовительным.

Расчет начальных условий, если они заданы, производится умножением соответствующего масштаба на начальное значение переменной, для которой заданы начальные значения при $t = 0$. При установке начальных условий на интеграторе учитывается знак переменной.

7.5. Нелинейные операционные блоки

При решении инженерных задач часто возникает необходимость в оценке влияния различных нелинейных характеристик устройств на поведение системы в целом. Во многих случаях моделирование на АВМ является единственной возможностью быстро оценить влияние нелинейности, не прибегая к специальным методам моделирования на ЦВМ. Для решения подобных задач в состав АВМ должны входить блоки, реализующие нелинейные зависимости.

Эти устройства должны воспроизводить заданные нелинейные зависимости с погрешностью не более 1—2% максимального значения, что достаточно для практики. В настоящее время наиболее распространенными считаются диодные блоки нелинейных функций.

Диодные блоки выполняются как универсальными, так и специализированными. В качестве специализированных устройств диодные блоки воспроизводят какую-либо из часто используемых зависимостей: тригонометрическую, степенную или логарифмическую.

Универсальные блоки предназначены для представления в АВМ полученных теоретически или экспериментально нелинейных зависимостей, имеющих конечное число разрывов первого рода. Реализация нелинейных зависимостей диодным блоком основана на кусочно-линейной аппроксимации заданной функции выражением вида

$$y = y_0 + ax + \sum_{i=1}^{n} b_i(x - x_{0i}), \tag{7.25}$$

где

$$b_i = \begin{cases} 0 & \text{при } x \leqslant x_{0i}; \\ b_i = \text{const} & \text{при } x > x_{0i}. \end{cases}$$

Пусть задана нелинейная функция (рис. 7.8). Кусочно-линейная аппроксимация этой функции выполняется четырьмя участками прямых, не выходящих за пределы допусков, изображенных пунктирными линиями, равноотстоящими от основной на величину половины ошибки. Физические переменные нелинейной зависимости заменяются машинными переменными — электрическими напряжениями U_x и U_y.

Для выходной переменной U_y можно записать:

$$U_y = U_0 + k_0 U_x + \sum_{i=1}^{3} k_i(U_x - U_{xi}), \tag{7.26}$$

где

$$k_i = \begin{cases} 0 & \text{при } V_x \leqslant V_{xi}; \\ k_i & \text{при } V_x > V_{xi}. \end{cases}$$

Из формулы видно, что для представления нелинейной функции в АВМ необходимо просуммировать пять напряжений (рис. 7.9). Первым слагаемым является постоянное напряжение U_0, соответствующее значению функции при нулевом входном напряжении U_x. Второе слагаемое представляет собой сигнал, пропорциональ-

ный выходному напряжению, т. е. результат обычного линейного преобразования напряжения. До того момента, пока входное напряжение не достигнет значения U_{x1}, схема является линейной. При увеличении входного напряжения свыше U_{x1} в работу включается нелинейное звено, обеспечивающее формирование третьего слагаемого — сигнала, пропорционального разности $U_x - U_{x1}$ с коэффициентом пропорциональности k_1. Если бы отсутствовало это слагаемое, то на участке $U_{x1} - U_{x2}$ выходное напряжение изменялось бы так, как показано горизонтальной пунктирной линией (см. рис. 7.8). При входном напряжении, превышающем U_{x2}, появляется четвертое слагаемое, пропорциональное разности $U_x - U_{x2}$.

Дальнейшее увеличение входного напряжения вызывает работу третьей нелинейности, которая формирует слагаемое, пропорциональное разности $U_x - U_{x3}$. Таким образом, на сумматор, помимо двух линейных входов, подаются еще три входа с нелинейных элементов *2, 3, 4* (см. рис. 7.9). При воспроизведении другой нелинейной функции может потребоваться большее число нелинейных элементов. В АВМ используются универсальные нелинейные функциональные преобразователи с числом нелинейных элементов 10—12 и погрешностью аппроксимации 1—2%.

Основу любого нелинейного элемента составляет электрическая схема, состоящая из диодов, которые могут переходить в проводящее и непроводящее состояние («открываться и закрываться») в зависимости от величины входного напряжения (рис. 7.10, *а* и *б*). Обычно $R1 = R2$ и входное напряжение, при котором диод открывается, равно $U_н$ и устанавливается потенциометром $R2$. Потенциометр $R3$ регулирует крутизну характеристики элемента (коэффициент k_i). Если потенциал точки A отрицательный, то диод закрыт и напряжение $U_{вых}$ равно нулю. Как только потен-

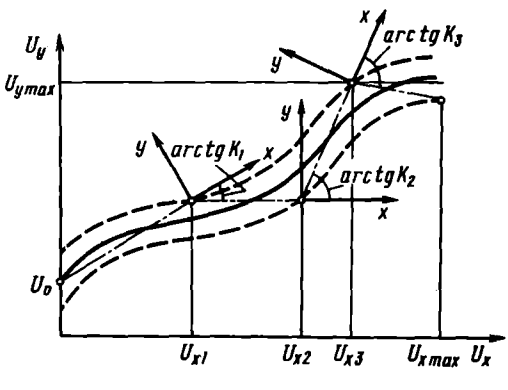


Рис. 7.8. Аппроксимация нелинейной функции $y(x)$

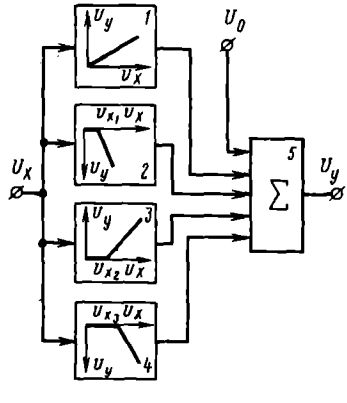


Рис. 7.9. Структурная схема соединения элементов для аппроксимации нелинейной функции $y(x)$

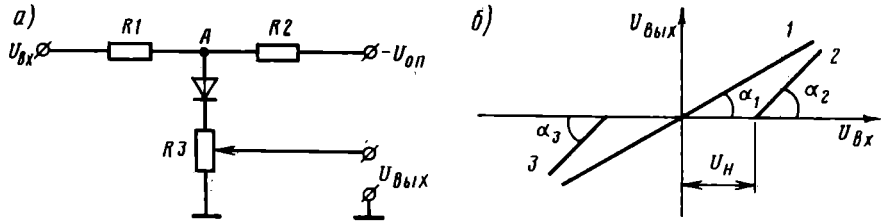


Рис. 7.10. Диодно-резисторная схема (*а*) и ее характерстики (*б*)

циал точки *A* станет больше нуля, то диод открывается и при увеличении $U_{вх}$ пропорционально увеличивается $U_{вых}$, т. е. получается зависимость, обозначенная цифрой *2*. В случае положительного потенциала точки *A* при $U_{вх} \neq 0$ можно получить зависимость *1* Зависимость *3* при помощи диодной схемы можно получить, если изменить полярности входного и опорного напряжений на обратные, поменяв при этом и проводимость диода. Таким образом, комбинируя способ включения диода, полярности входного и опорного напряжений, можно получить воспроизведение нелинейных зависимостей в первом и третьем квадрантах. Аналогичные зависимости, но во втором и четвертом квадрантах можно получить, применив инвертирующий операционный усилитель. Рассмотренные принципы использованы при построении нелинейных функциональных преобразователей.

Диодные схемы широко используются при построении типичных нелинейных зависимостей. На рис. 7.11, *а, б, в* представлен ряд аналоговых схем, которые воспроизводят характеристики таких устройств, как реле, усилители с ограниченным выходным диапазоном, устройства с зазорами, люфтами, гистерезисом и сухим трением. Для моделирования таких характеристик требуются две диодные схемы и один-два операционных усилителя.

Рассмотрим работу одной из наиболее распространенных схем (рис. 7.12, *а*), воспроизводящую характеристику ограничения координат по модулю (рис. 7.12, *в*). Такую характеристику имеют элементы, переходящие при больших уровнях сигналов в режим насыщения (например, операционные магнитные усилители). При нулевом входном напряжении диоды *Д1* и *Д2* закрыты соответственно положительными и отрицательными напряжениями, подаваемыми с потенциометром *R*3 и *R*4, поэтому напряжение на выходе также равно нулю. При увеличении входного напряжения напряжение на выходе уменьшается по линейному закону $U_{вых} = -U_{вх}R_2/R_1$, при этом запирающее напряжение на диоде *Д1* уменьшается. При $U_{вх} = E1$ диод *Д1* открывается и коэффициент передачи операционного усилителя резко падает, поскольку резистор *R2* шунтируется малым сопротивлением открытого диода *Д1* и частью потенциометра *R3*. При дальнейшем увеличении входного напряжения в области отрицательных значений выходное напряжение изменяется

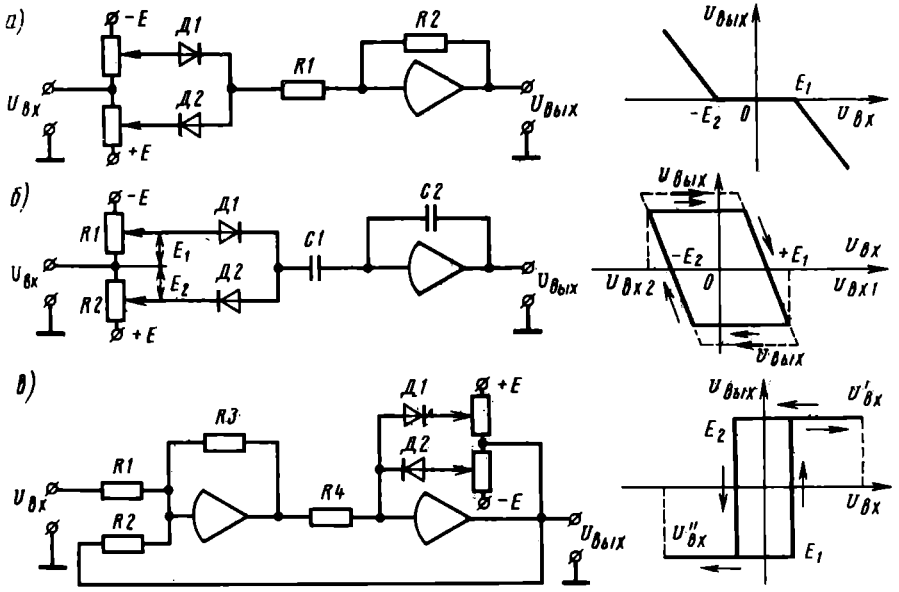


Рис. 7.11. Схемы моделирования типичных нелинейных зависимостей:
а — зазор, зона нечувствительности; *б* — гистерезис; *в* — релейная характеристика

незначительно. Если цепь обратной связи убрать ($R2=\infty$), то характеристика примет вид, показанный на рис. 7.12, *б*, и моделирует характеристику идеального реле или элемента с сухим трением (например, фрикционного гасителя колебаний рессорного подвешивания э. п. с.).

Если при решении задачи требуется осуществить перемножение или деление двух функций, то для этого АВМ использует специальные блоки умножения-деления. Наиболее распространен блок перемножения на квадраторах (нелинейных элементы с квадратичной характеристикой). В основе построения этого блока лежит зависимость $U_xU_y=0{,}25\,[U_x+U_y)^2-(U_x-U_y)^2]$, которая воспроиз-

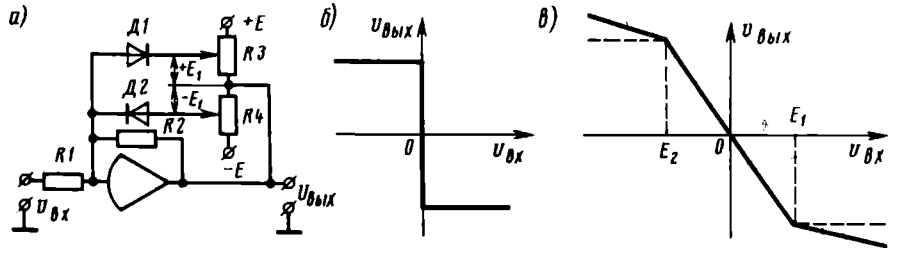


Рис. 7.12. Схема моделирования (*а*) и характеристики элементов, обладающих сухим трением (*в*) и ограничением координат по модулю (*б*)

водится структурной схемой (рис. 7.13) Для обеспечения работы в четырех квадрантах необходимо иметь перемножаемые напряжения с разными знаками. С этой целью в блок перемножения входят инвертирующие операционные усилители, с помощью которых формируются и полуразности входных сигналов y_1 и y_2. Затем эти напряжения возводятся в квадрат и подаются на сумматор, который формирует разность $y^2_1 - y^2_2$, т. е. напряжение, пропорциональное произведению перемножаемых блоком напряжений.

Для выполнения операций деления двух функций времени используют блок перемножения, включенный в обратную связь операционного усилителя *4* (рис. 7.14).

На вход операционного усилителя поступает инвертированное усилителем *1* напряжение $-U_x$ и напряжение с выхода блока умножения. Напряжение на выходе усилителя можно записать в виде

$$U_{вых} = k_0(k_1 U_{вых} U_y - U_x), \qquad (7.27)$$

где k_0— коэффициент усиления операционного усилителя без обратной связи; k_1— коэффициент пропорциональности.

При большом k_0 можно записать $k_1 U_{вых} U_y - U_x = (U_{вых}/k_0) \approx 0$, откуда

$$U_{вых} = U_x/k_1 U_y = kU_x/U_y. \qquad (7.28)$$

Усилители *2* и *3* используются для обеспечения работы множительного блока в четырех квадратах. Недостатками рассмотренных схем умножения и деления являются большие погрешности выполнения этих операций при относительно малых уровнях входных напряжений. Учитывая это, необходимо при решениях задач стремиться использовать эти блоки реже и заботиться о достаточных уровнях напряжений, подаваемых на их входы.

В измерительной технике применяются умножители сигналов, например 525ПС1, 526ПС1 и 525ПС2 (последний имеет встроенный операционный усилитель). В технике преобразования сигналов используется интегральная микросхема 140МА1 (балансный модулятор) — функциональный аналог умножителя сигналов. Указанные

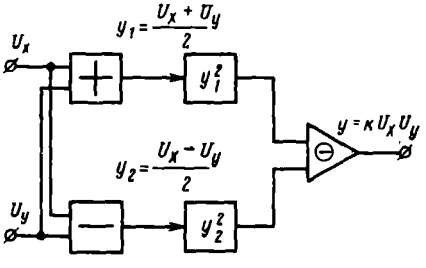


Рис. 7.13. Структурная схема блока умножения

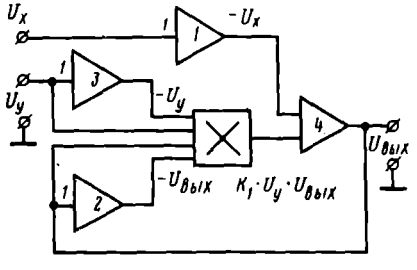


Рис. 7.14. Структурная схема блока деления

интегральные схемы работают с малыми сигналами (до 10 В) и имеют улучшенные характеристики.

В качестве примера рассмотрим моделирование колебаний элементов тяговой передачи электроподвижного состава при учете бокового зазора между зубьями зубчатых колес редуктора. На рис. 7.15 изображена расчетная схема тяговой передачи, где J_1— момент инерции якоря тягового двигателя; J_2— приведенный к валу якоря момент инерции колесной пары и зубчатого колеса редуктора (показанных тонкими линиями); ε — боковой зазор между зубьями, выраженный в единицах угла поворота вала якоря ($\varepsilon = c/r$, где c— боковой зазор, мм; r— радиус делительной окружности шестерни, мм). На расчетной схеме зазор между зубьями моделируется поводком с вилкой, где поводок является как бы зубом большого зубчатого колеса, а вилка — двумя соседними зубьями шестерни, между которыми находится зуб колеса. Колебания масс системы описываются координатами φ_1 и φ_2 и определяются из решения системы двух дифференциальных уравнений второго порядка:

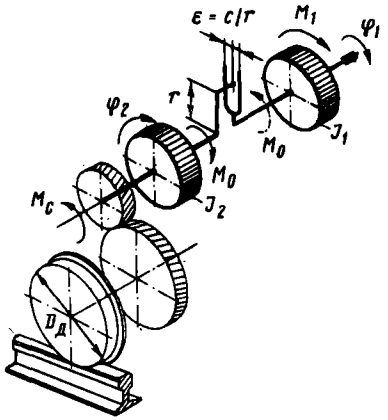


Рис. 7.15. Расчетная схема для исследования динамических процессов в тяговой передаче электровоза ВЛ80 при учете бокового зазора в зубчатом зацеплении

$$J_1\ddot{\varphi} = M_1 - M_0;\ J_2\ddot{\varphi}_2 = M_0 - M_c. \qquad (7.29)$$

Здесь M_1— момент, развиваемый тяговым двигателем.

Момент упругих сил M_0, возникающих при соприкосновении зубьев колеса и шестерни

$$M_0 = \begin{cases} k(\varphi_1 - \varphi_2 - \varepsilon/2) \text{ при } |\varphi_1 - \varphi_2| > \varepsilon/2,\ \varphi_1 > 0; \\ 0 \text{ при } |\varphi_1 - \varphi_2| \leqslant \varepsilon/2; \\ k(\varphi_1 - \varphi_2 + \varepsilon/2) \text{ при } |\varphi_1 - \varphi_2| > \varepsilon/2,\ \varphi_1 < 0. \end{cases} \qquad (7.30)$$

Здесь k— угловая жесткость кинематической цепи от якоря до колесной пары.

Приведенный к валу якоря момент сил сопротивления, возникающих при проскальзывании бандажа относительно рельса, при моделировании можно определить по формуле:

$$M_c = (90\Pi\psi D_б^2/vu^2)\dot{\varphi}_2, \qquad (7.31)$$

где Π— давление колеса на рельс, Н; ψ— предельный по сцеплению коэффициент силы тяги; $D_б$— диаметр бандажа по кругу катания, м; v— скорость движения электровоза, км/ч; u— передаточное число зубчатой пары редуктора.

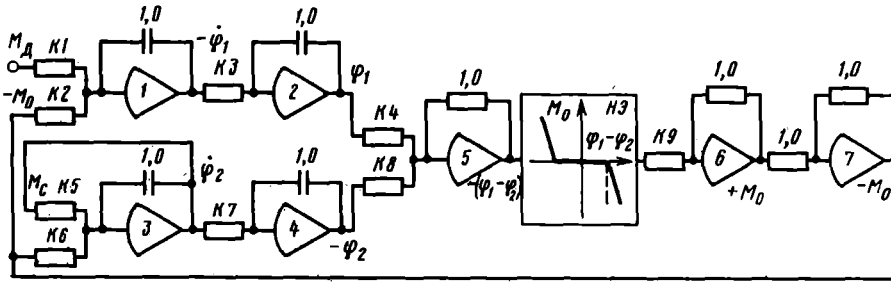


Рис. 7.16. Структурная схема соединений операционных блоков для исследования динамических процессов в тяговой передаче электровоза

Данная формула определяет момент сопротивления в пределах 2 %-ного проскальзывания колеса относительно рельса.

Момент, развиваемый двигателем, можно принять при моделировании постоянным и равным

$$M_д = П\psi D_б/u. \qquad (7.32)$$

Таким образом, когда движение масс происходит в пределах зазора, упругих связей нет, т. е. $M_0=0$, массы движутся независимо в соответствии с уравнениями (7.29). При выборке зазора появляется момент упругих сил и происходит совместное движение масс.

Для решения системы уравнений используется структурная схема соединения операционных блоков. Операционные усилители *1, 2, 3, 4* (рис. 7.16) моделируют колебания масс, описываемые уравнениями (7.29), на усилителе *6* собрана схема типичной нелинейности типа «зазор».

Усилитель *7* используется как инвертор для согласования знаков переменной M_0.

Расчет коэффициентов передач производится по методике, описанной в п. 7.4.

Пример. Рассчитаем коэффициент передачи k_1 по первому входу усилителя *1*. Для выбора масштабов определим приближенно максимальные значения переменных φ_1, φ_2, $\dot{\varphi}_1$, $\dot{\varphi}_2$, $M_д$, M_0, M_c, участвующих в задаче. Зададимся параметрами расчетной схемы применительно к электровозу ВЛ80.

$J_1=72{,}1$ кгм^2; $J_2=28{,}4$ кгм^2; $u=4{,}19$; $z=0{,}1155$ м; $k=10^6$ Нм/рад; $\psi=0{,}23$; $П=11500\cdot 9{,}8=112{,}7\cdot 10^4$ Н; $D_б=1{,}25$ м; $c=2{,}5$ мм. Расчет будем вести для скорости движения электровоза $v=60$ км/ч.

Максимальный угол поворота масс определяется как сумма углов поворота в пределах зазора и за счет деформации упругих участков валов $\varphi_{i\,max}=c/r+(2\div 3)c/r=2{,}5/115{,}5+3\cdot 2{,}5/115{,}5=0{,}087$ рад (коэффициент 3 учитывает динамическое перемещение масс) Максимальный момент определим как момент, предельный по сцеплению, увеличенный в 2—3 раза за счет динамических явлений, развивающихся в системе

$M_i^{max}=3П\psi D_б/u=3\cdot 112{,}7\cdot 10^4\cdot 0{,}23\cdot 1{,}25/4{,}19=23{,}2\cdot 10^4$Нм.

Максимальные угловые скорости можно оценить исходя из зависимости между амплитудами колебаний скоростей и перемещений $\dot{\varphi}_i^{max}=\omega\varphi_i^{max}$, где ω—

наибольшая угловая частота колебаний в системе. Для рассматриваемой механической системы

$$\dot{\varphi}_i^{max} = \omega\varphi_i^{max} = \sqrt{\frac{k}{J_1}}\,\varphi_i^{max} = \sqrt{\frac{10^6}{72{,}1}}\,0{,}087 = 10{,}25 \text{ рад/с}.$$

Определив таким образом максимальные величины переменных, произведем расчет масштабов. Расчет будем вести для максимального значения машинной переменной 10 В:

$$m_{\varphi_i} = \frac{10}{\varphi_i^{max}} = \frac{10}{0{,}087} = 115 \text{ В/рад.}; \quad m_{\dot{\varphi}_i} = \frac{10}{\dot{\varphi}_i^{max}} = \frac{10}{10{,}25} = 0{,}975 \text{ В/рад/с} \quad \text{или}$$

округляя, примем $m_{\dot{\varphi}_i} = 1$ В/рад/с;

$$m_{M_i} = \frac{10}{M_i^{max}} = \frac{10}{23{,}2 \cdot 10^4} = 0{,}43 \cdot 10^{-4} \text{В/Н·м}$$ или, округляя, примем $m_{M_i} = 0{,}5 \cdot 10^4$ В/Н·м.

Масштаб времени можно выбрать, оценив предварительно максимальные частоты колебательных процессов, развивающихся в системе, или оценить приближенно $f = 3\frac{1}{2\pi}\sqrt{\frac{k}{J_1}} = 3/2\pi\sqrt{10^6/72{,}1} = 56{,}2$ Гц, где 3—коэффициент запаса. С учетом того, что АВМ среднего класса допускают моделирование динамических процессов с частотой колебаний не выше 5—10 Гц в реальном масштабе времени, предварительно можно принять масштаб времени $m_\tau = 50/5 = 10$.

Рассчитаем коэффициенты передач операционных блоков. Так, для первого входа интегратора *1* (см. рис. 7.16) согласно формуле (7.24) можно записать $K_1 = a_1 \frac{m_{\dot{\varphi}}}{m_M m_\tau}$, где a_1— коэффициент моделируемого дифференциального уравнения при переменной $M_д$, который равен $1/J_1$. Тогда

$$k_1 = \frac{1}{J_1}\frac{m_{\dot{\varphi}}}{m_M m_\tau} = \frac{1 \cdot 1}{72{,}1 \quad 0{,}5 \cdot 10^{-4} \quad 10} = 27{,}4$$

Полученный коэффициент передачи превышает номинальное значение, равное 10, которое может быть установлено по любому входу, поэтому необходимо корректировать масштабы. Для этого лучше изменить масштаб времени и принять его $m_\tau = 30$, тогда $k_1 = \frac{1}{72{,}1 \cdot 0{,}5 \cdot 10^{-4} \cdot 30} = 9{,}24 < 10$, т. е. этот коэффициент передачи может быть установлен по первому входу. Дальнейший расчет коэффициентов передачи необходимо вести при новом значении $m_\tau = 30$. Например, коэффициент передачи для интегратора *2* $k_3 = \frac{m_\varphi \cdot 1}{m_\varphi m_\tau} = \frac{115 \cdot 1}{1 \cdot 30} = 3{,}83 < 10$. Так как интегрирование ведется с коэффициентом, равным единице, то коэффициент физического уравнения при расчете принимается также равным единице. Расчет остальных коэффициентов передач ведется аналогично, и, если какой-либо из них будет превышать 10, то необходимы корректировка масштабов (решение об этом принимает сам расчетчик) и последующий пересчет всех коэффициентов передач с откорректированными масштабами.

7.6. Установка коэффициентов на операционных блоках и их автоматическая настройка

После набора схемы соединений операционных блоков на них устанавливаются коэффициенты передачи по каждому входу операционного блока. Коэффициенты передачи определяются отношением сопротивления обратной связи к сопротивлению входной цепи. Как правило, во всех АВМ во входных цепях и цепях обратной связи

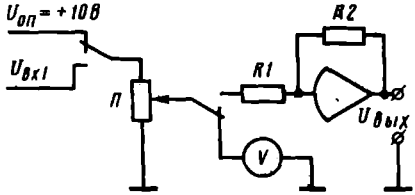


Рис. 7.17 Схема установки коэффициента деления потенциометра при помощи цифрового вольтметра

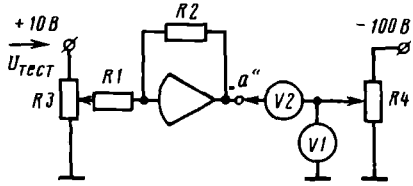


Рис. 7.18. Компенсационная схема установки коэффициентов передач операционных блоков

усилителей применяют резисторы с номинальными сопротивлениями 1,0 МОм или 0,1 МОм. У усилителей, предназначенных для работы в режиме интегрирования, в цепи обратной связи имеется конденсатор емкостью 1,0 мкФ. Такой ограниченный набор резисторов и конденсаторов не позволяет устанавливать коэффициенты передачи, изменяемые в широких пределах. Расширение пределов установки коэффициентов передачи происходит в различных машинах по-разному.

Наиболее распространен способ, в котором на входе применяется схема, состоящая из делителя и входного резистора (см. рис. 7.1, *г*). Настройка коэффициента передачи в этом случае производится вручную. При настройке потенциометра измерительный прибор, например высокоомный цифровой вольтметр, непосредственно подключают к выходу потенциометра *П* (рис. 7.17), на вход которого подано тестовое напряжение $U_{оп} = +10$ В. Перемещая движок потенциометра, изменяют выходное напряжение, устанавливая тем самым коэффициент передачи. После настройки потенциометр подключают к входному резистору.

На АВМ, не имеющих цифровых вольтметров, установка коэффициентов передачи может осуществляться с помощью стрелочных приборов. При этом измерительный прибор необходимо подключать к выходу операционного усилителя, а движок потенциометра соединять с входным резистором *R*. Для повышения точности установки коэффициента может использоваться компенсационный метод измерения (рис. 7.18), принцип которого сводится к следующему. На масштабном блоке требуется установить коэффициент передачи 3,75. На вход регулируемого потенциометра подключают тестовое напряжение +10 В от источника эталонного напряжения. При требуемой установке масштабного коэффициента на выходе усилителя должно установиться напряжение, равное 37,5 В (для усилителей с уровнем напряжения на выходе 50—100 В). Для этого на выходе регулируемого источника эталонных напряжений по вольтметру *V1* делителем *R4* устанавливают напряжение —37,5 В. Затем перемещают движок потенциометра *R3* до получения нулевого показания на стрелочном приборе *V2*. При этом будет установлено

заданное значение коэффициента. После этого отключают тестовое напряжение +10 В и отключают провод «*а*» с выхода операционного блока, подключаемого в схему автоматически или вручную. Подобная операция повторяется с каждым входом операционных блоков.

Если при настройке коэффициентов передач возникает необходимость установки резистора в цепи обратной связи, сопротивление которого превышает требуемое значение, то применяют схему с делителем в цепи обратной связи (рис. 7.19). Предполагая, что потенциал суммирующий точки равен нулю, и пренебрегая сопротивлением потенциометра *R3*, можно написать:

$$U_{вх}\frac{R_2}{R_1+R_2}=-U_{вых}\,\alpha\frac{R_1}{R_1+R_2}. \qquad (7.33)$$

Здесь α— коэффициент деления делителя.

При $R_3<<R_1$ имеем:

$$\frac{U_{вых}}{U_{вх}}=-\frac{(1/\alpha R_2)}{R_1}=-\frac{R_{экв}}{R_1}. \qquad (7.34)$$

Из полученного выражения видно, что введение потенциометра *R3* в цепь обратной связи приводит к увеличению сопротивления обратной связи в $1/\alpha$ раз.

Для ускорения набора задачи некоторые АВМ (например, ЭМУ-10, МН-17) комплектуются устройством автоматической установки потенциометров. Это устройство представляет собой электромеханическую следящую систему. Настройка таких потенциометров осуществляется с пульта управления в специальном режиме работы АВМ; при котором на потенциометры подается опорное напряжение. Подключение к пульту настраиваемого потенциометра осуществляется с помощью адресного селектора. На клавишном устройстве устанавливается численное значение задаваемого коэффициента, преобразуемое в напряжение для того, чтобы следящее устройство отработало рассогласование между задаваемым напряжением и напряжением на потенциометре. Время установки одного коэффициента 5—10 с.

С развитием аналого-цифровых вычислительных машин электромеханические блоки заменяются более быстродействующими электронными. Использование электронных блоков существенно повышает скорость установки коэффициентов.

Электронный блок состоит из входных резисторов R_i (рис. 7.20), которые шунтируются электронными ключами k_i. Электронные ключи управляются регистром P_{ri}. В АВМ вводятся схемы приема (из

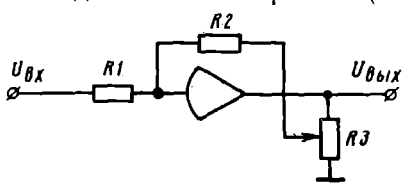


Рис. 7.19. Схема операционного блока с потенциометром в цепи обратной связи

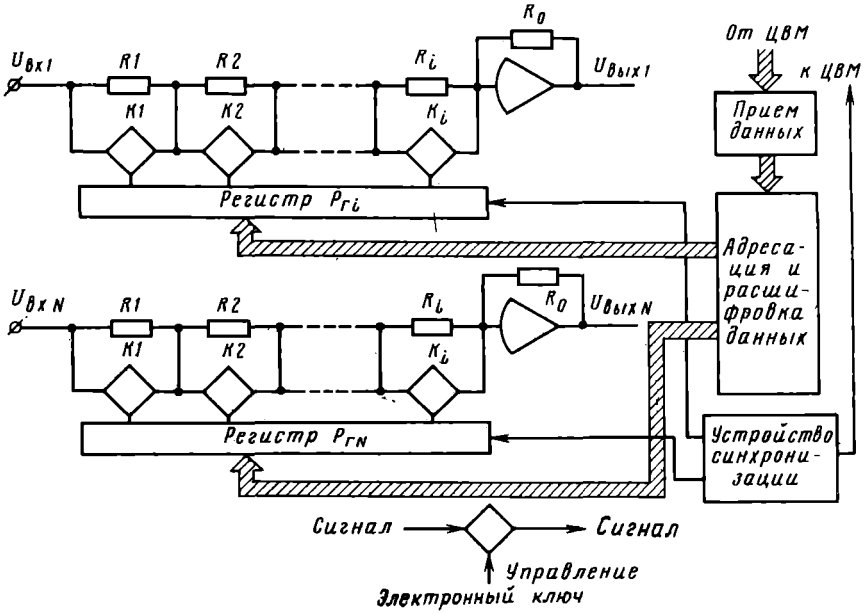


Рис. 7.20. Структурная схема автоматической настройки коэффициентов передач при управлении от ЦВМ

ЦВМ), адресации и преобразования данных, а также схемы синхронизации, позволяющие согласовать работу АВМ и ЦВМ при установке коэффициентов.

Из ЦВМ поступает информация, содержащая адрес электронного блока установки коэффициента и величину коэффициента. Код величины коэффициента поступает в электронный блок управления ключами согласно адресу коэффициента. После установки коэффициента в схему синхронизации поступает сигнал, который несет информацию о завершении установки, что позволяет перейти к установке коэффициента в следующем электронном блоке. Время установки одного коэффициента 1—2 мс при точности установки $(0{,}1 \div 0{,}05)\,\%$.

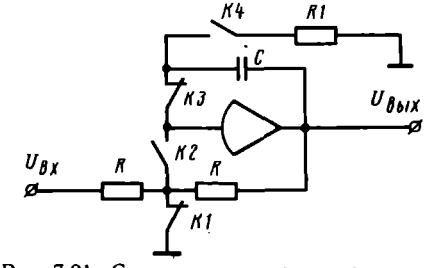


Рис. 7.21. Схема «слежения-хранения» на управляемом автономном интеграторе

Когда АВМ работает с реальной аппаратурой или ЦВМ, требуется выводить во внешние для АВМ цепи сигналы, которые запоминаются на заданное время. Для этого некоторые машины имеют в своем составе специальные буферные устройства, осуществляющие запоминание информации и переда

чу ее к внешним устройствам с задержкой во времени. В качестве этих устройств применяют схемы, получившие название схем «слежения-хранения» [34, 35]. Один из возможных вариантов схемы представлен на рис. 7.21. Она работает в двух режимах.

В режиме «слежение» замкнуты контакты *К2* и *К4*, а контакты *К1* и *К3* разомнуты. Сигнал на выходе операционного усилителя следит за входным и заряжает конденсатор *С*. При переходе в режим «хранение» замыкаются контакты *К1* и *К3*, а *К2* и *К4* размыкаются. Сигнал на выходе усилителя равен напряжению, до которого заряжен конденсатор *С*, образующий цепь обратной связи. Режим «хранение» может включаться как по специальной команде, так и при остановке процесса решения задачи на АВМ. Такую схему можно реализовать на АВМ, в составе которой имеются интеграторы с независимым управлением (ЭМУ-10, АВК-31, АВК-32). Время хранения информации зависит от качества примененного конденсатора, изоляции усилителя от токов утечек, дрейфа нуля усилителя.

7.7. Специализированный аналого-цифровой комплекс

Много вопросов, связанных с исследованием работы и расчетом параметров устройств электроснабжения, может быть успешно решено на сравнительно простых специализированных аналоговых устройствах — моделях. В некоторых случаях использование таких моделей даст даже больший эффект, чем применение универсальных машин. Однако они не позволяют воспроизводить процессы изменения тех или иных параметров режимов работы системы электроснабжения. Такую возможность дает специализированный аналого-цифровой комплекс.

В главе 6 рассмотрены реализуемые на ЭВМ имитационные модели, которые позволили рассматривать нагрузки устройств электроснабжения как случайные или детерминированные процессы.

Такое применение ЭВМ радикально изменило положение в проектных и эксплуатационных расчетах.

Однако из-за сложности описания процессов электроснабжения, их взаимного влияния, динамичности реальных условий разработанные методы расчета все же содержат нежелательные допущения, загрубления. В частности, еще встречает затруднение воспроизведение тяговой нагрузки как случайной функции времени при нелинейных параметрах устройств электроснабжения.

При реализации математических моделей на ЭВМ последовательно производится расчет отдельных мгновенных схем.

Анализ распределения затрат машинного времени показывает, что время, затрачиваемое на расчет мгновенных схем, даже при линейных параметрах составляет до 90 % общего расхода машинного времени на расчет системы. В ряде случаев встречаются нетиповые схемы электроснабжения электрических железных дорог. Поэтому не может существовать универсальной программы, а для каждого нетипового условия необходимо изменять и дополнять программу, что приводит к значительным затратам времени и может оказаться нецелесообразным, особенно при оперативных расчетах[1]

Между тем именно эти вопросы очень просто решаются на специальных моделирующих устройствах. При решении задач на моделирующих установках резко снижается работа по программированию, расчет мгновенных схем протекает практически мгновенно, представление нетиповых схем не представляет сложности, упрощается учет характеристик устройств электроснабжения. Однако эти установки не позволяют исследовать изменение нагрузки во времени и определять параметры, которые зависят от характера протекания этого процесса.

Обобщая сказанное, можно утверждать, что наиболее полное решение задач анализа работы устройств электроснабжения может быть получено при сочетании преимуществ, даваемых цифровой техникой и методами физического моделирования. Такое сочетание может дать гибридное устройство, состоящее из аналоговой и цифровой частей [24, 36].

Аналоговая часть представлена физической моделью системы электроснабжения. Контактная сеть, рельсы и переходное сопротивление заменяются сопротивлением отдельных отрезков пути в выбранном масштабе. Электроподвижной состав представлен отдельными блоками поездов, которые с помощью коммутатора подключаются к тяговой сети и переключаются вдоль участка пути с требуемой скоростью.

В качестве элемента тяговой нагрузки (блока поезда) может служить цифро-аналоговый преобразователь (ЦАП).

Блоки тяговых подстанций, тяговой сети и поездов, представляющие физическую часть устройства, позволяют моделировать в определенном масштабе реальную систему электроснабжения участка электрической железной дороги.

Хранение результатов тяговых расчетов, массивов, определяющих порядок следования поездов разного типа и интервалов между ними, возложено на микроЭВМ.

МикроЭВМ вырабатывает сигналы, необходимые для формирования мгновенных схем с заданной частотой.

Из массива тяговых расчетов последовательно выбираются коды мгновенных значений токов поездов. С помощью блоков сопряжения

[1] Этот недостаток в значительной мере снимается при матричном методе представления тяговой сети.

эти коды заносятся в цифро-аналоговые преобразователи блоков поездов. За время подготовки $t_п$ происходит выдача и запись кодов токов в ЦАП поездов, находящихся на рассматриваемом участке. По тактовому импульсу происходит включение устройства для расчета сформированной мгновенной схемы на период времени расчета $t_р$. Цикл следования мгновенных схем $\Delta t = t_п + t_р$.

От времени $t_р$ зависит требование к быстродействию микроЭВМ. Поэтому для снижения отношения $\Delta t/t_р$ (скважности) при моделировании процесса соблюдается условие $t_п/t_р \leqslant 0{,}1$.

Таким образом, микроЭВМ позволяют представить тяговую нагрузку изменяющейся во времени и в пространстве.

На каждом такте работы микроЭВМ с помощью блока сопряжения, включающего в себя аналого-цифровые преобразователи, получает информацию о токах и напряжениях в элементах аналоговой части и производит их обработку по соответствующим программам.

Расчетное устройство может функционировать как в динамическом режиме, при котором моделируется тяговая нагрузка, изменяющаяся во времени и в пространстве, так и в статическом, когда рассчитываются мгновенные схемы расположения поездов, сформированные на тумблерных регистрах устройства.

Гибридное вычислительное устройство позволяет выполнять все исследования и расчеты с учетом специфических свойств тяговой нагрузки как случайной функции времени.

При дальнейшем развитии устройства на него можно будет возложить решение задач не только в области электроснабжения, но и тяги: проведение тяговых расчетов с учетом фактического изменяющегося при движении поезда напряжения на токоприемнике, определение нагрева тяговых двигателей, выбор оптимального режима движения поезда.

Контрольные вопросы

1. В чем отличие аналоговой вычислительной машины от цифровой?
2. В чем смысл программирования АВМ?
3. Что такое машинная переменная и как она связана с математической (физической) переменной?
4. В чем проявляются ошибки операционных блоков, как их уменьшить? Что такое коэффициент передачи операционного блока?
5. Чем ограничивается длительность решения задачи на какой-либо АВМ?
6. В чем особенность структурных схем для АВМ? Что такое коммутационная схема?
7. Как осуществляется оценка максимальных значений математических (физических) переменных при масштабировании?
8. Почему возникает необходимость согласовывать масштабы зависимых переменных?
9. Каков порядок подготовки задачи для решения с помощью АВМ? Перечислить этапы, дать их характеристику.
10. Составьте общую структурную схему специализированной аналого-цифровой (гибридной) машины.
11. Перечислите преимущества и недостатки специализированной гибридной машины.

СПИСОК ИСПОЛЬЗОВАННОЙ И РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Бусленко Н. П. Моделирование сложных систем. М.: Наука, 1978. 400 с.
2. Марквардт Г Г., Полякова Т В. Алгоритм воспроизведения на ЭВМ процесса изменения тяговой нагрузки при расчете системы энергоснабжения// Тр. Всесоюз. заоч. ин-та инж. ж. -д. трансп. Вып. 65. 1973. с. 95—107.
3. Хьюз Д., Мичтом Дж. Структурный подход к программированию: Пер. с англ. М.: Мир. 1980. 278 с.
4. Микро- и Мини-ЭВМ / Е. П. Балашов, В. Л. Григорьев, Г А. Петров и др.: Учебное пособие. Л.: Энергоавтоматиздат. 1984. 376 с.
5. Айнберг В. Д., Геронимус Ю. В. Основы программирования для единой Системы ЭВМ. М.: Машиностроение. 1980. 336 с.
6. Вычислительная техника в инженерных и экономических расчетах: Учебник/ А. В. Петров, В. Е. Алексеева, М. А. Титов и др.; Под ред. А. В. Петрова. М.: Высшая школа. 1984. 320 с.
7. Коффрон Дж. Технические средства микропроцессорных систем: Практический курс: Пер. с англ. М.: Мир. 1983. 344 с.
8. МикроЭВМ: Пер. с англ./ Под ред. А. Дирксена: М.: Энергоатомиздат. 1982. 328 с.
9. Фрич В. Применение микропроцессоров в системах управления: Пер. с нем. М.: Мир. 1984. 464 с.
10. Вершинин О. Е. Применение микропроцессоров для автоматизации технологических процессов. Л.: Энергоатомиздат. 1986. 208 с.
11. Астанин Л. Ю , Доский Ю Д., Костылев А. А Применение программируемых калькуляторов для инженерных и научных расчетов. Л.: Энергоатомиздат. 1986. 176 с.
12. Дьяконов В. П. Расчет нелинейных и импульсных устройств на программируемых микрокалькуляторах: Справочное пособие. М.: Радио и связь. 1984. 176 с.
13. Трохименко Я. К., Любич Ф. Д Радиотехнические расчеты на микрокалькуляторах: Справочное пособие. М.: Радио и связь. 1988. 256 с.
14. Трохименко Я. К. Программирование микрокалькуляторов «Электроника МК-52». К.: Техника, 1987. 208 с.
15. Грис Д. Наука программирования: Пер. с англ. М.: Мир. 1984. 416 с.
16. Каган Б. М. Электронные вычислительные машины и системы: Учебное пособие М.: Энергоатомиздат. 1985. 552 с.
17. Микропроцессоры: Учебник. В 3-х кн./ Под ред. Л. Н. Преснухина. М.: Высшая школа. 1986.
18. Баранов Л. А., Ерофеев Е. В. Системы автоматического и телемеханического управления электроподвижным составом. М.: Транспорт. 1984. 311 с.
19. Фоминский Г В., Ерофеев Е. В Автоматические устройства для вождения поездов. М.: Транспорт. 1978. 103 с.
20. Вуд А. Микропроцессоры в вопросах и ответах: Пер. с англ.; Под ред. Д. А. Поспелова. М.: Энергоатомиздат. 1985. 184 с.
21. Единая система программной документации. М.: Государственный Комитет по стандартам. 1982. 128 с.
22. Лингер Р. Теория и практика структурного программирования: Пер. с англ. М.: Мир. 1982. 496 с.
23. Бутаков Е. А. Методы создания качественного программного обеспечения ЭВМ. М.: Энергоатомиздат. 1984. 230 с.
24. Марквардт К. Г. Электроснабжение электрифицированных железных дорог. М.: Транспорт. 1982. 528 с.

25. Катца Г Язык Фортран 77: Пер. с англ. М.: Мир. 1982. 208 с.
26. Грунд Ф. Программирование на языке Фортран IV: Пер. с нем. М.: Мир. 1976. 183 с.
27. Фортран ЕС ЭВМ/ З. С. Брич, Д. В. Капилевич, С. Ю. Котик и др. М.: Финансы и статистика. 1985. 287 с.
28. Операционная система ОС ЕС/ В. П. Данилочкин, В. В. Митрофанов, Б. В. Одинцов и др.: Справочное пособие. М.: Статистика. 1988.
29. Грунд Ф Принципы операционной системы ОС ЕС: Пер. с нем. М.: Финансы и статистика. 1984. 189 с.
30. Марквардт Г Г Исходные положения по созданию математической модели процесса работы устройств энергоснабжения электрических железных дорог//Всесоюз. заоч. ин-та инж. ж.-д трансп. вып. 37. 199. С. 46—51.
31. Железнов Д. Ф., Провезенцев Н. Н. Расчет на ЭЦВМ системы энергоснабжения по заданным межпоездным интервалам//Тр. Всесоюз. заоч. ин-та инж. ж.-д. трансп. Вып. 302. 1969. С. 154—167.
32. Быкодоров А. Л., Доманский В. П. Моделирование системы энергоснабжения со сложной конфигурацией тяговой сети//Тр. Всесоюз. заоч. ин-та инж. ж.-д. трансп. 1981. Вып. 115. С. 67—75.
33. Шиловская Р В. Математическая модель расчета системы энергоснабжения метрополитена на ЭВМ// Тр. Всесоюз. заоч. ин-та инж. ж.-д. трансп. Вып. 38. 169. С 36—45.
34. Урмаев А. С. Основы моделирования на аналоговых вычислительных машинах. М.: Наука. 1974. 320 с.
35. Горбацевич Е. Д., Левинзон Ф. Ф. Аналоговое моделирование систем управления. М.: Наука. 1984. 304 с.
36. Ильяшенко В. П. Специализированное вычислительное устройство для анализа и проверки режимов работы системы электроснабжения/Тр. Всесоюз. заоч. ин-та инж. ж.-д. трансп. Вып. 107. С. 84—87.

ОГЛАВЛЕНИЕ

Учебник

АНДРЕЕВ ВАЛЕРИЙ ВАСИЛЬЕВИЧ,
КУЛИКОВ ПАВЕЛ БОРИСОВИЧ,
МАРКВАРДТ ГЕОРГИЙ ГУСТАВОВИЧ,
РЫБНИКОВ ЕВГЕНИЙ КОНСТАНТИНОВИЧ,
ФЕОКТИСТОВ ВАЛЕРИЙ ПАВЛОВИЧ

Вычислительная и микропроцессорная техника в устройствах электрических железных дорог

Технические редакторы *М. А. Шуйская, Л. А. Кульбачинская*
Корректор-вычитчик *Е. А. Котляр*
Корректор *Т. А. Мельникова*
ИБ № 3985

Сдано в набор 16.12.88. Подписано в печать 09.11.89. Т-17430. Формат 60×88/16. Бум. офсетная № 2. Гарнитура литературная. Офсетная печать. Усл. печ. л. 17,64. Усл. кр.-отт. 17,88. Уч.-изд. л. 20,79. Тираж 9500 экз. Заказ 1778. Цена 75 коп.
Изд. № 1-1-1/5 № 4471
Ордена «Знак Почета», издательство «ТРАНСПОРТ», 103064, Москва, Басманный туп., 6а

Московская типография № 4 Союзполиграфпрома при Государственном комитете СССР по печати. 129041, Москва, Б. Переяславская, 46.

75 коп.