Л.Г. Осетинский
М.Г. Осетинский
А.Н. Писаревский

# focal

# ФОКАЛ

## ДЛЯ МИКРО- И МИНИ-КОМПЬЮТЕРОВ

машиностроение

**Л. Г. Осетинский,**
**М. Г. Осетинский,**
**А. Н. Писаревский**

# ФОКАЛ
## для микро- и мини-компьютеров

**Ленинград**
**«Машиностроение»**
**Ленинградское отделение**
**1988**

**Сканирование: Wesha**
**Оцифровка, верстка:**
**Andrew Samara**

ББК 32.97
О-72
УДК 681.3.06

Р е ц е н з е н т М. Л. Шапиро

**Осетинский Л. Г.** и др.

О-72 ФОКАЛ для микро- и мини-компьютеров/
Л. Г. Осетинский, М. Г. Осетинский, А. Н. Писаревский. — Л.: Машиностроение. Ленингр. отд-ние, 1988. — 303 с.

ISBN 5-217-00323-5

В книге описан диалоговый язык ФОКАЛ, входящий в программное обеспечение (ПО) микро- и мини-компьютеров ДВК-2, «Электроника-60», СМ-4 и др. Рассмотрено ПО, созданное на базе ФОКАЛа и используемое для программирования измерительных комплексов и автоматизированного проектирования; раскрыты особенности реализации ФОКАЛа на микроЭВМ индивидуального пользования «Электроника БК-0010»; приведены примеры использования ФОКАЛа для инженерных расчетов, систем невычислительного назначения и разработки игровых программ.

Для ИТР, использующих вычислительную технику в различных отраслях народного хозяйства.

О $\frac{2405000000\text{-}963}{035\ (01)\text{-}88}$ **КБ-49-18—87** **ББК 32.97**

ISBN 5-217-00323-5 

# ПРЕДИСЛОВИЕ

Важнейшее условие реализации программы ускорения, выдвинутой XXVII съездом КПСС, состоит в широком развитии и внедрении средств вычислительной техники во всех областях народного хозяйства. При этом компьютер становится необходимым атрибутом всей жизни современного общества, что неизбежно выдвигает проблему ликвидации компьютерной неграмотности населения, прежде всего учащихся и специалистов различного профиля.

Какой же смысл вкладывается в понятие компьютерной грамотности? Можно говорить о трех ее уровнях.

Первый уровень включает простейшие навыки работы с компьютером и внешними носителями информации, элементарные знания о функциональной организации компьютера и программного обеспечения. На этом уровне компьютерной грамотности даже неподготовленный человек может затратить на освоение компьютера от часа до нескольких дней, в зависимости от его сложности и конкретной сферы применения.

Предлагаемая книга не ориентирована на тот круг пользователей, которые намерены ограничиться первым уровнем знаний, так как использование прикладных программ и игры с компьютером не требуют знания основ программирования.

Второй уровень включает знакомство с основными принципами работы на компьютере и изучение основ программирования на одном из языков программирования высокого уровня (например, на языке ФОКАЛ). Освоение этого материала позволяет пользователю разрабатывать простые программы и понимать их. В той степени, в которой компьютер становится неотъемлемой частью современной цивилизации (как автомобиль, электричество, радио и т. д.), такая степень грамотности может рассматриваться как естественная часть общечеловеческой культуры, точно так же, как и элементарная математика, физика, география и т. д.

Третий уровень требует более глубокого понимания принципов организации и функционирования компьютера, знания машинного языка и системных средств, умения эффективно использовать все возможности языка программирования и микроЭВМ, т. е.

подразумевает профессиональное освоение средств вычислительной техники.

Настоящая книга адресована в первую очередь тем пользователям, которые ориентируются на второй уровень компьютерной грамотности. Наряду с языком ФОКАЛ и его реализациями как на промышленно выпускаемом бытовом персональном компьютере «Электроника БК-0010», так и на других мини-ЭВМ, в книге подробно рассмотрены принципы разработки прикладных программ и методология программирования на языке ФОКАЛ. Изложение материала иллюстрируется многочисленными примерами, и его изучение будет существенно эффективнее, если «под рукой» окажется компьютер.

Язык ФОКАЛ один из немногих языков программирования, на базе которых построено преподавание основ вычислительной техники и информатики в школах и ПТУ, а также строятся курсы обучения основам программирования в ряде вузов.

Интерпретатор ФОКАЛа, расширенный графическими возможностями, вошел в состав программного обеспечения системы машинного проектирования «Кулон», широко используемой для решения задач автоматизированного проектирования и автоматизации производства. Для профессионально подготовленного читателя в книге дается достаточно полная информация по аппаратной и системнопрограммной организации персонального компьютера «Электроника БК-0010», описываются реализация языка ФОКАЛ на нем и рассматривается использование ФОКАЛа в качестве инструментального средства разработки графических пакетов прикладных программ автоматизированного проектирования изделий электронной техники.

Авторы надеются, что книга окажется полезной всем, кто хочет расширить свои знания в сфере программирования и вычислительной техники, и заранее благодарны за критические замечания в адрес предлагаемой книги.

ГЛАВА 1

# ВВЕДЕНИЕ В АРХИТЕКТУРУ МИКРОКОМПЬЮТЕРОВ

## 1.1. МИКРОЭВМ И ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ

*МикроЭВМ (микрокомпьютер)* — это небольшая ЭВМ, основой которой служит *микропроцессор,* т. е. центральный процессор, реализованный в виде одной большой интегральной микросхемы. Важными характеристиками микроЭВМ служат габаритные размеры, стоимость и область применения. Низкая стоимость микроЭВМ позволяет ориентировать их на решение какой-то одной задачи или группы взаимосвязанных задач, например, на управление станком или автомобильным двигателем, стиральной машиной или кухонной плитой. Малые габаритные размеры микроЭВМ позволяют встраивать ее в измерительные приборы и устройства управления, швейные и пишущие машинки, фотоаппараты, кино- и видеокамеры.

*Персональный компьютер* — это универсальная микроЭВМ, ориентированная на индивидуального пользователя, отличающаяся *малыми габаритными* размерами, *высокой надежностью, простотой эксплуатации* и достаточно широкими функциональными возможностями. В зависимости от назначения и реализации персональные компьютеры могут быть *профессиональными, учебными* и *бытовыми.*

Персональный компьютер (особенно учебный или бытовой, предназначенный в первую очередь для малоопытных пользователей), работает, по крайней мере, с. одним из языков высокого уровня (таким как ФОКАЛ, БЕЙСИК или ПАСКАЛЬ) и имеет встроенное программное обеспечение, облегчающее взаимодействие пользователя с компьютером в режиме диалога.

МикроЭВМ «Электроника БК-0010», аппаратной организации и программному обеспечению которой посвящена значительная часть книги, представляет собой первый отечественный бытовой персональный микрокомпьютер.

## 1.2. АППАРАТНАЯ ОРГАНИЗАЦИЯ МИКРОКОМПЬЮТЕРА

В самом упрощенном виде микрокомпьютер можно представить в виде схемы (рис. 1.1), из которой видно, что микрокомпьютер состоит из четырех основных частей:

1) *устройства ввода,* позволяющего вводить инструкции и (или) данные в микрокомпьютер; основное устройство ввода для персональных компьютеров — *клавиатура;*

2) *микропроцессора,* предназначенного для управления работой всей системы и являющегося основным узлом микрокомпьютера;

3) *памяти,* в которой хранятся данные, результаты их обработки компьютером и инструкции, по которым эта обработка выполняется;

4) *устройства вывода,* позволяющего получить результаты обработки информации (чаще всего выходное устройство для микрокомпьютера — экран дисплея или телевизора, хотя для этого может использоваться и периферийная память в виде кассетных накопителей на магнитной ленте или на магнитных дисках).

Микропроцессор и память. Вычислительная мощность микропроцессора определяется двумя основными характеристиками — *размером его информационного слова* (ячеек памяти, внутренних регистров), определяющим число бит, которые могут быть переданы параллельно (одновременно) по шине данных, и *быстродействием* (частотой электронного тактового генератора). Наиболее распространены 8-ми и 16-битовые устройства. Информационное слово узкофункциональных встроенных микропроцессоров содержит 4 бита. Для профессиональных микрокомпьютеров все большее распространение получают 32-разрядные микропроцессоры. Тактовая частота лежит в диапазоне от 1 до 20 МГц (тактовая частота микроЭВМ «Электроника БК-0010» — 3 МГц).

Несмотря на то что за последнее время появились интегральные схемы, в которых на одном кристалле реализованы микропроцессор и память для записи программ и данных, принято рассматривать память как отдельный компонент микрокомпьютера. При этом различают память двух типов.

Во-первых, память, предназначенную для считывания и записи информации и реализованную на так называемом *запоминающем устройстве с произвольной выборкой* (ЗУПВ). В ЗУПВ хранится


**Рис. 1.1. Простейшая схема микрокомпьютера**

информация, которая изменяется в процессе работы микрокомпьютера, причем это могут быть как программный код, так и данные. Однако при отключении питания вся находящаяся в ЗУПВ информация исчезает. Для «спасения» находящейся в такой памяти информации и (или) ее загрузки в память необходимо использовать *внешние запоминающие устройства,* например кассетный магнитофон. Персональные компьютеры, как правило, имеют память от 16 до 64 Кбайт (К = 1024).

Во-вторых, в микрокомпьютере используется память, информация в которой сохраняется и при выключенном питании. Такая память предназначена только для чтения и реализована на *постоянном запоминающем устройстве* (ПЗУ). В микрокомпьютерах, ориентированных на конкретные приложения (например управление стиральной машиной), ПЗУ может хранить соответствующую прикладную программу. В универсальных микроЭВМ в ПЗУ записывают базовую операционную систему, компоненты которой «запускают» компьютер после включения питания, обеспечивают взаимодействие пользователя с компьютером, поддерживают соответствующую систему программирования и тестовое программное обеспечение.

Устройства ввода и вывода. Внешняя память. Основным устройством ввода информации в микрокомпьютер служит клавиатура или терминал, а дополнительными — периферийные устройства внешней памяти, такие, как накопители на магнитных кассетах (магнитная пленка) и гибких магнитных дисках (в профессиональных персональных компьютерах используют винчестерские диски). Емкость современных гибких дисков, как и кассет, равна 125—500 Кбайт, а винчестерских дисков существенно больше — до 80 Мбайт (миллионов байт).

В персональном компьютере вывод производится, как правило, на дисплей, выполненный на базе электроннолучевой трубки. Он либо поставляется в комплекте с микрокомпьютером, либо (обычно для бытовых микрокомпьютеров) в качестве растрового дисплея используется обычный телевизионный приемник. Кроме

того, информация может записываться на перечисленные устройства внешней памяти или выводиться на различные печатающие устройства — термопринтеры, точечно-матричные или лепестковые принтеры, входящие в комплект профессиональных микрокомпьютеров. Электронно-лучевые трубки в настоящее время начинают заменяться индикаторными панелями на жидких кристаллах.

## 1.3. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Как правило, пользователю (особенно программисту не профессионалу) не приходится иметь дело с аппаратурой компьютера. Связующим звеном между аппаратным обеспечением и пользователем служат программы самого разного уровня сложности, которые все вместе образуют *программное обеспечение* компьютера. Ту часть программного обеспечения, которая наиболее тесно связана с аппаратной частью компьютера, называют по-разному: *операционной системой, монитором, супервизором* и т. д. Например, в бытовом персональном микрокомпьютере «Электроника БК-0010» встроенное в ПЗУ программное обеспечение *(монитор)* предоставляет пользователю возможность работы с клавиатурой и устройством отображения информации, накопителем на кассетной магнитной ленте и портом ввода—вывода. Пользователю достаточно задать один единственный оператор, такой как «сохранить информацию в файле на магнитной ленте», и одна из программ монитора *(драйвер* магнитной ленты) обеспечит формирование всей последовательности команд микропроцессора и сигналов, необходимых для выполнения этой директивы.

Другая часть программного обеспечения — различные *прикладные программы.* К ним относятся и программы, разрабатываемые пользователем для своих конкретных задач, и *программы общего назначения,* такие как трансляторы, редакторы текстов и т. п., облегчающие и упрощающие подготовку программ пользователя.

Необходимо отчетливо представлять, что программа задает процесс или способ обработки информации, выраженный в инструкциях *машинного языка.* Подготовка программы на таком *машинно-ориентированном языке* очень утомительна, малопроизводительна и почти всегда приводит к многочисленным ошибкам. Поэтому в состав программного обеспечения

компьютеров входят программы трансляции (или просто *трансляторы*) для преобразования программ, написанных на языке Ассемблера или на языке высокого уровня, таком как ФОКАЛ или ФОРТРАН и т. п., в машинный код.

Существует две категории программ трансляции — *интерпретаторы и компиляторы.*

*Компилятор* переводит программу, написанную на языке высокого уровня, в эквивалентную рабочую программу в машинных кодах, которая после завершения преобразования запускается в работу.

Программа, написанная на интерпретирующем языке, вносится в память в виде символьных команд высокого уровня. В процессе выполнения этой программы другая программа (упомянутый выше *интерпретатор)* переводит все команды одну за другой в инструкции машинного языка, каждая из которых немедленно обрабатывается процессором. При этом интерпретатор должен находиться в памяти в течение всего времени выполнения программы пользователя.

Преимущество *интерпретатора* заключается в том, что он обеспечивает возможность диалогового режима подготовки, отладки и выполнения программы, записанной на языке высокого уровня. С другой стороны, программа, транслированная с помощью компилятора, выполняется значительно быстрее, поскольку она целиком уже переведена в машинный код.

В программное обеспечение микрокомпьютеров входят как языки интерпретирующего типа (БЕЙСИК, ФОКАЛ), так и языки, программы на которых обрабатываются компилятором (ФОРТРАН, ПАСКАЛЬ, СИ и др.). При работе с персональными (особенно бытовыми) компьютерами наиболее широко применяются языки интерпретирующего типа БЕЙСИК и ФОКАЛ, так как их проще всего освоить неподготовленному пользователю.

Написание программы на одном из этих языков состоит из нескольких этапов, включающих постановку задачи, разработку алгоритма ее решения и кодирования алгоритма на выбранном языке. Постановка задачи обычно не рассматривается как часть процесса программирования, однако точная формулировка задачи необходима для успешной реализации программы.

В идеальном случае работа программиста завершается составлением программы, так как выполнение этой программы на компьютере должно привести к требуемым результатам. Фактически, из-за сложности задач и ограниченности человеческих

возможностей, исходная программа, как правило, содержит ошибки. Программист вовлекается в цикл исправления и проверки своей программы до тех пор пока не убедиться, что программа работает правильно. Этот процесс нахождения и исправления ошибок в программе называется ее *отладкой*. При трансляции программы транслятор (компилятор или интерпретатор) выявляет ошибки, допущенные при записи программы и сообщает о них, указывая их тип и то место в программе, где они обнаружены. Такие ошибки принято называть *ошибками времени трансляции.*

Ситуации, приводящие к возникновению ошибки, могут возникать и при выполнении программы, например деление на нуль или извлечение квадратного корня из отрицательного числа. Такие ошибки называются *ошибками времени выполнения.*

И, наконец, программа, не имеющая ошибок трансляции и выполнения, может и не дать верные результаты из-за так называемых *логических ошибок* в выборе или реализации алгоритма.

Таким образом, отладка должна обеспечить устранение из программы ошибок трансляции, выполнения и логических ошибок.

Реализация языка программирования. Одно из преимуществ применения языка высокого уровня, такого как ФОКАЛ или ПАСКАЛЬ, — возможность использовать программы, написанные на этом языке, на любом компьютере, для которого разработан компилятор и (или) интерпретатор с этого языка. Именно это имеется в виду, когда говорят о *машинной независимости* программ на языках высокого уровня. Полная машинная независимость встречается на практике редко, но усилия, которые приходится затрачивать на перенос таких программ с компьютера одного типа на другой незначительны по сравнению с затратами на написание этих программ.

Транслятор с данного языка и для определенного типа компьютеров называется *реализацией языка.* Иногда реализация накладывает программные ограничения в дополнение к тем, которые обусловлены самим языком. Следовательно, в некоторых своих аспектах язык может быть зависим от реализации. При разработке программ специфические требования реализации необходимо соблюдать точно так же, как правила самого языка.

ГЛАВА 2

# ЯЗЫК ПРОГРАММИРОВАНИЯ ФОКАЛ

## 2.1. ОБЩАЯ ХАРАКТЕРИСТИКА ЯЗЫКА

Как уже упоминалось (гл. 1), ФОКАЛ является одним из диалоговых языков, входящих в программное обеспечение ЭВМ и реализованных на целом ряде микро- и мини-компьютеров (ДВК-2, «Электроника БК-0010», СМ-4 и т. п.). Как язык программирования ФОКАЛ характеризуется следующими свойствами: простотой общения и изучения; полным набором средств, необходимым для изучения основ программирования; широкими функциональными возможностями при небольшом объеме интерпретатора языка (4 Кслов); эффективными средствами составления и отладки программ, развитой диагностикой ошибок; наличием библиотеки элементарных математических функций и функций для работы со стандартным периферийным оборудованием; компактностью программы пользователя.

**Режимы работы.** Интерпретатор ФОКАЛа может работать в двух режимах: диалоговом и программном.

*Диалоговый режим* — такой режим общения пользователя с интерпретатором ФОКАЛа, при котором ввод любой информации с терминала вызывает немедленную реакцию компьютера. Для интерпретатора ФОКАЛа наименьшей единицей информации, вызывающей немедленное ответное действие компьютера, является строка. Строка может содержать любые разрешенные символы языка и должна заканчиваться символом, указывающим на завершение ввода информации. В дальнейшем мы будем обозначать этот символ как <ВК>, хотя его начертание на клавиатурах различных компьютеров может отличаться от принятого.

*Программный режим* характеризуется наличием у строки номера. Такая строка *(строка программы)* не будет выполняться сразу после ввода символа <ВК>, а будет просто запоминаться интерпретатором ФОКАЛа.

Длина строки, т. е. общее число символов, которое можно записать в строке, включая пробелы и номер строки, зависит от реализации. Таким образом, в ФОКАЛе используются строки двух

типов:

```
<строка>::= <директивная строка> | <строка
      программы>.
<директивная строка)::= <оператор>
      {;<оператор>} <ВК>.
<строка программы>::= <номер строки программы>
      <оператор> {;<оператор>} <ВК>.
```

(2.1)

Как видно из этого определения, строка состоит из операторов, разделенных точками с запятыми, завершается символом <ВК> и может иметь номер.

Основное различие между директивной строкой и строкой программы состоит в том, что директивная строка (т. е. все входящие в нее операторы) выполнится интерпретатором ФОКАЛа сразу же после ввода символа <ВК>. Например, если пользователь введет директивную строку TYPE «Фокал» <ВК>, то сразу же после ввода символа завершения строки будет выполнен оператор печати TYPE и выведется строка ФОКАЛ (без кавычек!):

```
*TYPE "ФОКАЛ" <ВК>
ФОКАЛ*
```

(2.2)

Директивная строка не запоминается интерпретатором ФОКАЛа, т. е. для того, чтобы еще раз вывести слово ФОКАЛ необходимо вновь ввести строку (2.2). Однако, если в директивной строке используются какие-либо *переменные*, то их значения сохранятся в *таблице переменных* и они могут использоваться в других директивных строках или строках программы. Например, первая директивная строка *SET К=1 <ВК> присвоит значение 1 переменной *К*, а вторая —

```
TYPE К <ВК>
   1.0000*
```

выведет его на терминал.

Совокупность строк программы образует *программу пользователя*

**<программа>::= <строка программы> {<строка программы>}.**

(2.3)

Таким образом, программа, написанная на языке ФОКАЛ, состоит из некоторого числа строк (одной и более) программы. Размеры программы ограничиваются диапазоном изменения номеров строк программы, объемом оперативной памяти компьютера, выделяемой для хранения программы, и, в некоторой степени, организацией самой программы.

Нумерация строк. *Номер строки программы* является составным номером вида *mm.nn,* где первую часть номера (*mm*) называют *номером группы* строк, а вторую часть (*nn*) — *номером строки* в группе:

**<номер строки программы>::=**
**<номер группы>.<номер строки>.** (2.4)

Для задания номеров строк программы можно использовать любые номера от 01.01 до 99.99 (с шагом 0.01), за исключением тех, которые оканчиваются нулем (т. е., например, такие, как 04.00 или 81.00), так как они используются в некоторых операторах для указания всей группы строк. В номере строки программы можно опускать незначащий ноль в номере группы и стоящий справа ноль в номере строки, хотя надо иметь в виду, что номер 2.1 эквивалентен номеру 02.10, но никак не 02.01. Для того чтобы выполнить строку программы (или всю программу), необходимо ввести директивную строку GOTO *mm.nn,* которая указывает интерпретатору ФОКАЛа, что программа должна быть запущена со строки с номером *mm.nn.* Например, если была введена строка

```
*2.1 TYPE 7 <ВК>
*
```

то для ее выполнения необходимо ввести директиву GOTO 2.1 <ВК>. В результате выполнения этой строки будет выведено число 7.

Рассмотрим пример простейшей программы на языке ФОКАЛ:

```
*1.10 SET I=3 <ВК>
*1.20 SET J=5 <ВК>
*1.30 SET K=J-I <ВК>
*1.40 TYPE I,J,к <ВК>
*
```

Эта программа присваивает (оператор SET) переменной *I* значение 3, переменной *J* — значение 5, переменной *K* — значение разности значений *J* и *I*, а потом выводит (оператор TYPE) эти значения. Символ * выводится интерпретатором ФОКАЛа. Хотя номера строк возрастают в приведенном примере с шагом 0.10, номера группы и номера строк могут выбираться произвольно, а шаг не обязательно должен быть постоянным во всей программе. Например, эту программу можно записать так

```
*1.10 SET I=3 <ВК>
*2.10 SET J=5 <ВК>
*41.01 SET K=J-I <ВК>
*63.17 TYPE I,J,к <ВК>
*
```

или так

```
*1.01 SET I=3 <ВК>
*1.02 SET J=5 <ВК>
*1.03 SET K=J-I <ВК>
*1.04 TYPE I,J,к <ВК>
*
```

(2.5)

Интервалы между номерами строк позволяют при необходимости вставлять в программу дополнительные строки. Именно поэтому настоятельно рекомендуется не записывать программы с непрерывной нумерацией строк, как в примере (2.5).

**Выполнение программы.** При запуске программы строки выполняются последовательно в порядке возрастания номеров групп, а в пределах одной группы — в порядке возрастания номеров строк в группе. При подготовке программы пользователь может вводить строки программы в произвольном порядке, так как интерпретатор производит сортировку строк программы в порядке возрастания их номеров. Это значительно упрощает отладку и

редактирование программ.

**Операторы и их запись в программе.** Как следует из определения (2.1), если в строке записываются несколько операторов, то они отделяются друг от друга *точкой с запятой.*

Каждый оператор начинается с английского слова (*имени оператора*), по которому интерпретатор ФОКАЛа отличает операторы друг от друга. Синтаксис языка ФОКАЛ требует, чтобы за именем оператора шел, по крайней мере, один пробел. В отличие от большинства языков программирования (таких как ФОРТРАН, ПЛ-1, ПАСКАЛЬ) имена операторов не являются стандартными идентификаторами ФОКАЛа. Интерпретатор распознает имя оператора по месту его появления в строке, а именно, именами операторов считаются идентификаторы, идущие непосредственно за следующими символами: *звездочкой,* которую выводит интерпретатор ФОКАЛа; *номером строки программы*; *точкой с запятой.* Таким образом, в строке

```
*2.07 IF (A+IF)2.1,1.03,2.17 <ВК>
```

первый идентификатор IF будет воспринят интерпретатором как имя условного оператора, а второй идентификатор IF будет рассматриваться как простая переменная, значение которой будет использовано для вычисления условного выражения.

Все операторы языка ФОКАЛ можно (в достаточной мере условно) сгруппировать по их функциональному назначению следующим образом:

```
<оператор>::= <пустой оператор> | <оператор
присваивания> | <оператор ввода-вывода> |
<оператор управления> | <оператор отладки>.
```

(2.6)

Синтаксис и назначение операторов (число которых различно в разных реализациях) рассматриваются в следующих параграфах. Единственным оператором, который будет рассмотрен в этом разделе, является *пустой оператор*, т. е. оператор, который не содержит никаких символов и указывает на отсутствие действий. Пустой оператор может иметь, например, следующий вид (назначение пустого оператора станет понятно позже): *1.35 <ВК>

Краткий справочник операторов и функций языка приведен в

приложении 1.

**Запись текста программы.** При подготовке текста программы программист имеет практически неограниченную *свободу в расположении* имен операторов, идентификаторов и чисел в строке, поскольку пробелы, находящиеся в строке, не влияют на работу программы. Для более компактной и быстрой записи программы имя оператора может заменяться *одной начальной буквой*, поскольку интерпретатор различает операторы по первому символу имени оператора. Например, вместо оператора SET достаточно написать символ S, вместо оператора TYPE — символ T и т. д. Это уменьшает объем оперативной памяти, который занимается программой, и до некоторой степени увеличивает скорость ее выполнения.

Поскольку интерпретатор различает операторы по первому символу имени оператора, то все символы (кроме первого) до пробела будут игнорироваться. Это означает, что операторы TYPEBACЯ 5, TYPE 5 и T 5 выполнятся одинаково.

**Сравнение языков ФОКАЛа и БЕЙСИКа.** ФОКАЛ — не единственный диалоговый язык, входящий в программное обеспечение микро- и мини-компьютеров. Большое распространение получил также БЕЙСИК, реализованный на большинстве выпускаемых в мире компьютеров. Оба языка обладают всеми свойствами, которые присущи диалоговым языкам, и во многом близки по структуре, но ФОКАЛ, по мнению ряда авторов [12, 13, 19], имеет ряд преимуществ. Так, в ФОКАЛе, в отличие от БЕЙСИКа:

1) идентификаторы переменных и переменных с индексами могут состоять более чем из двух алфавитно- цифровых символов (несмотря на то что интерпретатор различает идентификаторы только по первым двум символам, возможность записи длинных идентификаторов повышает читаемость программы и облегчает ее понимание);

2) имена операторов можно записывать в сокращенной мнемонике (первым символом); это позволяет быстрее вводить программы в компьютер, экономит оперативную память и уменьшает время выполнения программы;

3) номер строки программы является составным номером вида *mm.nn*, где *mm* — номер группы строк, *nn* — номер строки в группе; эта особенность облегчает структурирование программ;

4) числа представляются в форматах с фиксированной

и плавающей точкой с точностью до шести значащих десятичных цифр (имеется возможность управлять форматом чисел при их выводе);

5) предусмотрены развитые средства редактирования и отладки программ, включая трассировку;

6) существенно шире набор встроенных функций, включающий такие дополнительные функции, как ввод— вывод символов, управление системной шиной и т. д.

К особым достоинствам ФОКАЛа следует отнести компактность интерпретатора языка, объем которого не превышает 4 К слов.

Это дает, например, возможность записать интерпретатор в одну микросхему программируемого (К573РФ3) или масочного (К1801РЕ1) ПЗУ.

## 2.2. СЛОВАРЬ И КОМПОНЕНТЫ ЯЗЫКА

**Буквы, цифры и специальные символы.** Каждый язык, как разговорный, так и язык программирования, использует присущий ему словарь. Словарь языка программирования ФОКАЛ состоит из букв, цифр и специальных символов. Программы составляются на основе этого словаря в соответствии с синтаксисом языка.

**Буквы.** Поскольку язык ФОКАЛ построен на основе английского языка, синтаксическая категория *буква* имеет 26 альтернатив:

```
<буква>::= A|B|C|...|X|Y|Z.
```
(2.7)

Таким образом, операторы в языке ФОКАЛ можно записывать только с помощью прописных букв латинского алфавита. В комментариях и текстовых строках можно использовать любые символы, в том числе прописные и строчные буквы русского алфавита.

**Цифры.** В языке ФОКАЛ используются десять арабских цифр:

```
<цифра>::= 0|1|2|3|4|5|6|7|8|9.
```
(2.8)

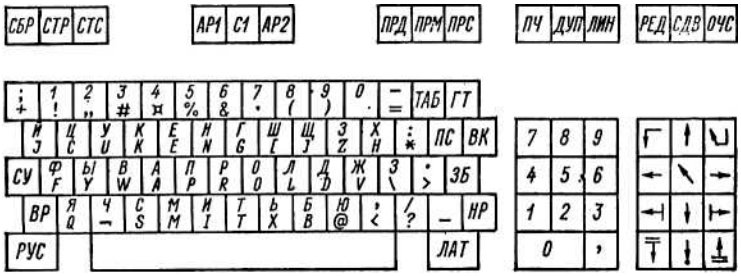



**Рис. 2.1 Клавиатура дисплея 15-ИЭ-00-013**

Отметим, что буква «О» и цифра «0» — два совершенно разных символа.

**Специальные символы.** Количество специальных символов, применяемых в ФОКАЛе для пунктуации и других целей, зависит от реализации, а их начертание связано с типом клавиатуры терминала, используемой для ввода программ и данных в компьютер. Приводимый ниже список специальных символов принадлежит клавиатуре дисплея 15-ИЭ-00-013 (рис. 2.1). В различных реализациях ФОКАЛа используется одно из подмножеств этого списка:

```
<специальный символ>::=+|-|*|/|↑|[|]|(|)|<|>|
    =|.|,|;|<пробел>|!|%|"|?|×|&|:|#|@|ВК|ПС|
    ГТ|ЗБ|_|'|←|→|↦|↤|↖|↖|СТС|СБР|↑|↓.
```

(2.9)

Кроме того, к специальным символам языка ФОКАЛ можно отнести и так называемые *служебные коды* символов, некоторые из которых используются как управляющие. Служебный код некоторого символа вырабатывается при одновременном нажатии двух клавиш — клавиши служебного кода (СУ на рис. 2.1) и клавиши этого символа (таблица кодов и служебных кодов символов клавиатуры микрокомпьютера «Электроника БК-0010» дается в приложении 2). Подчеркнем, что нажатие одной клавиши СУ не приводит к выработке какого-либо кода. В дальнейшем изложении для обозначения, например, служебного кода символа *L* будет использоваться запись СУ/L.

Символы +, —, *, /, представляют собой знаки арифметических операций сложения, вычитания, умножения, деления и возведения в степень соответственно. Кроме того, звездочка (*), которую

выводит интерпретатор ФОКАЛа, указывает на его готовность принять от оператора очередную строку.

Три вида скобок, [ ],( ),< >, эквивалентны и могут использоваться для выделения аргументов функций и индексов, либо для задания последовательности вычисления арифметических выражений. Например, оператор

```
*SET X=<A(K)+2*[(B-3)+D]>/N
```

эквивалентен алгебраическому выражению

$$x = \{a_k + 2\,[(b - 3) + d]\}/n.$$

Символы %, !, #, ¤, ?, ", @ используются для управления ходом выполнения как отдельных операторов, так и всей программы. Например, использование управляющего символа ¤ в операторе печати (TYPE ¤) вызовет распечатку таблицы переменных.

Символы =, ВК, «запятая» (,) и «точка с запятой» (;) используются как разделители. Например, в строке *SET A=3; SET B=1 <ВК> точка с запятой разделяет два оператора присваивания SET, а <ВК> указывает на конец строки.

Отдельно следует сказать о символе, который не имеет начертания, — *пробеле.* Пробел также используется для разделения символов, слов и выражений. В тех случаях, когда возникнет необходимость явно обозначить пробел, будет использоваться знак «␣».

Знак равенства (=) используется в ФОКАЛе в смысле, отличном от принятого в математике. Он не обозначает равенства двух величин, расположенных слева и справа от него, а служит символом *оператора присваивания.* При выполнении этого оператора значение величины или выражения, записанных справа от символа = , присваивается переменной, записанной слева от него. Например, запись $A = B - C$ означает, что переменной $A$ присваивается значение разности величин $B$ и $C$; $A = A + 1$ означает, что переменная $A$ в результате выполнения операции присваивания получит старое значение $A$ (т. е. значение переменной $A$ до начала операции присваивания), увеличенное на единицу. Последнее выражение должно быть прочитано так: *сложить значение переменной. А с единицей и старое значение переменной А заменить новым, только что полученным.*

Точка (.), являясь литеральным символом, несет, кроме того,

различную семантическую нагрузку в зависимости от той синтаксической конструкции, в которой она встречается. Точка используется, во-первых, как разделитель номера группы и номера строки в указателе номера строки программы, во-вторых, как разделитель в формате печати оператора TYPE и, в-третьих, как десятичная точка. Например, в строке

```
*01.15 SET A=1.37; TYPE %6.02,A
```

точка между двумя единицами в начале строки отделяет номер группы (01) от номера строки в группе (15), в операторе присваивания SET точка разделяет целую и дробную части числа, а в конструкции %6.02 она используется для управления форматом вывода.

Символ & используется как специальная переменная (формальный параметр) при организации функции, определяемой пользователем (функция FSBR). Остальные символы используются при подготовке и редактировании программы пользователя.

**Числа.** В ФОКАЛе используются только *вещественные десятичные числа.* Запись этих чисел в языке ФОКАЛ мало чем отличается от традиционной математической записи. Вещественное число может записываться как целое или как вещественное, а перед числом может (но не обязательно) стоять знак.

Способ изображения десятичного числа в ФОКАЛе отличается от традиционного тем, что вместо запятой используется точка. В дальнейшем точку, разделяющую целую и дробную части числа, будем называть *десятичной точкой* в отличие от точки в любом другом ее употреблении (например, для управления форматом печати в операторе TYPE или для отделения номера группы от номера строки).

Вещественное число можно записать в ФОКАЛе двумя различными способами. В первом варианте число записывается с десятичной точкой, за которой должна обязательно стоять хотя бы одна цифра. Такая запись называется представлением числа с *фиксированной точкой.* Ниже приведены примеры правильной записи вещественных чисел в ФОКАЛе и их эквивалентная математическая запись (отметим, что при записи правильной десятичной дроби нуль в целой части числа можно опускать):

| | | | | | | |
|---|---|---|---|---|---|---|
| Запись в ФОКАЛе | 5 | —74.1 | +3.14 | 0.16 | .45 | —.02 |
| Математическая запись | 5 | —74,1 | +3,14 | 0,16 | 0,45 | —0,02 |

Во втором варианте представления десятичное число записывается как целое или вещественное число, умноженное на целую степень 10 (так называемая *показательная форма*). При такой записи непосредственно за целым или вещественным числом идут буква *E* и целое число со знаком или без него. Такая запись называется представлением числа с *плавающей точкой.* Используя показательную форму представления, можно работать с очень большими или очень маленькими числами. Следующая запись вещественных чисел в показательной форме будет верной: 0*E*0 (=0); 8.73*E* + 02 (=873); —741*E* — 14 (=741 · $10^{-14}$); 0.733*E*20 (=0,733 · $10^{20}$). Таким образом, одно и то же вещественное число, например 177, может быть записано в ФОКАЛе самым произвольным образом: 177, 177.0, 1.77*E*2, 17.7*E* + 01, 177*E*0, 1770*E* — 1 и т. д.

В ФОКАЛе существует еще одна возможность задания чисел, так называемые *буквенные константы.* Если перед заглавной латинской буквой или последовательностью букв (кроме буквы *E*) стоит цифра нуль, то расположенная за ним последовательность интерпретируется ФОКАЛом как десятичное число с весовыми коэффициентами для единиц, десятков и т. д., равными порядковым номерам букв в алфавите. Например, число 0*NO* = 10 · 14 + + 15 = 155, поскольку порядковый номер буквы *N* = 14, а буквы *O* = 15.

Диапазон и число значащих цифр вещественных чисел, обеспечиваемых конкретными реализациями ФОКАЛа, ограничены из-за *машинного представления* вещественных чисел. Под мантиссу и порядок вещественного числа в памяти компьютера отводится определенное количество ячеек или разрядов ячейки. При этом разрядность порядка определяет диапазон изменения чисел, а разрядность мантиссы — число значащих цифр в изображении числа. В реализациях ФОКАЛа на 16-разрядных компьютерах типа СМ-4 или «Электроника БК-0010» числа могут лежать в диапазоне от ± $10^{-38}$ до ±$10^{38}$ и иметь не более шести значащих десятичных цифр. Поскольку точность представления вещественных чисел в компьютере всегда ограничена, то выполнение арифметических операций над такими приближенными величинами может привести к накоплению погрешности. Например, для вещественных чисел *x* и *y* тождество

(*x*/*y*) * *y* = *x* может оказаться не всегда верным в машинной арифметике.

**Идентификаторы и переменные.** Чтобы отличать объекты, с которыми должна работать программа, синтаксис языка ФОКАЛ позволяет присваивать им имена. Обычно эти имена называются идентификаторами и создаются программистом. Эквивалентом идентификатора в математической записи является буквенное обозначение различного рода величин. Синтаксическое определение идентификатора имеет вид:

**<идентификатор>::= <буква> {<буква>|<цифра>}.**

(2.10)

Из определения следует, что идентификатор должен начинаться с буквы, за которой может идти произвольная последовательность букв и цифр.

*Замечание.* В языке ФОКАЛ буква *F* используется для выделения идентификаторов стандартных функций, которые будут рассмотрены ниже. Поэтому запрещается использовать букву *F* в качестве *первой* буквы идентификатора.

Из определения (2.10) видно, что в ФОКАЛе можно создавать идентификаторы произвольной длины. Это позволяет использовать в программе идентификаторы, имеющие смысловое значение, что делает программу более наглядной и облегчает ее отладку. Например, вычисляя объем цилиндра как функцию радиуса основания и высоты, величины, описывающие эти параметры, можно обозначить идентификаторами *VOLUME* (объем), *RADIUS* (радиус) и *HIGHNESS* (высота). В этом случае выражение *VOLUME*=3.14159 * (*RADIUS2)* * *HIGHNESS* имеет совершенно очевидный смысл. Отметим, что использовать идентификаторы ОБЪЕМ, РАДИУС и ВЫСОТА нельзя из-за того, что в языке ФОКАЛ запрещается употреблять буквы русского алфавита; программист, не знающий английский язык, может просто обозначить эти величины буквами *V, R* и *H*. Однако конкретные реализации ФОКАЛа накладывают ограничения на то количество символов, по которым они различают идентификаторы. Интерпретатор ФОКАЛа считает значащими только два первых символа идентификатора, поэтому, например, идентификаторы *PARAMETER* (параметр) и *PART* (часть) будут восприниматься как один и тот же

идентификатор *PA,* который и будет помещен интерпретатором языка в свою внутреннюю таблицу переменных. Примерами правильных идентификаторов являются: *J, A*1, *DATA* (последний воспринимается как идентификатор *DA).*

Следующие идентификаторы записаны ошибочно:

| Запись | Ошибка |
|---|---|
| 2*A* | Идентификатор должен начинаться с буквы. |
| *T*- | Идентификатор может содержать только буквы и цифры. |
| *F*3 | Идентификатор не должен начинаться с буквы *F*. |

В каждой реализации языка высокого уровня существуют так называемые *стандартные идентификаторы.* Они имеют строго фиксированное значение и написание и являются элементами словаря этого языка. Программист может использовать стандартные идентификаторы только в том значении, которое им приписано в синтаксисе языка. В языке ФОКАЛ такими идентификаторами являются идентификаторы стандартных функций:

```
<идентификатор стандартной функции>::=
   FABS | FCHR | FCOS | FITR | FEXP | FLOG |
   FRAN | FSBR | FSGN | FSQT | FSIN | FX.
```

(2.11)

Обратите внимание, что все идентификаторы стандартных функций начинаются с буквы *F* и состоят не обязательно из двух символов. В различных реализациях языка ФОКАЛ наборы стандартных функций могут отличаться от приведенного в (2.11) набора не только количественно, но и написанием идентификаторов.

Идентификаторы могут быть использованы для обозначения числовых величин. Числовые величины, используемые в программе, можно разделить на два класса: остающиеся неизменными за время выполнения программы и изменяющими свои значения в процессе выполнения программы. Первые известны как *константы,* вторые — как *переменные.*

*Константы,* т. е. некоторые вещественные числа, можно записывать в тех местах программы, где они используются, и обозначать их идентификаторами не обязательно. Однако, как показывает следующий пример, такой подход не всегда оправдан.

Допустим что в программе часто используется число 3,14159, являющееся приближенным значением математической константы π. Явная запись этой константы всюду, где она требуется, имеет следующие недостатки:

1) последовательность цифр 3,14159 менее выразительна чем символ π;

2) многократное написание этой цифровой последовательности утомительно и может привести к ошибкам (программист может случайно написать 3,15149 или 3,14169 и т. п.), которые очень трудно выявлять;

3) решение изменить точность используемого для π значения потребует внесения исправления во всех местах программы, где оно встречается, что может оказаться еще одним источником ошибок.

В этой ситуации целесообразно использовать идентификатор, значение которого определяется только в одном месте программы (как правило, в ее начале). Учитывая, что в словаре языка ФОКАЛ отсутствуют греческие буквы, в вышеприведенном примере можно было бы использовать идентификатор *PI*.

В ФОКАЛе используются переменные двух типов — *простые переменные* и *переменные с индексом.* Первые обозначаются идентификаторами, а последние — идентификаторами, за которыми в круглых скобках помещается индекс переменной (в некотором смысле ее порядковый номер). Переменные с индексом более удобны, чем простые переменные, в тех случаях, когда необходимо обрабатывать наборы чисел или, иначе говоря, *массивы* чисел.

Рассмотрим массив из двадцати чисел $t_0$, $t_1$, ..., $t_{19}$. Элементы этого массива можно было бы обозначить различными идентификаторами, например, *T*0, *T*1, ..., *T*9, *V*0, *V*1, ..., *V*9 (использовать, скажем, *T*10 уже нельзя, так как интерпретатор ФОКАЛа различает идентификаторы только по первым двум символам и *T*10 воспримется как *T*1), но существует и другой способ, принятый в математике: ввести для всех элементов массива чисел единое обозначение, различая элементы по индексам. Поскольку обычные для математики обозначения, подобные $T_0$, $T_1$, ..., $T_{19}$, не допускаются в ФОКАЛе, для описания таких величин ФОКАЛ позволяет использовать переменные вида *T* (0), *T* (1), ..., *T* (19). Указанные переменные не являются простыми переменными, так как *T* (0), *T* (1),

..., *T* (19) — не идентификаторы (см. определение (2.10)). Это особый вид переменных — *переменные с индексом.* Таким образом, синтаксическое определение переменной имеет следующий вид:

**<переменная>::= <простая переменная> |**

```
    <переменная с индексом>.
<простая переменная>::= <идентификатор>.
<переменная с индексом>::= <идентификатор
    массива>(<индексное выражение>
            [,<индексное выражение>]).
<идентификатор массива>::= <идентификатор>.
    <индексное выражение>::= <целое без знака> |
    <переменная> | <арифметическое выражение>.
```

(2.12)

Итак, ФОКАЛ позволяет обозначать массив чисел единым идентификатором (называемым идентификатором массива), за которым в круглых скобках помещается индексное выражение, значение которого представляет собой индекс данного элемента или, другими словами, его порядковый номер. Значение индексного выражения, т. е. *индекс,* по смыслу всегда число целое. Если индексу соответствует не целое число, то оно округляется до ближайшего целого (например, *T* (2.15) эквивалентно *T* (2)). Индексация может начинаться от любого целого числа (необязательно от нуля). Если двадцать чисел рассматриваются в программе как массив и обозначены *T* (0), *T* (1), ..., *T* (19), а простая переменная *J* принимает целочисленные значения в интервале от 0 до 19, то в программных выражениях можно использовать переменную *T* (*J*). В этом существенное преимущество обозначений *T* (0), *T* (1), ..., *T* (19) перед обозначениями вида *T*0, *T*1, ..., *T*9, *V*0, *V*1, ..., *V*9.

Реализация языка ФОКАЛ накладывает ограничения на пределы изменения индекса. Кроме того, из (2.12) видно, что допускается использование переменных с двумя индексами — аналога двумерных массивов, матриц, а не только одномерных массивов.

Следующая запись представляет переменные с индексом: *X* (1), *A* (*L*), *A*1 (*K* + 5), *ST* (3**C*, *D*), *L* (7, 12).

В индексном выражении могут встречаться и переменные с индексами: *T* (A (*N*) + 1), *ST* (*B* (0), *C* (0)).

*Замечание.* Из предыдущего изложения у читателя могло возникнуть представление о том что переменные с индексом и массивы — это одно и то же. Однако следует четко понимать отличие переменных с индексом, которые используются в ФОКАЛе, от *массивов переменных,* используемых в таких языках программирования как ФОРТРАН, ПАСКАЛЬ и т. д.

Всякий раз, когда в программе впервые встречается новый идентификатор (как простой переменной, так и переменной с индексом), ФОКАЛ помещает ее в таблицу переменных. Переменные с индексом, имеющие одинаковый идентификатор массива и отличающиеся только значением индекса, могут размещаться в таблице переменных в произвольном порядке; более того, можно завести переменные *T* (1), *T* (37) и *T* (99), не заводя переменных со значениями индексам от 2 до 36 и от 38 до 98. Поиск переменной с индексом в таблице переменных осуществляется интерпретатором ФОКАЛа так же, как поиск простой переменной, причем просмотр всегда начинается с самого начала таблицы переменных.

Напротив, массив переменных хранится в «сплошной» области памяти, т. е. ячейки, отводимые под элементы массива, располагаются подряд и не один элемент массива не может быть опущен. Поскольку элементы массива строго упорядочены в оперативной памяти (индекс последующего на единицу превышает предыдущий), то нахождение требуемого элемента массива осуществляется не просмотром, а вычислением адреса элемента относительно начала массива, что существенно ускоряет поиск.

## 2.3. ВЫРАЖЕНИЯ И ПРИСВАИВАНИЯ

**Выражения. Операции и их приоритет.** В программе выражение является правилом для нахождения числового значения или результата. Простейшими выражениями являются числовые константы, простые переменные, переменные с индексом и стандартные функции. Более сложные выражения образуются из чисел, переменных и функций с помощью знаков арифметических операций: сложения, вычитания, умножения, деления и возведения в степень.

Символы, используемые для обозначения арифметических операций, и примеры их записи в ФОКАЛе при водятся в табл. 2.1. Если в выражении встречаются несколько операций, то порядок их выполнения определяется правилами языка ФОКАЛ, либо явно указывается с помощью скобок. В выражениях могут одновременно использоваться круглые, квадратные и угловые скобки, причем открывающая и закрывающая скобки должны быть скобками одного типа.

Понятие выражения в программировании близко понятию алгебраического выражения. Однако в отличие от последних

выражения в ФОКАЛе записываются по особым правилам.

Таблица 2.1

**Символы арифметических операций и их запись в ФОКАЛе**

| Операция | Символ | Алгебраическая запись | Арифметическое выражение в ФОКАЛе |
|---|---|---|---|
| Сложение | + | $a+b$ | A+B |
| Вычитание | — | $a—b$ | $A—B$ |
| Умножение | * | $ab$ | $A * B$ |
| Деление | / | $a : b$ или $a/b$ | $A/B$ |
| Возведение в степень | ^ | $a^B$ | $A^B$ |

Во-первых, в ФОКАЛе (в отличие от алгебры) для записи арифметических выражений можно использовать только прописные буквы латинского алфавита. Например, алгебраическое выражение $x + y$ должно быть записано в ФОКАЛе в виде $X + Y$. Во-вторых, знак операции умножения * в программе нельзя опускать или изображать точкой. Выражение $ax^2 + bx + c$ можно написать в программе только как $A*X\hat{}2 + B*X + C$.

В-третьих, все арифметические выражения должны записываться без каких-либо выходов из строки, в один «этаж». Операции деления (/) и возведения в степень (), а также применение скобок позволяют записать алгебраические выражения $a/b$, $y^3 + 3x^2$, $(ax + b)/(cx + d)$ в соответствии с этим требованием: $A/B$, $Y\hat{}3 + 3*X\hat{}2$, $(A*X + B)/(C*X + D)$. Заметьте, что если в последнем из этих арифметических выражений не поставить скобки, т. е. написать $A*X + B/C*X$ + D, то эта запись будет соответствовать алгебраическому выражению $ax + b/(cx) + d$.

Отметим, что алгебраическую запись $x/(—y)$ нельзя записать в ФОКАЛе как X/—Y, потому что нельзя ставить подряд два знака арифметических операций. Верным арифметическим выражением будет одно из следующих: X/(—Y) или —X/Y. При вычислении арифметических выражений в ФОКАЛе действуют следующие правила приоритета операций: прежде всего выполняется операция возведения в степень, затем операции умножения, деления и в последнюю очередь операции сложения и вычитания.

Кроме того, имеются следующие правила вычисления выражений: 1) если все операции в выражении одинаковы по приоритету, то вычисление выражения происходит строго слева направо; 2) при наличии различных по приоритету операций сначала выполняются операции наибольшего приоритета (тоже слева направо), затем выполняются следующие по приоритету операции; 3) правила 1 и 2 могут быть обойдены за счет использования в выражении скобок — в этом случае первой вычисляется та часть выражения, которая заключена в скобки, причем внутри скобок действуют вышеупомянутые правила приоритета.

*Замечание.* В реализациях ФОКАЛа на микро- и мини-компьютерах операция умножения имеет больший приоритет, т. е. выполняется раньше, чем операция деления. В этих реализациях выражение *A/B * C* соответствует алгебраической записи *a*/(*bc*) в отличие от записи *ac*/*b* в случае одинакового приоритета операций умножения и деления.

Действие этих правил поясняется с помощью нескольких примеров.

1. Так как операция умножения имеет более высокий приоритет, то выражение 6*2 + 4*3 эквивалентно выражению (6*2) + (4*3), значение которого равно 24.
2. Если нужно сложить 4 и 2, а результат умножить на 3, то можно использовать следующее выражение: (4 + 2) * 3. Заметим, что выражение 4 + 2 * 3 не дает правильный результат так как первой будет выполняться операция умножения, как имеющая более высокий приоритет. Как уже отмечалось, в арифметических выражениях допускается использование стандартных функций. При этом вслед за идентификатором стандартной функции в круглых скобках указывается аргумент, который сам может быть арифметическим выражением.
3. В выражении 2 + FSQT (3^2 + 11*5) первым вычисляется содержимое скобки; вычисляется квадрат числа 3 и произведение 11*5, а потом эти два результата складываются, давая значение, равное 64. Затем из него извлекается квадратный корень (FSQT — функция квадратного корня) и суммируется с 2, давая окончательный результат, равный 10.

В табл. 2.2 приводятся некоторые математические выражения и формы их записи на ФОКАЛе.

Отметим еще три существенных обстоятельства, относящихся к записи арифметических выражений на ФОКАЛе.

1. Использование избыточных скобок не приводит к ошибке, т. е. если *A* — арифметическое выражение, то (*A*), [(*A*)] и так далее также являются арифметическими выражениями. Следовательно, при возникновении сомнений в правильности записи выражения на языке ФОКАЛ следует употреблять при записи выражений дополнительные скобки.

Таблица 2.2

**Математические выражения и их запись в ФОКАЛе**

| Математическое выражение | Правильная запись | Неверная запись |
|---|---|---|
| *ab* | *A * B* | *AB* |
| *a*(—*b*) | *A ** (—*B*) или —*A * B* | *A * B* |
| —(*a*+*b*) | —(*A*+*B*) или —*A*—*B* | —A+B или —+A+B |
| $a^{i+2}$ | *A* (*I*+2) | A^ *I*+2 |
| $a^{j-3}$b | *A* (*J*—3) * *B* | A^ *J*—3 * *B* |
| $[(a+b)/c]^3$ | [(*A*+*B*)/*C*] ^ 3 | (*A*+*B*)/*C*^ 3 |
| $(-a)^i$ | (—*A*) ^ *I* | —A^ *I* |

2. Если переменная, используемая в выражении, ранее не встречалась (т. е. ее значение не задавалось в операторе присваивания SET или в операторе ввода ASK), то ей присваивается *нулевое значение.* Например, если значение переменной *A* не было определено где-нибудь ранее, то оператор присваивания SET *B* = *A* присвоит переменной *B* нулевое значение. Наличие такой ситуации может привести к трудно обнаруживаемым ошибкам в программе.

3. Никакие операции, кроме арифметических, использовать при записи выражений на ФОКАЛе нельзя.

Способы построения арифметических выражений регулируются синтаксическими правилами, приведенными ниже:

```
<арифметическое выражение>::= <первичное
     выражение> | [+|-]<первичное выражение>|
     <арифметическое выражение><арифметическая
     операция><первичное выражение>.
<первичное выражение>::= <вещественное без
     знака>|<переменная>|<стандартная функция>|
     <левая скобка><арифметическое выражение>
     <правая скобка>.
<арифметическая операция>::= +|-|*|/|^.
<стандартная функция>::= <идентификатор
     стандартной функции><левая скобка>
```

```
    [<арифметическое выражение>]<правая скобка>.
<левая скобка>::= ( | [ | < .
<правая скобка>::= ) | ] | > .
```

(2.13)

**Оператор присваивания SET.** Этот оператор имеет следующий вид:

```
<оператор присваивания>::= S[ET] <переменная>
    =<арифметическое выражение>.
```

(2.14)

Выполнение оператора присваивания сводится к вычислению арифметического выражения справа от знака равенства и присваиванию полученного значения простой переменной или переменной с индексом, стоящей слева от него. Например, оператор *SET A=3 <ВК> (здесь все символы, кроме * в начале строки, набираются пользователем) вызовет замену текущего значения переменной *A* на значение 3; оператор *SET *I*=*J* <ВК> означает замену текущего значения переменной *I* на текущее значение переменной *J*.

Примеры операторов присваивания:

```
*SET PI=3.14159 <ВК>
*SET C=X+12.7^3-100*FSIN(Y^2) <ВК>
*SET X=X+1 <ВК>
```

Последний оператор увеличивает на единицу значение простой переменной *X*. Если до выполнения этой команды переменная *X* имела значение 3, то после ее выполнения значение *X* станет равным 4.

После выполнения оператора присваивания идентификатор переменной с вычисленным значением помещается в таблицу переменных. Когда идентификатор встречается в выражении, то вместо него подставляется его значение. В каждом операторе SET может быть инициализирована только одна переменная. Таким образом, записи вида SET $A = B = 1$ или SET $A = 3$, $B = 2$ будут ошибочными.

## 2.4. ОПЕРАТОРЫ ВВОДА И ВЫВОДА

Эти операторы обеспечивают ввод исходных данных в

программу и вывод результатов ее работы из ЭВМ.

Следует отметить, что близкой по своему назначению к этим операторам является функция FCHR, которая будет рассмотрена ниже.

**Оператор ввода ASK.** Оператор ASK, так же как и оператор SET, используется для присвоения числовых значений переменным. Различие между ними заключается в том, что при использовании оператора SET значения переменных задаются или вычисляются внутри программы, а при выполнении оператора ASK они вводятся пользователем с клавиатуры терминала.

Оператор ASK имеет следующий синтаксис:

```
<оператор ввода>::= A[SK] <список элементов
      ввода-вывода>.
<список элементов ввода-вывода>::= <элемент
      ввода-вывода>{,<элемент ввода-вывода>}.
<элемент ввода-вывода>::= <переменная>|
      "<текст>" | ! | ¤ | %.
```

(2.15)

Элемент ввода—вывода «текст», помещаемый в кавычках в операторе ASK, может содержать любые символы (кроме кавычек) и при выполнении оператора ASK полностью выводится на печать. Эту особенность оператора ASK удобно использовать для «подсказок» при вводе значений переменных и примечаний.

Управляющий символ ! (восклицательный знак) используется для организации строк вывода, символ денежной единицы а указывает на необходимость распечатки таблицы переменных, а знак процента % управляет форматом вывода переменных.

Ввод числового значения переменной, идентификатор которой указан в операторе ASK, может быть осуществлен после появления выводимого оператором ASK двоеточия (:), а заканчивается вводом одного из допустимых ограничителей:

```
<ограничитель>::= ВК | <пробел> | , | ; | @.
```

(2.16)

Рассмотрим на примере простейшей программы работу оператора ASK. Выполнение программы

```
*1.10 ASK B <ВК>
*1.20 SET A=B+3 <ВК>
*1.30 TYPE B,A <ВК>                                    (2.17)
*
```

начнется, как только пользователь наберет директивную строку *GOTO 1.1 <ВК>

Сначала будет напечатано двоеточие и наступит пауза до тех пор, пока пользователь не введет с клавиатуры терминала числовое значение переменной *B*, а следом за ним ограничитель, например, :7<ВК>

После этого выполнение программы (2.17) продолжится, переменной *A* будет присвоено значение выражения *B* + 3 (=10), а затем оператор вывода TYPE напечатает значения переменных *B* и *A*: 7.0000 10.0000*

С помощью оператора ASK можно ввести сразу несколько значений переменных, например: *ASK K, L(3), S1 <ВК>

Если при вводе чисел в качестве ограничителя используется возврат каретки, то каждое следующее число запрашивается на новой строке:

```
:3<ВК>
:-1<ВК>
:77<ВК>
*
```

Если же в качестве ограничителя используются точка с запятой, запятая или пробел, то все будет печататься в одной строке:

```
*ASK K,L(3),S1 <ВК>
:3,:-1 :77;*
```

В любом из этих вариантов переменная *K* получит значение 3, *L* (3) — 1, а S1 — 77.

Использование ограничителя @ позволяет оставить без изменения значение ранее определенного идентификатора. Если после двоеточия (которое печатается интерпретатором) не задавать числовое значение, а просто нажать клавишу @, то значение соответствующей переменной не изменится. Например, зададим еще раз оператор

```
*ASK K,L(3),S1 <ВК>
```

```
:11<ВК>
:
```
и после второго двоеточия (программа ждет значение переменной *L* (3)) нажмем клавишу :@:—8 <ВК>

Теперь значение переменной *K* станет равным 11, S1 = —8, а значение переменной *L* (3) сохранится без изменения (т. е. *L* (3) = —1).

Элементы ввода (текст) и восклицательный знак удобно использовать для «подсказок» и форматирования вывода — при выполнении оператора ASK (текст) выводится на терминал, а восклицательный знак эквивалентен возврату каретки, например:

```
*ASK "СКОРОСТЬ(КМ/Ч)",V,!,"РАССТОЯНИЕ(км)",S
СКОРОСТЬ(КМ/Ч):60,
РАССТОЯНИЕ(КМ):120<ВК>
*
```

В тексте можно использовать любые символы ФОКАЛа, в том числе буквы русского алфавита, за исключением кавычек. Кавычки на печать не выводятся.

Оператор ASK позволяет пользователю в ответ на двоеточие задавать не только число, но и выражение, значение которого после вычисления присваивается переменной.

В программе

```
*1.10 SET A=1; SET B=2; SET X=3 <ВК>
*1.20 ASK "ЗАДАЙТЕ ЗНАЧЕНИЕ Y",Y <ВК>
*1.30 TYPE Y <ВК>
*
```

после директивы *GOTO 1.1 <ВК> сначала присваивается значение переменным *A, B* и *X*, а потом оператор ASK печатает ЗАДАЙТЕ ЗНАЧЕНИЕ Y: и ждет ввода.

Если пользователь наберет, например, после двоеточия

```
ЗАДАЙТЕ ЗНАЧЕНИЕ Y:A*X^2+B<ВК>
```

то программа вычислит значение этого выражения, используя значения переменных *A, B* и *X*, определенные в строке 1.1, и присвоит это значение переменной *Y*. После чего оператор печати TYPE выведет его на печать:

```
ЗАДАЙТЕ ЗНАЧЕНИЕ Y:A*X^2+B<ВК>
   11.0000*
```

В ответ на запрос оператора ASK можно вводить последовательность заглавных букв латинского алфавита от *A* до *Z* включительно, например:

```
*ASK X,! <ВК>
:ABC,
*
```

Переменной *X* будет присвоено значение 123, которое вычисляется следующим образом: $X = 1 \cdot 10^2 + 2 \cdot 10 + 3 = 123$, так как указанная последовательность букв интерпретируется ФОКАЛом как трехзначное число с весовыми коэффициентами для единиц, десятков и сотен, равными порядковым номерам букв в латинском алфавите (*A* = 1, *B* = 2, ..., *Z* = 26). Из этого правила выпадает буква *E,* которая воспринимается как степень десяти:

```
*ASK L,! <ВК>
:BEC,
*
```

Переменной *L* будет присвоено значение $2 \cdot 10^3$, т. е. 2000.

Рассмотренная особенность оператора ASK крайне удобна, например, при написании обучающих и игровых программ. Фрагмент такой программы мог бы выглядеть так:

```
*10.05 ASK "ПРОДОЛЖАТЬ? (YES ИЛИ NO)",ANS
*10.10 IF (ANS-0NO)10.20,10.15,10.20
*10.15 TYPE !,"КОНЕЦ"; QUIT
*10.20 TYPE !,"ПРОДОЛЖЕНИЕ...",!
```

Если при задании значения переменной вводится слишком много символов, то происходит переполнение буфера ввода и выдается сообщение об ошибке.

В заключение отметим, что численное значение присваивается переменной только после набора ограничителя, поэтому до его задания присваиваемая величина может быть изменена. Для этого надо нажать клавишу ЗБ (забой; отметьте, что на терминале появляется символ подчеркивание), а затем указать желаемое значение переменной, например:

```
*ASK A,D,E <ВК>
:12,:14_17,:8<ВК>
*
```

При выполнении этого оператора пользователь присвоил переменной *A* значение 12, *D* — 17 (вместо ранее набранного, но не введенного числа 14), а *E* — 8.

**Оператор вывода TYPE.** Оператор TYPE применяется для вывода чисел, текущих значений переменных, текстовой информации, таблицы переменных. Кроме того, этот оператор может использоваться для вычисления выражений и вывода полученных результатов. Отметим, что в операторе TYPE допускается любая комбинация чисел, текста, переменных, вычисляемых выражений и управляющих символов:

```
<оператор вывода>::= T[YPE] <список элементов
      ввода-вывода>.
```

(2.18)

Как и в операторе ввода ASK, текст в операторе TYPE должен заключаться в кавычки и может содержать любые символы ФОКАЛа, в том числе буквы русского алфавита, за исключением кавычек. Например, оператор

```
*TYPE "ТАБЛИЦА ЗНАЧЕНИЙ ФУНКЦИИ SIN(X)" <ВК>
```

(2.19)

напечатает: ТАБЛИЦА ЗНАЧЕНИЙ ФУНКЦИИ SIN(X)*

В некоторых реализациях в операторе TYPE символ ВК может выполнять ту же функцию, что и вторая (закрывающая) кавычка. Это означает, что директивная строка (2.19) может быть записана так:

```
*TYPE "ТАБЛИЦА ЗНАЧЕНИЙ ФУНКЦИИ SIN(X) <ВК>
ТАБЛИЦА ЗНАЧЕНИЙ ФУНКЦИИ SIN(X)*
```

(звездочка после текста печатается интерпретатором ФОКАЛа).

Если с помощью одного оператора TYPE выводятся текст, значения переменных, и (или) выражений, то они отделяются в строке программы друг от друга запятыми. Например, программа

```
*1.10 SET A=77.77; SET B=1234.5678; SET C=.5 <ВК>
*1.20 TYPE "A",A,"B",B,"C",C,! <ВК>
*1.30 TYPE "ВЫВОД ЗАКОНЧЕН" <ВК> *
```

после задания директивы *GOTO 1.1 <ВК> присвоит значения переменным *A, B, C* и напечатает следующее:

```
A= 77.7700 B=1234.5600 C= 0.5000
ВЫВОД ЗАКОНЧЕН*
```

Существенно, что значение переменной *B* печатается как 1234.5600. Обратите внимание, что строка 1.2 включает и текст и переменные.

Вывод чисел, а также значений переменных и выражений выполняется оператором TYPE в определенном формате. Первоначально установленный в ФОКАЛе формат обеспечивает вывод чисел с восемью цифрами: четыре слева от десятичной точки (целая часть числа) и четыре справа от десятичной точки (дробная часть числа). При этом нули, предшествующие первой значащей цифре, не печатаются, тогда как в дробной части воспроизводятся все четыре цифры, включая нули. Несмотря на то что в этом формате печатаются восемь цифр, значащими являются только первые шесть цифр, причем пять из них точны, а шестая может быть неточной из-за ошибок округления.

В операторе TYPE для изменения формата вывода чисел и размещения строк текста используются различные управляющие символы.

Формат вывода чисел может быть изменен с помощью символа управления % (процент).

Если используется запись *TYPE *% W.0d, A*, то *W* указывает общее число печатаемых цифр, а *d* — число цифр справа от десятичной точки, с которыми выводится значение переменной *A*. Параметры *W* и *d* должны быть положительными целыми числами. Параметр *d* может опускаться, тогда ни десятичная точка, ни дробная часть числа на печать выводиться не будут. Если параметр *d* указывается, то он должен состоять из двух цифр, т. е. печать двух знаков в дробной части числа должна указываться как % *W*.02, а не как % *W*.2. В качестве примера рассмотрим представление числа 67823,1 при различных форматах вывода:

```
*SET A=67823.1 <ВК>
*TYPE %6.01,A <ВК>
 67823.1*
*TYPE %5,A <ВК>
 67823*
*TYPE %8.03,A <ВК>
 67823.100*
```

Если пользователь хочет выводить результаты в формате с плавающей точкой, он должен указать в операторе TYPE только символ управления форматом: *TYPE %,X где *X* — число, переменная или выражение, значения которых выводятся на печать.

Например:

```
*TYPE %,67.8 <ВК>
 0.678000E+02*
```

где *E* — символ десятичного основания степени, а значение напечатанного выражения определяется как 0,678, умноженное на $10^2$, т. е. 67,8.

После того как пользователь устанавливает определенный формат вывода, все последующие числовые значения будут выводиться в этом формате до тех пор, пока он не изменится. Например, при выполнении программы

```
*1.10 SET A=3.75; SET B=5 <ВК>
*1.20 TYPE %4.02,A,B,! <ВК>
*1.30 TYPE A+B,%1,A,B! <ВК>
*1.40 TYPE B-A,%,A,B,! <ВК>
*1.50 TYPE A*B <ВК>
*GOTO 1.1 <ВК>
```

переменным *A* и *B* будут присвоены сначала значения 3,75 и 5 соответственно. Затем при выполнении строки 1.2 переменные *A* и *B* будут напечатаны в одной строчке в формате % 4.02: 3.75 5.00

При выполнении строки 1.3 значение выражения *A* + *B* будет по-прежнему напечатано в формате %4.02, а потом в той же строке

*A* и *B* будут выведены в формате % 1:8.75 4 5

При выполнении строки 1.4 значение выражения *B—A* будет напечатано в формате %1, а следующие за ним значения *A* и *B* в формате с плавающей точкой, так же как и значение выражения *A* * *B*, которое будет напечатано в следующей строке:

```
1 0.375000E+01 0.500000E+01
0.187500E+02
```

Если пользователь задал некоторый формат вывода, например %3 (т. е. он хочет печатать целые числа, состоящие не более чем из трех цифр), а выводит на печать большее число, то программа игнорирует явное указание формата и выводит это число в формате с плавающей точкой.

Например:

```
*SET X=6785; SET Y=21 <ВК>
*TYPE %3,X,Y <ВК>
0.678500E+04 21*
```

Существенно, что значение переменной *Y* выводится в указанном формате (%3)!

После выполнения оператора TYPE автоматический переход в начало новой строки не выполняется. В ФОКАЛе для организации строк вывода используется управляющий символ ! (восклицательный знак). Указание в операторе TYPE восклицательного знака вызывает печать следующей за ним информации с новой строки до очередного восклицательного знака, приводящего к переходу на новую строку:

```
*TYPE %2.01,5,6,!,7.1,!,"КОНЕЦ",! <ВК>
 5.0 6.0
 7.1
КОНЕЦ
*
```

Этот оператор сначала напечатает числа 5 и 6; потом первый восклицательный знак переведет печать на новую строку, куда будет выведено число 7,1, затем в новой строке напечатается заключенный в кавычки текст. Последний восклицательный знак обеспечивает печать звездочки в начале новой строки. Если бы его не было, то интерпретатор напечатал бы звездочку непосредственно за словом «КОНЕЦ».

Отдельно рассмотрим символ ¤. Использование этого символа в операторе TYPE вызывает распечатку всех идентификаторов и их значений, находящихся в таблице переменных. В эту таблицу помещается каждый идентификатор, впервые использованный в операторах или функциях языка ФОКАЛ.

Следует отметить, что оператор TYPE ¤ выводит информацию, как запись операторов SET в режиме директивных команд. Это сделано с тем, чтобы выведенные на перфоленту или на другой внешний носитель значения переменных можно было ввести с устройства ввода и присвоить эти значения указанным переменным.

Таким образом, если значения переменных *K, L* (1) и RS будут заданы в строке: *SET K=12; SET L(1)=5; SET RS=0.1 <ВК>, то *TYPE ¤ <ВК> выведет таблицу переменных в следующем формате:

```
S K() = 12.0
S L() = 5.0
S RS()= 0.1 *
```

*Замечание.* В некоторых реализациях ФОКАЛа символ ¤ должен быть последним в строке оператора TYPE.

**Оператор распечатки WRITE.** Этот оператор предназначен для вывода строки, группы строк или текста всей программы на устройство вывода. Оператор WRITE имеет следующий синтаксис:

```
<оператор распечатки>::= W[RITE] [ <номер
      строки> | <номер группы> | A[LL] ].
```

(2.20)

Оператор WRITE ALL, так же как и оператор без параметров WRITE, выведет текст всей программы. Если пользователь подготовил программу

```
*1.10 TYPE "A"; SET X=-1; DO 2; TYPE "C",!
*1.20 DO 3; QUIT
*2.10 TYPE "B",!
*2.20 IF (X)2.3,2.4,2.4
*2.30 RETURN
*2.40 TYPE "D",!
```

```
*3.10 TYPE "E",!
```
(2.21)

то оператор WRITE 1 вызовет распечатку всех строк группы 1:

```
*WRITE 1 <BK>

1.10 TYPE "A"; SET X=-1; DO 2; TYPE "C",!
1.20 DO 3; QUIT
*
```

Оператор WRITE 3.1 выведет на устройство печати строку 3.10:

```
*WRITE 3.1 <BK>

3.10 TYPE "E",!
*
```

а оператор WRITE (или WRITE ALL) распечатает всю программу (2.21).

Оператор WRITE задается, как правило, в директивной строке и используется для просмотра программы или ее части (например, при редактировании программы).

Указание в операторе WRITE несуществующего номера строки или группы строк не приводит к возникновению ошибки.

## 2.5. ОПЕРАТОРЫ УПРАВЛЕНИЯ. ПОДПРОГРАММЫ

**Оператор перехода GOTO.** В ранее рассмотренных примерах программ операторы выполнялись в том порядке, в каком они были записаны, т. е. в порядке возрастания номеров строк. В языке ФОКАЛ существует группа операторов, управляющих ходом выполнения программы. К ним относится оператор перехода

<оператор перехода>::= G[OTO] [<номер строки программы>].

(2.22)

Этот оператор изменяет естественную последовательность выполнения операторов. Следом за оператором перехода выполняется та строка программы, номер которой служит аргументом оператора перехода. Таким образом, оператор

перехода передает управление на строку с указанным номером, а саму процедуру называют безусловной передачей управления.

Рассмотрим программу, которая содержит следующую последовательность операторов:

```
*1.10 SET X=2; SET A=5; GOTO 1.4 <ВК>
*1.20 SET A=0; SET B=3 <ВК>
*1.30 GOTO 1.5 <ВК>
*1.40 TYPE X,A,!; GOTO 1.2 <ВК>
*1.50 TYPE A,B <ВК>
*
```
(2.23)

После выполнения операторов присваивания управление передается на строку 1.3, в результате чего будут напечатаны значения переменных *X* и *A* и управление будет передано на строку 1.2. После присвоения значений переменным *A* и *B* управление передается на строку 1.4, выводятся значения *A* и *B* и работа программы на этом завершается. Для запуска программы оператор перехода задают в директивной строке

```
*G[OTO] [mm.nn] <ВК>
```
(2.24)

где *mm.nn* — номер строки, с которой должна начать выполняться программа.

Если в формате оператора (2.24) номер строки опускается, например: *G[OTO], <ВК>, то управление передается строке с наименьшим номером, т. е. первой строке программы.

Таким образом, для запуска рассмотренной нами программы (2.23) можно задать как оператор *GOTO <ВК>, так и оператор *GOTO 1.1 <ВК>

В обоих случаях программа начнет выполняться с первой строки и в результате ее выполнения будет напечатано

```
2.0000     5.0000
0.0000     3.0000*
```

Оператор *GOTO 1.2 <ВК> запустит программу (2.23) со второй строки, а результатом ее выполнения будет:

```
0.0000        3.0000
*
```

*Замечания.* 1. Если в операторе GOTO указан отсутствующий в программе номер строки, то будет выведено сообщение об ошибке, а выполнение программы прекратится. 2. В большинстве реализаций ФОКАЛа в качестве аргумента оператора GOTO может использоваться переменная. 3. В больших программах следует использовать операторы GOTO как можно реже, чтобы не запутывать логику программы и не создавать дополнительные трудности при ее отладке и сопровождении.

**Условный оператор IF.** Оператор IF служит для условной передачи управления, т. е. передает управление строке программы в зависимости от значения некоторого арифметического выражения.

В общем виде оператор IF можно определить следующим образом:

**<условный оператор>::= I[F] (<арифметическое выражение>)<ii.jj>[,<kk.ll>[,<mm.nn>]].**

(2.25)

Здесь *ii.jj*, *kk.ll*, *mm.nn* — номера строк программы.

Когда программа доходит до оператора IF, то она вычисляет значение арифметического выражения, заключенного в круглые скобки и осуществляет переход на строку программы в соответствии со следующими правилами:

1) если значение арифметического выражения *меньше нуля,* то управление передается строке с номером *ii.jj;*

2) если значение арифметического выражения *равно нулю,* то управление передается строке с номером *kk.ll;* 3) если значение арифметического выражения *больше нуля,* то управление передается строке с номером *mm.nn.*

Рассмотрим программу, после выполнения которой меньшее из двух заданных неравных чисел будет увеличено вдвое, а большее оставлено без изменений; если числа равны, то печатается информационное сообщение:

```
*1.10 ASK "ВВЕДИТЕ ДВА ЧИСЛА",X,Y <ВК>
*1.20 IF (X-Y)1.3,1.4,1.5 <ВК>
*1.30 SET x=2*x; GOTO 1.6 <ВК>
*1.40 TYPE "ЧИСЛА РАВНЫ",!; GOTO 1.6 <ВК>
*1.50 SET Y=2*Y; <ВК>
```

```
*1.60 TYPE X,Y,! <ВК>
*
```

Если из двух введенных оператором ASK чисел одно, например $X$, будет меньшим, т. е. $X < Y$, то вычисляемое в строке 1.2 выражение $X — Y < 0$ и управление будет передано строке 1.3; если $X$ и $Y$ таковы, что $X > Y$, то $X — Y > 0$ и управление передается строке 1.5 и, наконец, если $X = Y$, то $X — Y = 0$ и управление передается строке 1.4:

```
*GOTO <ВК>
ВВЕДИТЕ ДВА ЧИСЛА: 3, : 5<ВК>
   6.0000   5.0000

*GO <ВК>
ВВЕДИТЕ ДВА ЧИСЛА: 3, : 1<ВК>
   3.0000   1.0000

*GO <ВК>
ВВЕДИТЕ ДВА ЧИСЛА: 2, : 2<ВК>
ЧИСЛА РАВНЫ
2.0000        2.0000
```

В ФОКАЛе допускается сокращенная форма условного оператора

```
IF (<арифметическое выражение>)<ii.jj>,<kk.ll>
```

(2.26)

или

```
IF (<арифметическое выражение>)<ii.jj>
```

(2.27)

Если выражение в операторе IF, используемое в виде (2.26), меньше нуля, то управление передается строке с номером *ii.jj*; строке с номером *kk.ll*, если выражение равно нулю; и следующему за IF оператору, если выражение больше нуля:

```
*3.19 IF (B)1.8,1.9 <ВК>
```

```
*3.20 TYPE "ВЫРАЖЕНИЕ БОЛЬШЕ НУЛЯ" <ВК>
```

Если $B < 0$, то управление передается строке 1.8; если $B = 0$, то управление передается строке 1.9; и если $B > 0$, то выполняется следующий за IF оператор, т. е. строка 3.20.

Если в операторе IF точка с запятой или ВК следуют за единственным номером строки *ii.jj* (формат записи (2.27)), то управление передается на строку *ii.jj*, в том случае, когда выражение меньше нуля, во всех остальных случаях выполняется следующий за IF оператор. Например, *2.20 IF (X)1.8; TYPE «Q» <ВК>

Если $X < 0$, то управление передается строке 1.8, в противном случае будет напечатана буква *Q*.

Если в операторе IF в качестве арифметического выражения используются вещественные (нецелые) числа, то проверка выражения на равенство нулю не всегда осуществима из-за особенностей реализации арифметики с плавающими числами в ЭВМ. Во избежание ошибок такого рода пользователь либо не должен использовать в операторе IF нецелочисленную арифметику, либо сравнивать (возможно, по модулю) значение выражения с некоторой малой величиной ε.

*Замечание.* 1. Задание в операторе IF отсутствующего в программе номера строки приведет к прекращению выполнения программы и печати сообщения об ошибке. 2. В большинстве реализаций ФОКАЛа вместо номеров строк могут использоваться переменные.

**Организация циклов.** Одним из основных достоинств вычислительной машины является ее способность многократно повторять некоторую последовательность действий, т. е. работать в цикле. Повторение циклической части программы заканчивается при выполнении условия выхода из цикла, которое определяется решаемой задачей.

Предположим, надо напечатать квадраты всех чисел от 1 до 20. Программа может быть написана следующим образом:

```
*1.10 SET N=1 <ВК>
*1.20 SET S=N^2 <ВК>
*1.30 TYPE N,S,! <ВК>
*1.40 SET N=N+1 <ВК>
*1.50 IF (N-20)1.2,1.2,1.6 <ВК>
*1.60 TYPE "КОНЕЦ ЦИКЛА",! <ВК>
```

```
*
```
(2.28)

Сначала программа присваивает переменной *N* значение 1. Переменной S присваивается значение $N^2$, и программа выводит значения N и S, т. е. 1 и 1. Затем N увеличивается на 1, и в строке 1.5 значение переменной N сравнивается с числом 20. Если N меньше или равно 20, то происходит переход на строку 1.2, где N возводится в квадрат, присваивается S и так далее. Программа находится в цикле до тех пор, пока N не станет больше 20, т. е. N будет равно 21. Тогда управление передается на строку 1.6 и цикл (а вместе с ним и программа) завершится. Таким образом, для программы (2.28) условием выхода из цикла является выполнение неравенства N > 20.

Строки 1.2, 1.3, 1.4 и 1.5 в программе (2.28) выполняются более одного раза (точнее, двадцать раз). Эти четыре строки образуют *тело цикла.* Отметим, что условие выхода из цикла можно проверить в любой удобной точке программы. Например, программу (2.28) можно записать и так:

```
*1.10 SET N=1 <ВК>
*1.20 IF (N-20)1.3,1.3,1.6 <ВК>
*1.30 SET S=N^2 <ВК>
*1.40 TYPE N,S,! <ВК>
*1.50 SET N=N+1; GOTO 1.2 <ВК>
*1.60 TYPE "КОНЕЦ ЦИКЛА",! <ВК)
*
```

Во многих случаях пользователь заранее знает, сколько раз должен быть выполнен цикл. При этом можно использовать *переменную цикла* и увеличивать ее значение на единицу при каждом выполнении цикла. Часто при подготовке цикла пользователь может проверять либо условие окончания цикла, либо переменную цикла. Пусть требуется построить таблицу синусов при изменении аргументов от 0 до 0,999 с шагом 0,001. Эту программу можно записать так (проверяется условие окончания цикла)

```
*1.10 SET X=0 <ВК>
*1.20 TYPE X,FSIN(X),! <ВК>
*1.30 SET X=X+0.001 <ВК>
*1.40 IF (X-0.999)1.2,1.2 <ВК>
*
```
(2.29)

или так (проверяется переменная цикла)

```
*1.10 SET X=0; SET C=X <ВК>
*1.20 TYPE X,FSIN(X),! <ВК>
*1.30 SET X=X+0.001; SET C=C+1 <ВК>
*1.40 IF (0-1000)1.2,1.2 <ВК>
*
```

**Оператор повторения FOR.** Оператор FOR используется для автоматизации построения циклов, содержащих переменную цикла.

Синтаксис оператора FOR определяется следующим образом:

```
<оператор повторения>::= F[OR] <переменная
      цикла>=<начальное значение>[[,<шаг>],
      <предельное значение>]; <тело цикла>.
<переменная цикла>::= <переменная>.
<начальное значение>::= <арифметическое
      выражение>.
<шаг>::= <арифметическое выражение>.
<предельное значение>::= <арифметическое
      выражение>.
<тело цикла>::= {<оператор>}.                (2.30)
```

Оператор FOR определяет выполнение операторов из тела цикла некоторое число раз. Это число итераций определяется значениями управляющих параметров, заданных в операторе FOR (начальным значением, шагом, предельным значением). Если через $n_1$, $n_2$, $n_3$ обозначить перечисленные управляющие параметры, то число итераций равно значению следующего выражения:

$$\max \{E [(n_3 — n_1 + n_2)/n_2], 1\}, \qquad (2.31)$$

где E [ ] — целая часть числа.

Управляющие параметры могут быть положительными или отрицательными; когда шаг равен единице, то его можно опускать в записи оператора FOR.

Когда в программе пользователя встречается оператор FOR, например, *FOR K=N1,N2,N3; <ОПЕРАТОРЫ> <ВК>, то обработка цикла начинается с того, что переменной цикла К присваивается начальное значение N1, а затем последовательно

выполняются все операторы, входящие в *область действия цикла*. В эту область входят не только те операторы, которые образуют *тело цикла,* но и те строки и (или) группы строк, на которые (с помощью операторов GOTO, IF, DO) передается управление из тела цикла. После этого производится приращение переменной цикла, включающее в себя следующие шаги:

1) к значению переменной цикла прибавляется шаг N2;
2) число итераций уменьшается на 1; 3) если число итераций положительно, то выполнение операторов, входящих в область действия цикла, будет продолжено с новым значением переменной цикла (когда число итераций уменьшится до нуля, произойдет выход из цикла).

Значение переменной К при выходе из цикла сохраняется и равно последнему значению управляющей переменной К (при котором выполнялось тело цикла) увеличенному на приращение N2, которое равно 2, в приведенном примере:

```
*1.10 SET S=0 <ВК>
*1.20 FOR K=1,2,10; SET S=S+K <ВК>
*1.30 TYPE %2,K,! <ВК>
*GOTO <ВК>
11
*
```

Из этого примера видно, что после выхода из цикла значение К равно 11.

Таким образом, операторы, находящиеся в одной строке с оператором FOR (и, конечно, операторы из области действия цикла), выполняются, по крайней мере, один раз (при К = N1).

Вернемся к программе (2.29), вычислявшей значения функции sin (*x*) при изменении аргумента от 0 до 0,999 с шагом 0,001. Цикл в этой программе был организован с помощью оператора IF. Применяя оператор FOR, мы получим программу, отличающуюся большей компактностью и наглядностью:

```
*1.10 SET X=0 <BK>
*1.20 FOR C=1,1,1000; TYPE X,FSIN(X),!;


SET X=X+0.001 <BK>                                    (2.32)
*
```

Более того, программу (2.32) вообще можно записать одной

строкой, используя аргумент X как переменную цикла:

```
   *1.10 FOR x=0,0.001,0.999; TYPE X,FSIN(X),!
<ВК> *
```

Если *шаг цикла* равен 1, то соответствующая переменная или число могут опускаться:

```
*FOR K=1,3; TYPE %1,K,! <ВК>
1
2
3
*
```

Здесь K изменяется от 1 до 3 с шагом 1 и при каждом значении K выполняется оператор TYPE.

Шаг может быть как положительным, так и отрицательным. При этом необходимо быть внимательным, чтобы не построить неправильный цикл. Например, оператор

```
*FOR K=10,-1,15; TYPE K,! <ВК>
```

не имеет смысла, так как в соответствии с (2.31) должен выполниться отрицательное число раз.

Рассмотрим в качестве примера программу вычисления $n!$ ($n! = n\,(n-1)\,(n-2)\,\ldots\,2\cdot1$):

```
*1.10 ASK "N",N; SET NF=1 <ВК>
*1.20 FOR J=2,N; SET NF=NF*(N-J+2) <ВК>
*1.30 TYPE %,"N-ФАКТОРИАЛ=",NF,! <ВК>
*
```

При запуске программы она запрашивает значение *N:*

```
*G <ВК>
N:
```

Если в ответ на это пользователь задает число 5, то программа напечатает:

```
N:5 <ВК>
N-ФАКТОРИАЛ= 0.120000E+03
*
```

Циклы, образованные с помощью оператора FOR, могут быть вложены в другие циклы, т. е. один цикл может быть записан в теле или области действия другого цикла. Например, следующая директивная строка написана правильно:

```
*FOR N=1,2; FOR M=1,3; TYPE %1,N,M,N*M,! <ВК>
```

Так как она содержит как внешний цикл (FOR N = = 1, 2), так и внутренний цикл (FOR M = 1, 3). За одно выполнение внешнего цикла внутренний цикл выполнится полностью.

В результате выполнения этой директивной строки будет напечатано 6 строк:

```
1 1 1
1 2 2
1 3 3
2 1 2
2 2 4
2 3 6
*
```

*Замечания*. 1. Значение переменной цикла может изменяться в теле цикла. Поскольку дальнейшая обработка приращения будет выполняться при измененном значении переменной цикла, то операторы, входящие в область действия цикла, могут управлять числом повторений цикла. 2. Изменение значений управляющих параметров в теле цикла не влияет на число итераций. Это объясняется тем, что при первоначальной обработке оператора FOR интерпретатор пересылает исходные значения управляющих параметров в специальную область памяти, в которую пользователь из своей программы доступа не имеет.

**Оператор завершения QUIT.** Оператор QUIT не имеет аргументов и предназначен для завершения работы программы пользователя и перевода интерпретатора ФОКАЛа в режим задания директивных строк. Синтаксис оператора QUIT следующий:

```
<оператор завершения>::= Q[UIT]
```
(2.33)

Оператор QUIT — всегда последний выполняемый оператор в данной строке программы.

Например,

```
*1.10 FOR I=1,3; SET B(I)=I; TYPE B(I),! <BK>
*1.20 QUIT; TYPE "1.2" <BK>
*1.30 SET C=3; QUIT <BK>
```

```
*G <ВК>
   1.0000
   2.0000
   3.0000
*
```

Оператор TYPE "1.2" в строке 1.2 и строка 1.3 никогда не выполнятся, т. к. расположены за оператором завершения программы QUIT.

Когда в строке встретится оператор QUIT, выполнение программы заканчивается и на терминал выводится звездочка, означающая, что пользователь может вводить новые директивные строки или строки программы. Завершение программы оператором QUIT является «хорошим тоном» с точки зрения современной методики программирования.

**Организация подпрограмм. Операторы DO и RETURN.** Довольно часто возникает необходимость выполнить определенную последовательность операторов в разных местах некоторой программы. Оформление часто повторяющейся части программы в виде подпрограммы (т. е. такой последовательности операторов, на выполнение которой можно перейти из различных мест программы) обеспечивает как наглядность и простоту организации программы, так и более эффективное использование памяти. Естественно, что после завершения выполнения подпрограммы управление должно быть возвращено в ту точку программы, из которой был сделан переход в подпрограмму.

Предположим, что в какой-то программе нужно многократно вычислять площади различных треугольников по трем сторонам. Вычисления можно вести по формуле Герона:

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

где $p = (a + b + c)/2$; $a$, $b$, $c$ — длины сторон треугольника.

Если же подкоренное выражение в формуле, приведенной выше, отрицательно, то в программе печатается сообщение об ошибке.

Программу вычисления площади треугольника можно записать так (в предположении, что A, B и C заданы);

```
*30.10 SET P=(A+B+C)/2; SET P=P*(P-A)*(P-B)*(P-C)
*30.20 IF (P)3.30; SET S=FSQT(P)
```

```
*30.25 QUIT
*30.30 TYPE "ЭТО НЕ ТРЕУГОЛЬНИК!",!
*
```

(2.34)

Если подкоренное значение P, вычисленное в строке 30.1, меньше нуля, то оператор IF передает управление на строку 30.3; если P > 0, то из него извлекается квадратный корень (FSQT — функция квадратного корня) и в строке 30.25 по оператору QUIT программа завершит свое выполнение.

Язык ФОКАЛ позволяет не записывать последовательность операторов (2.34) каждый раз в тех местах программы где требуется вычисление площади треугольника, а просто обращаться к (2.34) из различных точек программы как к подпрограмме. Для использования некоторой части программы как подпрограммы в языке ФОКАЛ применяются операторы DO, RETURN и функция FSBR.

**Оператор вызова подпрограммы DO.** Оператор DO используется для обращения к строке или группе строк как к подпрограмме и имеет следующий синтаксис:

```
<оператор вызова подпрограммы>::= D[O] [<номер
      строки программы> | <номер группы> | A[LL]].
```

(2.35)

Если по оператору DO происходит обращение к *единственной строке*, то выполняются все операторы в этой строке, после чего управление передается оператору, непосредственно следующему за оператором DO. Например,

```
*1.10 TYPE "X"
*1.20 DO 4.5; TYPE "Y"
*1.30 DO 4.5; TYPE "Z"
*4.50 TYPE "A"
*G
XAYAZA*
```

(2.36)

При выполнении этой программы сначала напечатается буква X, потом по первому оператору DO выполнится как подпрограмма строка 4.5 и напечатается буква A, после чего выполнится

непосредственно идущий за оператором DO оператор TYPE "Y" и выведется буква Y. Аналогично будет выполнена и строка 1.3. Обратите внимание, что потом выполнится строка 4.5, но уже не как подпрограмма, а как последняя строка программы. Пользователь должен быть внимателен, чтобы избегать подобных ситуаций, если он хочет, чтобы строка или группа строк выполнялась только как подпрограмма. Для этого необходимо завершать программу оператором QUIT. Например, в программе (2.36) надо добавить для этого строку 1.4: *1.40 QUIT тогда программа (2.36) выполнится так:

```
*GOTO
XAYAZ*
```

Если по оператору DO происходит обращение к группе строк, то выполнение этой подпрограммы начинается со строки, имеющей в группе наименьший номер. Строки указанной группы выполняются последовательно в порядке возрастания номеров строк (если не встречаются операторы IF или GOTO) либо до исчерпания строк в группе, либо до встречи с оператором RETURN; после этого управление передается оператору, следующему непосредственно за оператором DO. Например,

```
*1.05 SET A=5
*1.10 DO 2; DO 2
*1.15 TYPE A,!; QUIT
*2.10 TYPE A,!
*2.20 SET A=A-1
*G
   5.0000
   4.0000
   3.0000
*
```
(2.37)

Оператор, непосредственно следующий за оператором DO, будем называть *оператором выхода*. Таким оператором может быть любой оператор; например, в программе (2.37) для первого оператора DO в строке 1.1. оператором выхода является второй оператор DO.

Оператор DO ALL (или просто DO) вызывает выполнение всей

программы, начиная со строки с наименьшим номером (как GOTO).

В программе может быть создано любое число подпрограмм. Из одной подпрограммы можно вызвать другие подпрограммы. Они, в свою очередь, могут вызывать следующие подпрограммы и т. д. Это означает, что подпрограммы могут быть *вложенными*. Глубина вложенности подпрограмм ограничивается только размером свободной динамической памяти, оставшейся после распределения памяти между программой пользователя и таблицей переменных. Например,

```
*1.05 ASK "X",X,"Y",Y,"Z",Z
*1.10 TYPE !,"ТАБЛИЦА ПАРАМЕТРОВ",%2.01,!; DO 2.4
*1.20 DO 2; QUIT
*2.10 DO 3; TYPE "X=",X,!
*2.20 DO 3; TYPE "Y=",Y,!
*2.30 DO 3; TYPE "Z=",Z,!
*2.40 TYPE " -------------- ",!
*3.01 TYPE "ПАРАМЕТР  "
*G
X: 1.2,Y: 2.3,Z: 4

ТАБЛИЦА ПАРАМЕТРОВ

-----------------

ПАРАМЕТР X= 1.2
ПАРАМЕТР Y= 2.3
ПАРАМЕТР Z= 4.0
-----------------
*
```

При большой глубине вложенности операторов DO возможно переполнение памяти, отведенной для программы пользователя, что вызовет сообщение об ошибке. Если при выполнении подпрограммы встречаются операторы GOTO или IF, то возможны следующие варианты.

1. Оператор GOTO или IF передает управление на строку внутри подпрограммы. Выполнение подпрограммы продолжится с

этой строки до нормального завершения, после чего управление передается на оператор выхода.

2. Оператор GOTO или IF передает управление на строку, находящуюся вне подпрограммы. В этом случае выполнятся все операторы этой строки (если только она сама не содержит операторы GOTO или IF), а затем, не возобновляя выполнения подпрограммы, управление передается на оператор выхода. Таким образом, та часть подпрограммы, которая расположена за операторами GOTO и IF, выполнена не будет.

Следующая программа хорошо иллюстрирует эти положения:

```
*1.10 TYPE "A"; SET x=-1; DO 3.1; TYPE "D"
*1.20 DO 2; DO 2
*2.10 TYPE "G"
*2.20 IF (x)2.5,2.6,2.7
*2.50 TYPE "H"
*2.60 TYPE "I"
*2.70 TYPE "J"
*2.90 TYPE %2,X," "; SET   X=X+1
*3.10 TYPE "B"; GOTO 5.1;  TYPE "F"
*5.10 TYPE "C"*5.20 TYPE   "EL",!
*G
ABCDGHIJ - 1 GIF 0 GJ 1 BCEL
*
```

Заметим, что, во-первых, группа строк 2 и строка 3.1 выполняются и как подпрограммы, и как часть программы; во-вторых, оператор TYPE "F" в строке 3.1 так ни разу и не выполнился.

*Замечания*. 1. В качестве указателя номера группы или строки в операторе DO может использоваться переменная.

2. Если в операторе DO указывается несуществующий номер строки или группы, то выдается сообщение об ошибке.

**Оператор завершения подпрограммы RETURN.** Оператор RETURN используется для выхода из подпрограммы, т. е. для возврата к оператору, следующему за оператором DO или за оператором, содержащим обращение к функции FSBR, и имеет следующий синтаксис:

```
<оператор завершения подпрограммы>::= R[ETURN].
```

(2.38)

Когда в подпрограмме встречается оператор RETURN, то выполнение подпрограммы прекращается, а управление передается оператору выхода. Операторы, стоящие за оператором RETURN в той же строке, никогда не выполняются, поэтому за оператором RETURN целесообразно помещать только комментарии.

Рассмотрим пример, показывающий организацию программы с двумя выходами из нее.

```
*1.10 SET X=-1; DO 2; DO 2; DO 2
*1.20 QUIT
*2.10 IF (X)2.2,2.3,2.4
*2.20 TYPE %2,"ВЫХОД1 X=",X,!; SET X=X+1; RETURN
*2.30 TYPE "ВЫХ0Д2 X=",X,!; SET X=X+1; RETURN
*2.40 TYPE "КОНЕЦ",!
*G
ВЫХОД1 X= - 1
ВЫХОД2 X= 0
КОНЕЦ
*
```

При первом входе в подпрограмму X = —1, и выход из нее происходит по оператору RETURN в строке 2.2. При втором входе X = 0, и выход происходит по оператору RETURN в строке 2.3. При последнем входе X = 1, и выход из подпрограммы происходит после выполнения ее последней строки.

**Организация больших циклов.** Рассматривая организацию циклов с помощью оператора FOR, мы говорили, что в цикле могут быть выполнены только операторы, находящиеся в одной строке с оператором FOR. Если решаемая задача требует циклического выполнения большого количества операторов, то указанное ограничение легко преодолевается совместным использованием операторов FOR и DO. В этом случае за оператором цикла записывается оператор DO. Телом цикла является тогда оператор обращения к подпрограмме, а следовательно, та группа строк, на которую он указывает. Такой подход позволяет организовать циклы произвольной вложенности и сложности.

*Пример.* Предположим, что требуется вычислить и вывести на печать значения переменной X в диапазоне от 0 до 1 с шагом 0,25 и значения

синуса от X, причем переменная X должна выводиться в форме с фиксированной точкой, а синус — в форме с плавающей точкой:

```
*1.10 FOR X=0,0.25,1; DO 2; QUIT
*2.10 TYPE "  ",%3.02,"X=",X
*2.20 TYPE "  ",%,"SIN(X)=",FSIN(X),!
*G
  X= 0.00  SIN(X)= -0.953674E-06
  X= 0.25  SIN(X)=  0.247404E+00
  X= 0.50  SIN(X)=  0.479425E+00
  X= 0.75  SIN(X)=  0.681638E+00
  X= 1.00  SIN(X)=  0.841471E+00
*
```

## 2.6. СРЕДСТВА ОТЛАДКИ ПРОГРАММ

**Комментарии в программе.** При написании программы рекомендуется включать в программу поясняющий текст — комментарии, которые могут объяснить назначение программы или любой ее части, детально описать реализованный в программе алгоритм и используемые переменные, сообщить о формате вывода предполагаемых результатов. Для включения комментариев в программе в языке ФОКАЛ используется оператор COMMENT:

```
<оператор комментария>::= C[OMMENT] <любая
      последовательность символов>.          (2.39)
```

Комментарии никак не влияют на выполнение программы и при счете на печать не выводятся. Поэтому они могут располагаться в любом месте программы и содержать любые символы, в том числе буквы русского алфавита.

Комментарии распечатываются вместе с текстом программы оператором WRITE. Например,

```
*1.10 COMMENT ПРОГРАММА ВЫЧИСЛЯЕТ СУММУ N ЧЛЕНОВ
*1.12 COMMENT РЯДА 1 + 1/2 + ... + 1/N
*1.20 ASK "N",N; COMMENT ВВОД ЧИСЛА N
*1.30 SET S=0; FOR J=1,N; SET S=S+1/J
*1.40 TYPE %,"S=",S,!; COMMENT S ВЫВОДИТСЯ В ФОРМЕ
*1.50 COMMENT С ПЛАВАЮЩЕЙ ЗАПЯТОЙ
```

При трассировке программы на печать выводится сам оператор COMMENT и текст комментария до первого ограничителя. Если при трассировке нужна распечатка всего текста комментария, то вместо пробела следует печатать другой символ, например, двоеточие.

Комментарии занимают место в оперативной памяти и могут создать определенные трудности в разработке и выполнении очень больших программ. Для разрешения этого противоречия полезно иметь две копии одной программы: одну с комментариями (для удобства отладки и сопровождения), а другую без них (для выполнения).

**Оператор стирания ERASE.** С помощью оператора ERASE пользователь может стереть находящиеся в оперативной памяти строку, группу строк, всю программу и (или) таблицу переменных.

Оператор ERASE может записываться в одном из следующих форматов:

```
<оператор стирания>::= E[RASE] [<номер строки
      программы> | <номер группы> | A[LL] |
      T[EXT]].
```

(2.40)

При этом оператор ERASE *mm.nn* стирает строку программы с номером *mm.nn*; ERASE *mm* — все строки в группе с номером *mm*; ERASE — все идентификаторы и их значения в таблице переменных; ERASE ALL — всю программу пользователя и таблицу переменных, а ERASE TEXT — только всю программу пользователя.

Перед началом работы пользователю рекомендуется всегда использовать оператор ERASE ALL, а перед запуском отлаженной программы — оператор ERASE (он может быть первым оператором программы).

*Пример.* Оператор ERASE используется для стирания строки 1.5:

```
*ERASE ALL
*1.20 SET A=2
*1.30 SET C=3
*1.50 TYPE A-B
*1.60 TYPE A-C
*ERASE 1.5
*WRITE
 1.20 SET A=2
```

```
 1.30 SET C=3
 1.60 TYPE A-C
*
```

**Оператор редактирования MODIFY.** Для устранения ошибок, выявленных при отладке программы, можно либо набрать заново всю строку, в которой обнаружены ошибки, либо исправить ошибки в этой строке, используя оператор MODIFY. Оператор MODIFY может указываться *только* в директивной строке и имеет следующий синтаксис:

**<оператор редактирования>::= M[ODIFY] <номер строки программы>.**

(2.41)

В отличие от других директивных операторов языка, повлиять на выполнение которых после нажатия символа ВК пользователь не может, редактирование указанной в операторе MODIFY строки начинается после того, как оператор распечатает эту строку, например,

```
*1.30 T "ПРИМЕР N1",!
.
.
.
.
*MODIFY 1.3
T "ПРИМЕР N1",!
```

Поскольку дальнейшая работа с оператором MODIFY существенно зависит от реализации ФОКАЛа и, в частности, от используемого терминала, то конкретные особенности его работы будут рассмотрены позднее, в разделах, описывающих конкретные реализации.

Подчеркнем, что оператор MODIFY применяется для редактирования текста строки программы и *не может* использоваться для редактирования номера строки.

**Исправление ошибок при подготовке программы.** Кроме оператора MODIFY ФОКАЛ имеет средства, позволяющие немедленно исправлять ошибки, допущенные пользователем в

процессе ввода программы в компьютер, и (в некоторых реализациях) редактировать строку до ее окончательного ввода (т. е. до нажатия клавиши ВК). Поскольку эти средства также зависят от реализации ФОКАЛа, они будут рассмотрены позднее, применительно к конкретной реализации.

**Трассировка программы.** При отладке программы пользователь может использовать знак вопроса (?) для управления трассировкой программы, т. е. для вывода на печать содержимого строк программы по мере их выполнения. Средства трассировки позволяют пользователю получать диагностическую информацию о ходе выполнения его программы без изменения ее логики или структуры.

Знаки вопроса можно помещать в любых местах строки. При выполнении программы текст строки (или строк), заключенный между двумя знаками вопроса или знаком вопроса и ВК в последней строке программы, выводится на печать. Оператор печати TYPE выполняется при этом обычным образом. Знак вопроса, используемый в текстовых элементах операторов ASK и TYPE, не воспринимается как символ трассировки.

Например, при следующей организации программы

```
*1.10 SET A=1; SET B=5; SET C=3
*1.20 TYPE %1,?A+B-C?,!
*1.30 TYPE ?B+C/A?,!
*1.40 TYPE ?B-C/A,!
*1.50 FOR D=1,3; TYPE D+A,!
*
```

на печать будет выводиться такая информация

```
*G
A+B-C  3
B+C/A  8
B-C/A,  2!
FOR D=1,3; TYPE D+A, 2!
TYPE D+A, 3!
TYPE D+A, 4!
*
```

Вывод символов A + B — C обеспечивает первая пара знаков «?» в строке 1.2; вывод символов B + C/A обеспечивает вторая пара знаков «?» в строке 1.3; после этого выводится информация между

знаком «?» в строке 1.4 и концом последней строки программы (та часть строки 1.5, которая выполняется в цикле, распечатывается столько раз, сколько выполняется цикл). При задании оператора *GOTO? выполняется трассировка всей программы, т. е. эта команда эквивалентна заданию знака «?» в качестве первого символа первой строки программы. Например,

```
*1.01 SET X=1
*1.03 TYPE %1,X,!
*G?
SET X=1
TYPE %1,X,  1!
*
```

**Диагностика ошибок в языке ФОКАЛ.** Если при выполнении программы в ней встречается ошибка (неверно записанное арифметическое выражение, неправильное использование функции, недопустимый идентификатор и т. п.), то интерпретатор ФОКАЛа прерывает выполнение программы и автоматически выводит на терминал сообщение об ошибке, печатая его в следующей форме: ?XX AT *mm.nn* (если ошибка встретилась в строке программы) или ?XX (если ошибка встретилась в директивной строке). Здесь ?XX — цифровой код ошибки, а *mm.nn* — номер строки программы, в которой обнаружена ошибка.

В зависимости от реализации (и желания пользователя) за этим кодовым сообщением выводится текстовое сообщение об этой ошибке. Код ошибки (и текст) указывает на причину, по которой она произошла, а номер строки ускоряет ее нахождение и исправление.

Например, пользователь подготовил и запустил следующую программу:

```
*1.10 SET A=2; TYPE %1,"A=",A,!
*1.20 SET B=4; TYPE "B=",B,!
*1.30 GOTO 1.01
*G
A= 2
B= 4
?03 AT 01.30
*
```

Сообщение об ошибке в строке 1.3 вызвано передачей управления на строку с номером 1.01, которой в программе нет.

Коды ошибок отличаются для различных реализаций ФОКАЛа и их списки даются в приложении 3.

## 2.7. СТАНДАРТНЫЕ ФУНКЦИИ. ОПЕРАТОР XECUTE

Стандартные функции в языке ФОКАЛ являются процедурами-функциями, т. е. в процессе своего выполнения получают значение и, следовательно, могут использоваться подобно переменным в арифметических выражениях. Библиотека стандартных функций ФОКАЛа зависит от реализации и включает не менее 10—12 функций.

Обращение к стандартным функциям осуществляется по имени, первым символом которого обязательно является символ F. За именем процедуры следуют аргументы, заключенные в скобки. У некоторых стандартных функций аргументов нет. В этом случае непосредственно за открывающей скобкой следует закрывающая. В качестве аргументов можно использовать числа, переменные, выражения или стандартные функции.

Отметим, что стандартные функции можно использовать только совместно с операторами ФОКАЛа. Например,

```
*SET Z=A+FSQT(FSIN(X))
*
```

Эта запись эквивалентна математической записи

$$z = a + \sqrt{\sin(x)}.$$

**Оператор XECUTE.** Среди стандартных процедур- функций языка ФОКАЛ есть функции, которые помимо передачи вычисленного в процессе своей работы результирующего значения выполняют определенные операции ввода и (или) вывода на терминал, на общую шину компьютера и т. д. В тех случаях, когда пользователя интересует не значение, которое принимает такая процедура- функция, а производимые ею операции ввода—вывода, для вызова и выполнения функции применяется оператор XECUTE, который имеет следующий синтаксис:

```
<оператор выполнения>::= X[ECUTE] <вызов
```

```
      процедуры-функции>.
<вызов процедуры-функции>::= <имя функции>
      ([<список фактических параметров>]).
```
(2.42)

Например, в результате выполнения директивной строки

```
*XECUTE FCHR(65)
A*
```

на терминал будет выведен символ А, которому соответствует десятичный код 65 в КОИ-7.

**Тригонометрические функци**и. Функция FSIN. Функция FSIN (X) используется для вычисления синуса угла X, заданного в радианах. Например,

```
*TYPE %,FSIN(3.14159),!
-0.953674E-06
*
```

Для преобразования значения угла из градусов в радианы используется множитель $\pi/180$

```
FSIN(X*3.14159/180)
```
(2.43)

где *X* — значение угла в градусах.

Значение функции FSIN (X) находится в диапазоне —1 ... +1, а аргумент должен лежать в интервале $-10^{38}$ ... $+10^{38}$.

**Функция FCOS.** Функция FCOS (X) используется для вычисления косинуса угла X, заданного в радианах. Например,

```
*TYPE %,FC0S(2*3.14159)
0.100000E+01
*
```

Значение функции находится в интервале —1 ... +1, а аргумент может принимать значения из интервала $-10^{38}$ ...$+10^{38}$. Для указания угла в градусах используется формат, аналогичный (2.43).

**Функция FATN.** Функция арктангенс FATN (X) используется для вычисления угла (в радианах), тангенс которого задается выражением X. Например,

```
*TYPE %,FATN(1),!
0.785398E+00
*
```

Аргумент функции арктангенс может принимать любые значения из интервала —$10^{19}$ ... +$10^{19}$, а значения функции лежат в интервале —π/2 ... π/2 или —90 ... +90°.

*Замечание.* Идентификатор функции арктангенс может незначительно отличаться в разных реализациях.

**Элементарные математические функции. Функция FEXP.** Эта функция используется для вычисления значений экспоненциальной функции $\mathbf{e}^x$ (**e** = 2,71828). Например,

```
*TYPE %,FEXP(-1),!
0.367879E+00
*
```

(функция FEXP — обратная по отношению к функции FLOG).

**Функция FLOG.** Эта функция используется для вычисления натурального логарифма выражения X (X > 0). Например,

```
*TYPE %1,FLOG(1),!
0.000000E+00
*
```

(функция FLOG — обратная по отношению к функции FEXP).

**Функция FLOG10.** Эта функция используется для вычисления десятичного логарифма выражения X (X > 0). Для перехода к вычислению по любому основанию можно использовать тождество

$$\log_a X = \log_b X / \log_b a.$$

Например, программа, приведенная ниже, вычисляет десятичный и двоичный логарифм вводимого числа A:

```
*1.10 ASK "ВВЕДИТЕ ЧИСЛО",A
*1.20 SET B=FLOG10(A); SET C=B/FLOG10(2)
```

```
*1.30 TYPE "ДЕСЯТИЧНЫЙ ЛОГАРИФМ ",A
*1.35 TYPE "РАВЕН",B,!
*1.40 TYPE "ДВОИЧНЫЙ ЛОГАРИФМ ",A
*1.45 TYPE " РАВЕН",C,!
```

*Замечание.* Идентификаторы функций натурального и десятичного логарифмов могут по-разному записываться в различных реализациях языка ФОКАЛ.

**Функция FSQT.** Эта функция используется для вычисления квадратного корня из выражения X ($X \geq 0$). Например,

```
*TYPE %1,FSQT(9),!
3
*
```

**Функция FABS.** Эта функция используется для получения абсолютного значения выражения X. Например,

```
*TYPE %3,FABS(-66),!
 66
*
```

Подпрограмма вычисления абсолютного значения использует следующее соотношение:

$$\text{FABS}(X) = \begin{cases} -X, & \text{если } X < 0; \\ 0, & \text{если } X = 0; \\ X, & \text{если } X > 0. \end{cases}$$

Чаще всего функция FABS (X) используется для нахождения модуля разности двух чисел, если их относительная величина неизвестна.

**Функция FSGN.** Эта функция используется для определения знака выражения X. Подпрограмма вычисления функции FSGN использует следующее соотношение:

$$\text{FSGN}(X) = \begin{cases} -1, & \text{если } X < 0; \\ 0, & \text{если } X = 0; \\ 1, & \text{если } X > 0. \end{cases}$$

Функция FSGN применяется главным образом в тех вычислениях, в которых не могут использоваться числа определенного знака. Иллюстрацией этого утверждения служит следующий пример.

*Пример.* Вычислим кубический корень из любого числа. При этом на этапе извлечения кубического корня (возведения в степень 1/3) знак «отделяется» от числа, а потом «добавляется» к результату:

```
*1.10 ASK "ВВЕДИТЕ ЧИСЛО",A
*1.20 SET B=FSGN(A); SET C=FABS(A)
*1.30 SET D=B*C^(1/3)
*1.40 TYPE "КУБИЧЕСКИЙ КОРЕНЬ ИЗ ",A
*1.45 TYPE " РАВЕН",D,!
```

**Функция FITR.** Эта функция используется для получения целой части выражения X (с учетом знака). Например,

```
*TYPE %3,FITR(5.2),FITR(-4.9),!
  5 -4
*
```

Функция FITR может также использоваться для выделения дробной части вещественного числа. Это достигается вычитанием целой части из исходного числа:

```
*1.10 SET B=FITR(A); SET C=FABS(A-B)
*1.20 TYPE "ЦЕЛАЯ ЧАСТЬ ЧИСЛА ",A
*1.25 TYPE " РАВНА",B,!
*1.30 TYPE "ДРОБНАЯ ЧАСТЬ ЧИСЛА ",A
*1.35 TYPE " РАВНА",C,!
```

**Функция ввода—вывода символов FCHR.** Функция FCHR используется для ввода и (или) вывода символов, соответствующих кодам КОИ-7. Эта функция при вводе преобразует символы КОИ-7 в десятичную форму, а при выводе — из десятичной формы в символы КОИ-7. Функция FCHR всегда оперирует с одним символом через устройство ввода—вывода. Ее можно использовать в двух операторах: выполнения XECUTE FCHR (список аргументов) и присваивания SET A = FCHR (список аргументов).

Аргументами функции могут быть как отрицательные, так и положительные числа. Когда аргумент отрицателен, функция будет считывать следующий символ с устройства ввода. При этом

значение функции равно десятичному значению кода КОИ-7 считываемого символа. Например, *SET A = FCHR(—1)

В данном примере с устройства ввода принимается символ (пусть это будет символ Q), а переменной A присваивается значение 81, соответствующее десятичному значению кода символа Q. Если аргумент функции неотрицателен, то целая часть искомого десятичного числа, взятая по модулю 256, преобразуется в соответствующий код КОИ-7, который передается на устройство вывода. Значение функции при этом равно ближайшему целому числу, не превосходящему значение аргумента.

Например, в результате выполнения каждой из двух директивных строк

```
*XECUTE FCHR(86)
V*
*SET X=FCH(86.6)
V*
```

напечатается символ V, и при выполнении второй строки переменной X присвоится значение 86.

Как следует из формата обращения, функция FCHR допускает использование нескольких аргументов. В этом случае она последовательно производит необходимые операции ввода—вывода для каждого аргумента, а ее значение будет равно десятичному значению кода КОИ-7 последнего аргумента. Например, по директиве

```
*XECUTE FCHR(85,86,87,-1)
UVW*
```

будут выведены три символа U, V, W, соответствующие десятичным значениям кодов КОИ-7, указанных аргументами функции, и введен один символ с устройства ввода.

Десятичное значение кода введенного символа становится значением самой функции FCHR.

Аргументом функции FCHR может быть любое арифметическое выражение, в том числе содержащее обращение к функции FCHR, т. е. функция FCHR может использоваться рекурсивно. Рассмотрим пример рекурсивного использования функции FCHR для одновременного ввода и печати символов A, B, C, D, E на терминале:

```
*1.10 SET J=0
```

```
*1.20 SET J=J+1
*1.30 IF (J-5)2.10,2.20,1.50
*1.50 QUIT
*2.10 XECUTE FCHR(FCHR(-1))
*2.15 G 1.2
*2.20 SET Z=FCHR(FCHR(-1))
*2.30 TYPE !,Z,!
*G
ABCDE
69
*
```

При запуске этой программы директивная строка с номером 2.10 выполняется четыре раза, каждый раз при этом на терминале печатается тот символ, который вводится пользователем. Предположим, что пользователь последовательно вводит символы A, B, C, D, E. Действительно, при выполнении первой вложенной функции FCHR с устройства ввода задается символ A, а сама функция принимает десятичное значение, соответствующее коду этого символа; в результате выполнения второй функции FCHR печатается заданный символ A. При выполнении строки 2.20 на терминале печатается символ E, а следующая строка выводит значение переменной Z.

Иногда бывает удобно использовать функцию FCHR в операторе TYPE. Однако при таком ее использовании следует помнить, что на терминал будут выведены результаты работы как функции FCHR, так и оператора TYPE. Результат работы функции — пересылка кодов символов на терминал, результат работы оператора TYPE — вывод на терминал значения функции. Например,

```
*TYPE FCHR(65)
A  65*
*TYPE FCHR(FCHR(-1))
B  66*
```

В первой директивной строке функция FCHR выводит символ A, а оператор TYPE печатает значение этой функции, равное 65. При выполнении второй директивной строки с устройства ввода считывается какой-либо символ, например B, который и выводится на терминал вместе со своим десятичным кодом.

В заключение необходимо отметить, что функция FCHR с успехом применяется при анализе символа или строки символов. Например, ее можно использовать для анализа правильности ответов в обучающей программе, содержащей вопросы и ответы:

```
*1.10 TYPE "В ЛАТИНСКОМ АЛФАВИТЕ ЧИСЛО БУКВ:"
*1.20 TYPE !," A)32; B)26; C)22",!
*1.30 TYPE "ВАШ ОТВЕТ?"
*1.40 SET REQ=FCHR(FCHR(-1))
*1.50 IF (REQ-66)2.10,3.10,2.10
*2.10 TYPE !,"ОТВЕТ НЕВЕРЕН, ПОПРОБУЙТЕ ЕЩЕ"
*2.15 TYPE "РАЗ",!
*2.15 GOTO 1.3
*3.10 TYPE !,"ПРАВИЛЬНО",!
*G
В ЛАТИНСКОМ АЛФАВИТЕ ЧИСЛО БУКВ:
A)32; B)26; C)22
ВАШ ОТВЕТ? A
ОТВЕТ НЕВЕРЕН, ПОПРОБУЙТЕ ЕЩЕ РАЗ ВАШ ОТВЕТ? B
ПРАВИЛЬНО
*
```

**Генерация случайных чисел.** Функция FRAN предназначена для получения псевдослучайных чисел в диапазоне —1 ... +1 с равномерным законом распределения и нулевым математическим ожиданием.

Синтаксис функции FRAN имеет вид

```
SET <переменная>=FRAN([1])
```

или

```
XECUTE FRAN([1])
```

Для генерации одной и той же последовательности случайных чисел необходимо выполнить функцию FRAN (1) (например так: XECUTE FRAN (1)). Это объясняется наличием в интерпретаторе системной (внутренней) переменной, которая изменяется при

обращении к функции FRAN вида FRAN ( ), но устанавливается в начальное состояние при обращении вида FRAN (1).

Приведем пример обращения к функции FRAN ( ) без аргумента:

```
*1.10 TYPE "ГЕНЕРАЦИЯ СЛУЧАЙНЫХ ЧИСЕЛ",!
*2.10 FOR J=1,50; SET A(J)=FRAN()
*2.20 FOR J=1,10; DO 3
*2.30 QUIT
*3.10 FOR K=1,5; TYPE A[5*(J-1)+K]
*3.20 TYPE !
*
```

В результате выполнения этой программы генерируется последовательность из 50 псевдослучайных чисел, которые записываются в индексные переменные с именем A (строка с номером 2.10) и печатаются на устройстве вывода до пяти в каждой строке. Чтобы повторить ту же последовательность случайных чисел, необходимо обратиться к функции FRAN следующим образом: XECUTE FRAN (1) и вновь выполнить приведенную выше программу.

Приведем пример программы генерации последовательности случайных чисел в интервале A ... B:

```
*1.05 TYPE "ЭТА ПРОГРАММА ГЕНЕРИРУЕТ 15 СЛУЧ"
*1.07 TYPE "АЙНЫХ ЧИСЕЛ В ЗАДАННЫХ ГРАНИЦАХ",!
*1.10 ASK "ВВЕДИТЕ НИЖНЮЮ ГРАНИЦУ",A,!
*1.20 ASK "ВВЕДИТЕ ВЕРХНЮЮ ГРАНИЦУ",B,!
*1.30 FOR J=1,15; DO 3
*1.40 QUIT
*3.10 SET X=A+[FRAN()+1]*(B-A)/2
*3.20 TYPE X,!
```

**Функция FSBR. Специальная переменная &.** В языке ФОКАЛ, реализованном на ЭВМ с системой команд PDP-11 (ДВК-2, СМ-4, «Электроника БК-0010» и т. п.), введена *специальная переменная*, обозначаемая знаком амперсанда &. Эта переменная используется функцией FSBR, программируемой пользователем. Пользователь может использовать эту специальную переменную независимо от функции FSBR как

обычную простую переменную, например,

```
*4.10 SET &=A(L);SET A(L)=A(L+1); SET A(L+1)=&
```

Функция FSBR позволяет пользователю организовать любую часть своей программы как подпрограмму. Функция FSBR может использоваться или в арифметических выражениях, как любая стандартная функция, или в операторе XECUTE; формат обращения к ней имеет вид:

```
SET V=FSBR(mm.nn,ARG)
```

или

```
XECUTE FSBR(mm.nn,ARG),
```

где V — идентификатор переменной, которой присваивается значение функции; *mm.nn* — номер первой строки программы пользователя, реализующей алгоритм программируемой функции FSBR; ARG — аргумент функции FSBR (переменная или выражение).

Программируемая пользователем функция — процедура-функция с одним формальным параметром, роль которого играет специальная переменная & . В результате обращения к функции *последнее вычисленное значение специальной переменной* становится ее значением.

Интерпретатор ФОКАЛа, обнаружив в тексте программы обращение к функции FSBR, вычисляет значение аргумента функции, присваивает полученное значение специальной переменной & и передает управление на строку программы с номером *mm.nn*. Возврат из подпрограммы, реализующей алгоритм программируемой функции, происходит по оператору RETURN или по исчерпанию строк в группе, например,

```
*1.10 SET A=5; SET B=2
*2.50 SET Y=2*FSBR(5,A+B)
*2.70 TYPE Y; QUIT
*5.10 SET &=(&+1)^2; RETURN
```

```
*G
 128.0000*
```

В этом примере при обращении к функции FSBR вычисляется значение ее аргумента A + B, которое автоматически присваивается переменной &, это значение используется в строке с номером 5.10; вычисленное значение переменной &, равное $(A + B + 1)^2$, возвращается в качестве окончательного значения функции FSBR. В конце работы программы печатается значение переменной Y, равное удвоенному значению функции.

Указатель номера первой строки программы, реализующей алгоритм программируемой функции, может быть не только числом, но и переменной или переменной с индексом, например,

```
*1.10 A "L<0 - ВЫЧИТАНИЕ, L>=0 - СЛОЖЕНИЕ",L,!
*1.20 I (L)1.3; S MM=10; G 2.10
*1.30 S MM=15
*2.10 S B=10; S C=FSBR(MM,B); T C,!; Q
*10.10 C ПОДПРОГРАММА СЛОЖЕНИЯ
*10.15 S &=MM+B
*15.10 C ПОДПРОГРАММА ВЫЧИТАНИЯ
*15.15 S &=MM-B
*G
L<0 - ВЫЧИТАНИЕ, L>=0 - СЛОЖЕНИЕ:-1
   5.0000
*
```

В данном примере в зависимости от значения L формируется значение MM и функция FSBR принимает одно из двух значений: (MM — B) или (MM + B); выход из функции осуществляется по исчерпанию соответствующей группы строк. Если *mm.nn* является указателем группы (т. е. *nn* не указывается), то строка с минимальным номером из этой группы должна быть первой строкой программируемой функции; если строк этой группы недостаточно для записи алгоритма, то можно использовать строки других групп, передавая им управление оператором DO. Это объясняется тем, что функция FSBR, передавая управление группе или строке, работает как оператор DO и, исчерпав данную группу или строку, возвращает управление в то место программы, которое идет за вызовом FSBR. Например, программа вычисления фак-

ториала $f$ (N) = N1 = 1·2·...·N с помощью итеративного цикла и рекуррентных соотношений:

$$f(0) = 1;$$

$$f(k) = k \cdot f(k—1),\ k \neq 0$$

имеет вид

```
*1.05 T "ЗАДАЙТЕ N ДЛЯ ВЫЧИСЛЕНИЯ N-ФАКТОРИАЛ"
*1.10 A N,!
*1.20 S K=FSBR(5,N)
*1.30 T "ЗНАЧЕНИЕ ",N," ФАКТОРИАЛ =",K,!; Q
*5.10 S H=1; F J=1,&; D 6.1
*5.20 S &=H; R
*6.10 S H=H*J
*G
ЗАДАЙТЕ N ДЛЯ ВЫЧИСЛЕНИЯ N-ФАКТОРИАЛ:5
ЗНАЧЕНИЕ 5.0000 ФАКТОРИАЛ = 120.0000
*
```

Значение искомой функции присваивается переменной K. Оператор DO в строке с номером 5.10 передает управление строке с номером 6.10 и тем самым позволяет совместить две различные группы строк (группы строк 5 и 6) в одном описании программируемой функции.

ГЛАВА 3

# РЕАЛИЗАЦИЯ ЯЗЫКА ФОКАЛ НА БЫТОВОМ ПЕРСОНАЛЬНОМ КОМПЬЮТЕРЕ «ЭЛЕКТРОНИКА БК-0010»

Реализация базового ФОКАЛа на БК расширена новыми операторами и стандартными функциями. Введенные операторы обеспечивают работу с кассетным магнитофоном (оператор LIBRARY), инициализацию пускового монитора (оператор PASS) и вывод справочной информации (HELP, VACANT). Набор стандартных функций дополнен обратными тригонометрическими функциями, функциями формирования графических примитивов, управления курсором (FT, FV, FK) и функцией работы с портом ввода— вывода (FP).

## 3.1. ОРГАНИЗАЦИЯ ПРОГРАММ. ЭЛЕМЕНТЫ ДАННЫХ

**Нумерация строк.** Для нумерации строк программы реализация ФОКАЛа на БК (в дальнейшем ФОКАЛ—БК) разрешает кроме номеров от 1.01 до 99.99 использовать номера от 100.1 до 127.9 с шагом 0.1, за исключением тех, которые оканчиваются нулем (т. е. нельзя использовать номера типа 110.0 и т. п.).

**Числа.** Числа в ФОКАЛе—БК изменяются по абсолютной величине приблизительно от $0{,}15 \cdot 10^{-38}$ до $1{,}7 \cdot 10^{38}$. При выходе значений переменных из этого диапазона ФОКАЛ—БК выдает сообщение об ошибке.

**Переменные.** В ФОКАЛе—БК могут использоваться как переменные с одним индексом (D3 (J), A1 (15)), так и переменные с двумя индексами (S (K, L), BN (7, —3)). Индекс может изменяться в диапазоне —32768 ... +32767 для переменных с одним индексом и —128 ... +127 для переменных с двумя индексами. Индекс может быть и числом и переменной. При выходе индекса за диапазон его значение изменяется с учетом особенностей представления положительных и отрицательных чисел в БК. В соответствии с рис. 3.1, *а* переменная L (129,1) есть не что иное, как L (127 + 2,1) или L (—127,1), а переменная L (—139,1) есть L ( — 128 — 11,1) или L (117,1). То же самое справедливо для переменных с одним

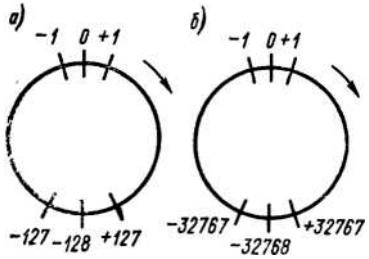



**Рис. 3.1. Диаграмма изменения значений индексов. — для переменных с одним индексом**

индексом (рис. 3.1, *б*).

*Замечания.* 1. Хотя индекс может изменяться в большом диапазоне, память, выделяемая для таблицы переменных в ФОКАЛе—БК, организуется так, что в ней можно разместить не более 2000 переменных (с индексами или без них). 2. Следует иметь в виду, что интерпретатор ФОКАЛа—БК не делает различия между переменной с нулевым индексом и такой же переменной без индекса, т. е. для ФОКАЛа— БК переменные *A* (0) и *A* тождественны:

```
*S A(0)=1; S A=5; T A(0),!
   5.0000
*
```

3. Все переменные в таблице переменных хранятся как переменные с двумя индексами. Например,

```
*S L(7)=5; S L(129)=3; S L(257)=4
*T ¤
S L(+07,+00)=  5.0000
S L(-127,+00)=  3.0000
S L(+01,+01)=  4.0000
*
```

(3.1)

Преобразование линейного индекса (257) в двойной индекс (+01, +01) (последняя строка в таблице переменных в 3.1) происходит по следующему алгоритму. Второй индекс равен целой части от деления линейного индекса на число 256 (FITR (257/256) = 1), а значение первого индекса есть остаток от этого деления, преобразованный в соответствии с диаграммой на рис. 3.1, *а.* Например,

```
*S L(510)=3; T ¤;
S L(-02,+01)=  3.0000
*
```

Если при выполнении программной или директивной строки переменная (любая) встречается впервые (т. е. до этого она отсутствовала в таблице переменных), то интерпретатор помещает

ее в таблицу либо со значением, которое присваивается ей в операторах SET, ASK или FOR, либо с *нулевым* значением. Например,

```
*1.10 S A=A+1; S B=3
*1.20 T "A=",A,!,"B=",!,"C=",C,!
*G
A=   1.0000
B=   3.0000
C=   0.0000
*
```

При выполнении этой программы переменная *A,* которая впервые встречается в *левой* части оператора SET, будет помещена в таблицу переменных с нулевым значением, а потом увеличена на единицу (A = A + 1). Переменная B помещается в таблицу со значением 3, а переменная C (которая впервые появилась в операторе TYPE) — с нулевым значением.

## 3.2. ОПЕРАТОРЫ ЯЗЫКА

**Оператор TYPE.** Оператор TYPE ¤ — последний выполняемый в строке оператор. Все расположенные за ним операторы никогда не выполнятся:

```
*S K=1; T ¤; T "ПРИМЕР ОПЕРАТОРА T ¤"
S K()=  1.0000
*
```

**Оператор DO.** При организации вложенных подпрограмм с помощью оператора DO возможно переполнение области памяти, используемой ФОКАЛом—БК для управления ходом выполнения программы. Число вложений оператора DO не превышает 43.

**Оператор FOR.** Оператор цикла FOR I = N1, N2, N3; <тело цикла> характеризуется в ФОКАЛе—БК следующими свойствами.

1. Операторы из области действия цикла выполняются по крайней мере один раз:

```
*F K=15,1,10; T K
   15.0000*
```

2. Переменная цикла I при выполнении оператора FOR должна возрастать, т. е. должны выполняться условия N3 > N1 и N2 > 0!

При N3 < N1 и любом значении шага цикла N2 операторы из области действия цикла выполнятся ровно один раз:

```
*F K=3,-1,1; T K
   3.0000*
```

Чтобы организовать цикл с убывающим значением переменной цикла (т. е. при N3 < N1 и N2 < 0) можно, например, поступить так:

```
*F L=1,M; S K=M-L+1; <ТЕЛО ЦИКЛА>
```

Здесь при изменении значения переменной L от 1 до M значение переменной K будет меняться от M до 1.

При N3 ≥ N1 и N2 < 0 цикл становится бесконечным:

```
*F K=5,-1,10; T K,!
   5.0000
   4.0000
   3.0000
   ...
```

3. При выходе из цикла значение переменной цикла равно последнему значению, при котором выполнялось тело цикла, плюс шаг:

```
*F K=1,0.3,2; COMMENT ТЕЛО ЦИКЛА ПУСТО
*T K
   2.2000*
```

**Оператор COMMENT.** При трассировке программы (см. п. 2.6) на печать выводится сам оператор COMMENT и текст комментария до первого ограничителя:

```
*1.10 S P=2; ?T P; C КОММЕНТАРИЙ
*G
T P;   2.0000 C*
```

Если при трассировке необходимо печатать весь комментарий, то вместо пробела можно использовать, например, символы «—» или « :»:

```
*1.10 S P=2; ?T P; C-КОММЕНТАРИЙ
*G
T P;   2.0000 C-КОММЕНТАРИЙ*
```

**Оператор VACANT.** Программа занимает в памяти компьютера какое-то число ячеек. Оператор VACANT позволяет пользователю в любой момент времени узнать число свободных ячеек в памяти:

```
*V[ACANT]
СВОБОДНО XXXXXX Б.ОЗУ
```

где XXXXXX — восьмеричное число, указывающее число ячеек памяти в байтах, доступное для размещения программы.

Оператор VACANT не требует операндов.

**Оператор KILL.** Оператор *сброса внешних устройств* K[ILL] предназначен для установки всех внешних устройств в исходное состояние. При его выполнении обнуляются буферные регистры и сбрасываются разряды разрешения прерывания. При этом выполняется инструкция ассемблера RESET. У оператора KILL нет операндов. При его выполнении в программном режиме в регистре состояния клавиатуры сбрасывается разряд разрешения прерывания от клавиатуры, поэтому прервать выполнение программы становится невозможным до ее окончания.

**Оператор HELP.** Этот оператор выводит на экран дисплея общий список операторов и функций языка ФОКАЛ, а также следующую информацию: ШАГ — приостанов выполнения программы; ВВОД — продолжение; СТОП — останов выполнения программы; СБР — сброс экрана; НР/СБР — управление размером памяти и экрана — сброс выполнения строки.

**Оператор PASS.** Этот оператор передает управление из ФОКАЛа пусковому монитору.

## 3.3. РАБОТА С МАГНИТОФОНОМ

Оперативная память современных компьютеров имеет одно «неприятное» (особенно для персональных компьютеров) свойство: при выключении питания, информация, находящаяся в памяти, исчезает. Для сохранения подготовленных программ и (или) данных (т. е. результатов работы этих программ),

информацию *сохраняют* на внешних носителях, таких как магнитные диски, магнитные ленты, перфоленты, перфокарты и т. п., откуда она может быть в любой момент снова *введена* в память компьютера.

«Электроника БК-0010» имеет разъем, который позволяет подключать к ней в качестве внешнего запоминающего устройства бытовой магнитофон, а в ФОКАЛе—БК существует группа операторов LIBRARY, предназначенная для работы с ним. Точнее говоря, операторы группы LIBRARY предназначены для управления обменом информацией между памятью БК и магнитной лентой.

Информацию, которую операторы группы LIBRARY помещают на магнитную ленту, принято называть *файлами*. Каждому файлу присваивается *имя* (произвольно выбираемая пользователем последовательность символов), которое однозначно идентифицирует связанную с ним информацию (файл) и используется для ввода этой информации в память БК.

Синтаксис операторов группы LIBRARY следующий:

**<оператор работы с магнитофоном>::= L[IBRARY] <ОПР> [<имя файла>].**

(3.2)

Здесь (ОПР) — подоператор, который указывает, какой именно оператор группы LIBRARY будет выполняться. В данном синтаксисе возможны подоператоры, которые выполняют следующие действия: S [AVE] — выводит текст находящейся в памяти программы в файл с указанным именем; G [ET] — загружает в память программу из указанного файла (если в памяти находилась программа, то она будет уничтожена); F [GET] — просматривает магнитную ленту и выводит на экран телевизора имена файлов; O [UTPUT] — выводит находящуюся в памяти БК таблицу переменных в указанный файл; I [NPUT] — помещает в память таблицу переменных, считывая ее из указанного файла; M [OTOR ] — включает двигатель магнитофона; R [ESET ] — выключает двигатель магнитофона.

Последние два оператора будут выполняться только на магнитофоне, которым можно управлять дистанционно.

Указываемое в операторе группы LIBRARY имя файла может

состоять из любых алфавитно-цифровых символов (в том числе строчных и прописных букв русского алфавита). Максимальная длина имени файла 15 символов. При неверном задании имени файла интерпретатор выводит сообщение об ошибке.

Рассмотрим работу операторов группы LIBRARY.

**Оператор LIBRARY SAVE.** Этот оператор имеет следующий вид:

```
L[IBRARY] S[AVE] <имя файла>
```

и записывает в файл с указанным именем весь текст программы, находящейся в памяти БК.

Можно рекомендовать следующий порядок задания оператора LIBRARY SAVE:

1) наберите оператор вывода L S <имя файла>, но не нажимайте на клавиатуре БК клавишу «ВВОД», например,

```
*L S TEST1                                    (3.3)
```

2) нажмите на панели магнитофона клавиши «ВОС-ПРОИЗВЕДЕНИЕ» и «ЗАПИСЬ»;

3) нажмите на клавиатуре БК клавишу «ВВОД» — теперь оператор (3.3) начал выполняться.

Когда вывод текста программы в файл завершится, интерпретатор выведет на экран звездочку:

```
*L S TEST1 <ВВОД>

*
```

**Оператор LIBRARY GET.** Этот оператор имеет следующий вид:

```
L[IBRARY] G[ET] <имя файла>
```

и вводит программу, находящуюся в указанном файле, в память БК.

Допустим, мы хотим занести в память программу, сохраненную ранее оператором L S в файле Test1. Чтобы осуществить это, нужно:

1) перемотать магнитную ленту так, чтобы магнитная головка оказалась перед тем участком ленты, на котором записан файл;

2) набрать оператор L G <имя файла> и нажать клавишу «ВВОД»

```
*L G TEST1
```
(3.4)

3) нажать на панели магнитофона клавишу «ВОСПРОИЗВЕДЕНИЕ», после чего интерпретатор начинает выполнять оператор (3.4).

Выполнение оператора (3.4) происходит следующим образом. Компьютер читает имена всех встречающихся на магнитной ленте файлов и сравнивает их с именем, указанным в операторе LIBRARY GET (в нашем примере с именем Test1). Если имена не совпадают, то имя встретившегося на магнитной ленте файла выводится на экран терминала, файл в память не считывается и поиск требуемого файла продолжается. Когда встречается искомый файл, то его имя на экран не выводится, а файл считывается в память компьютера. Если операция чтения выполнилась без ошибки, то на экран выводится символ «*», в противном случае выдается сообщение об ошибке:

```
ОШИБКА КОНТРОЛЬНОЙ СУММЫ
```

и операцию чтения файла с магнитной ленты необходимо повторить сначала.

*Замечание.* При вводе оператора L G в компьютер вся информация, находившаяся до этого в памяти, исчезает (как программа, так и таблица переменных) независимо от того, выполнялось или нет фактическое считывание информации с магнитной ленты!

**Оператор LIBRARY FGET.** Этот оператор имеет следующий синтаксис:

```
L[IBRARY] F[GET] <произвольное имя>
```

и выводит на экран имена файлов, записанных на магнитной ленте, т. е. оператор L F распечатывает каталог магнитной ленты.

Выполнение оператора L F (в отличие от оператора L G) не разрушает содержимое памяти компьютера.

Синтаксис оператора L F требует задания имени файла, но так как при выполнении оператора чтение файла не выполняется, то имя может быть любым, например однобуквенным, а его совпадение с одним из находящихся на магнитной ленте имен файлов не оказывает никакого влияния на работу оператора L F.

**Оператор LIBRARY OUTPUT** обеспечивает вывод данных из таблицы переменных в указанный файл на магнитную ленту и имеет следующий синтаксис: L [IBRARY] O [UTPUT] <имя файла>

Сохранение данных на магнитной ленте может оказаться полезным (если не необходимым) в одной из следующих ситуаций.

1. Объем вводимых с клавиатуры исходных данных достаточно велик (например, для каких-либо вычислений необходимо ввести массив из 100 значений):

```
*1.10 F J=1,100; T "L(",%3,J,")"; A L(J),!
```
(3.5)

После однократного выполнения строки 1.10 и ввода всех 100 значений массива программист может вывести всю таблицу переменных в файл на магнитную ленту:

```
*L O DATA1
*
```
(3.6)

При последующих запусках программы данные можно будет ввести не с клавиатуры, а с магнитной ленты.

2. Необходимо прервать работу программы с большим временем выполнения (например, порядка часа). Чтобы не повторять весь расчет сначала, данные можно вывести на магнитную ленту, а потом, загрузив в директивном режиме и программу и ее данные, продолжить выполнение программы.

3. В процессе своей работы программа генерирует данные в таком количестве, что целиком они не помещаются в таблицу переменных. В этом случае программу можно организовать так, чтобы после генерации (например, каждой 1000 значений) она приостанавливала свою работу. После этого программист выводит таблицу переменных в очередной файл, стирает таблицу

переменных с помощью оператора ERASE и возобновляет работу программы.

При выводе данных на магнитную ленту полезно присваивать файлу такое имя, из которого было бы видно, во- первых, что это файл данных, а во-вторых, к какому программному файлу он относится. Формируя имя файла данных, необходимо придерживаться следующей методики:

1) использовать составное имя, первая часть которого повторяет имя программного файла, относящегося к нему; вторая часть есть последовательность символов «DAT» (от DATA — данные); третья, необязательная часть может быть порядковым номером (в том случае, если для одной программы необходимо иметь несколько файлов данных);

2) для повышения читаемости имени файла использовать как прописные, так и строчные буквы.

Например, если программа, содержащая строку из примера (3.5), сохранена в файле Program, то введенный массив данных удобно сохранить в файле ProgramDAT.

Оператор L O выводит всю таблицу переменных.

**Оператор LIBRARY INPUT.** Этот оператор имеет следующий синтаксис: L [IBRARY] I [NPUT] _<имя файла>, который ищет на магнитной ленте файл данных с указанным именем и считывает содержимое файла в таблицу переменных.

При правильном завершении чтения данных на экран выведется символ «*». Если в памяти компьютера до задания оператора L I существовала таблица переменных, то после правильного выполнения оператора L I:

1) значения переменных, идентификаторы которых отсутствуют в файле данных, в таблице переменных не изменяются;

2) переменные, идентификаторы которых есть в файле данных, примут новые значения (введенные из файла данных);

3) переменные, отсутствовавшие в таблице переменных, будут помещены в нее со своими значениями.

На текст находящейся в памяти программы оператор L I не влияет.

*Пример.* Возвращаясь к примерам (3.5), (3.6), продемонстрируем одну из возможностей использования оператора L I. После вывода подготовленных данных в файл Data1, программист может заменить строку 1.10 в программе с тем, чтобы в дальнейшем вводить данные из этого файла. Лучше всего сделать это следующим образом:

```
1.10 T "ПОДГОТОВЬТЕ МАГНИТОФОН ДЛЯ ВВОДА ФАЙЛА DATA1"
*1.11 T " И НАЖМИТЕ ЛЮБУЮ КЛАВИШУ>"; X FCHR(-1); T !
*1.12 L I DATA1
```

*Замечание.* Порядок работы с магнитофоном при использовании операторов L O и L I такой же, как для операторов L P и L G соответственно.

Использование операторов L [IBRARY] M [OTOR] и L [IBRARY] R [ESET] создает пользователю определенные удобства в работе и позволяет экономно расходовать магнитную ленту при записи программ и данных, но требует аппаратной доработки магнитофона.

**Рекомендации по работе с магнитофоном.** Исходя из опыта своей работы на БК с магнитофонами различных марок, авторы хотят отметить следующее.

1. Для работы с БК можно использовать практически любой (не только кассетный) магнитофон при единственном условии: для устойчивой работы канала чтения- записи в магнитофоне должна быть отключена *автоматическая регулировка уровня записи* (АРУЗ). Как правило, в магнитофонах 3-го класса на лицевой панели присутствует тумблер отключения АРУЗ.

2. Прежде чем использовать магнитофон в качестве внешнего запоминающего устройства, его необходимо настроить на *оптимальный режим записи* так, чтобы обеспечить устойчивую работу при вводе—выводе программ и данных. Это осуществляется опытным путем с помощью регулятора уровня записи магнитофона.

3. Для повышения надежности работы с информацией на магнитной ленте необходимо *дублировать* запись этой информации, помещая ее в ряд последовательных файлов с одинаковым именем. Для этого можно рекомендовать, например, использование директивной строки вида *FOR K = 0,1; L S <ИМЯ ФАЙЛА>, выполнение которой приведет к двукратной записи программы на магнитную ленту в файл с указанным именем.

4. Объем магнитной ленты позволяет записывать на нее несколько десятков программ.

Как правило, на магнитофонах устанавливается счетчик метража магнитной ленты. Наличие счетчика позволяет пользователю вести рукописный каталог, в который (при записи файла на магнитную ленту) заносятся имя файла и показание счетчика относительно начала ленты. Такой каталог позволяет *максимально быстро* находить требуемый файл, используя

клавишу «Перемотка вперед» или «Перемотка назад» (разумеется, до начала работы магнитная лента должна быть установлена на начало, а счетчик обнулен).

Если перемотать магнитную ленту примерно в то место, где записан требуемый файл, набрать на клавиатуре БК оператор L G <имя файла> и нажать клавишу «ВОСПРОИЗВЕДЕНИЕ», то компьютер будет просматривать ленту и, когда встретит указанный файл, считает его в память. Недостаток этого подхода очевиден — это ошибка в оценке местоположения файла может существенно увеличить время его поиска.

*Замечание.* На выполнение операторов L G или L F нажатие клавиш «Стоп», «Перемотка вперед» или «Перемотка назад» не влияет. Поэтому при поиске файла в любой момент можно «подогнать» ленту в ту или иную сторону, заменить кассету и т. д.

Одним из оригинальных методов организации файлов на магнитной ленте является использование речевых меток файла. Практически на всех современных магнитофонах есть встроенный микрофон. Совмещая стандартную процедуру записи программ на магнитную ленту и использование микрофонного входа, можно перед каждым файлом помещать речевую информацию, содержащую название программы, ее назначение и т. п. При поиске программы достаточно включить «Воспроизведение» и выставить громкость. Услышав название программы, можно нажать клавишу «Стоп», набрать на клавиатуре БК оператор L G <имя файла> и ввести программу в память. Запись речевой информации на магнитную ленту может быть выполнена в стандартном режиме записи используемого магнитофона.

5. Кроме магнитных лент, на которые пользователь записывает информацию сам, ему зачастую приходится иметь дело с лентами, записанными на «чужих» магнитофонах. При этом не исключена ситуация (вызываемая так называемой «несовместимостью головок записи — чтения»), при которой прочесть такую магнитную ленту не удается. Единственным выходом из этого положения может быть «подстройка» считывающей головки магнитофона. Следует иметь в виду, что это нестандартный режим магнитофона и его следует использовать в крайнем случае.

Подстройка считывающей головки выполняется следующим образом. При фиксированном положении клавиши «Громкость» и нажатой клавише «Воспроизведение», вращая регулирующий крепежный винт считывающей головки, необходимо добиться

максимальной громкости звука. После этого операцию считывания файла в память можно повторить. Если и на этот раз она завершится неудачно, то лучшее, что можно сделать — это вернуть ленту ее владельцу.

## 3.4. РЕДАКТИРОВАНИЕ ТЕКСТА

Когда в компьютер вводится текст программы или эта программа отлаживается, возникает необходимость выполнить редактирование текста, т. е. удалить, вставить или заменить какие-нибудь символы. В ФОКАЛе—БК реализованы удобные средства, которые позволяют редактировать текстовую строку, набранную, но не введенную (т. е. клавиша ВВОД еще не нажата) или вызванную на редактирование оператором MODIFY.

Прежде чем рассмотреть средства редактирования, сделаем несколько замечаний, касающихся реализации оператора MODIFY. Как уже отмечалось в п. 2.6, он может задаваться только в директивной строке и должен быть в ней последним оператором. Задание оператора MODIFY не приводит к стиранию таблицы переменных. Указание в операторе MODIFY несуществующего номера строки вызовет сообщение об ошибке. И, наконец, оператор MODIFY *не позволяет* редактировать номер строки!

При вводе текстовой информации и ее редактировании на экране необходимо отмечать то место, куда может быть выведен очередной символ. Для этого служит специальный символ — курсор. Если курсор указывает на позицию, не занятую другим символом, то это просто светящийся прямоугольник; если же на этой позиции уже есть какой-нибудь символ, то он виден на фоне курсора. Для редактирования строки используются восемь клавиш редактирования, расположенных справа на клавиатуре БК и окрашенных (в большинстве своем) в желтый цвет. Рассмотрим назначение этих клавиш.

Клавиши «→», «←», «ВС» и «ГТ» предназначены для управления положением курсора. Однократное нажатие клавиш «→» или «←» перемещает курсор на одну позицию вправо или влево соответственно. Нажатие клавиши «ВС» перемещает курсор в начало строки, а клавиша «ГТ» вызывает горизонтальную табуляцию, т. е. сдвиг курсора на восемь позиций вправо. Перемещение курсора ничего не меняет в редактируемой строке; оно позволяет указать место в строке, в котором можно что-то

изменить.

Клавиши «|—>», «<—|», «<+» и «СБР |—>» выполняют собственно редактирование текста.

Клавиша «СБР |—>» позволяет удалить текст, начиная с позиции курсора, до конца редактируемой строки. При нажатии клавиши «<+» удаляется символ слева от курсора, при этом курсор смещается на одну позицию влево; при нажатии клавиши «|—>» происходит «раздвигание» строки: курсор остается на месте, а все символы, начиная с позиции курсора, сдвигаются на одну позицию вправо; при нажатии клавиши «<—|» происходит смыкание строки: курсор остается на месте, символ в позиции курсора удаляется, а все символы, расположенные справа от курсора, сдвигаются на одну позицию влево.

Таким образом, при необходимости удалить какие-либо символы можно использовать клавиши «<—|», «<+» или «СБР |—>». Для вставки символов в любое место строки необходимо сначала освободить для них место с помощью клавиши «|—>», а затем набрать на клавиатуре требуемые символы. Если какой-либо символ необходимо заменить, то на этот символ нужно установить курсор и нажать на клавиатуре желаемый символ.

Редактирование можно завершить в любой момент, независимо от положения курсора, нажав клавишу «ВВОД».

## 3.5. СТАНДАРТНЫЕ ФУНКЦИИ

Библиотека стандартных функций расширена в ФОКАЛе—БК обратными тригонометрическими и другими элементарными функциями, функциями формирования графических примитивов, управления курсором и работы с линейными часами, портом ввода—вывода и общей шиной (всего включает 21 функцию).

**Элементарные математические функции.** Для вычисления натурального и десятичного логарифмов в ФОКАЛ—БК включены функции FLOG и FLOG10 соответственно.

В число прямых тригонометрических функций включена функция тангенс (FTAN), вычисляющая тангенс угла, заданного в радианах. Аргумент функции может быть арифметическим выражением. Например,

```
*T %,FTAN(3.141592/4),!
0.100000E+01
```

```
*
```

В библиотеку стандартных функций ФОКАЛа—БК включены три обратные тригонометрические функции — *арккосинус* (FACOS), *арксинус* (FASIN) и *арктангенс* (FATAN).

Функция арккосинус вычисляет угол (в радианах), косинус которого задается как аргумент функции. Значение аргумента функции должно лежать в диапазоне —1 ... +1. Если значение аргумента по модулю больше 1, то выдается сообщение об ошибке, например:

```
*T %,FACOS(0),!
0.157079E+01
*
```

Функция арксинус вычисляет угол (в радианах), синус которого задается как аргумент функции. Значение аргумента функции должно находиться в диапазоне —1 ... +1; выход за диапазон вызывает сообщение об ошибке, например:

```
*T %,FASIN(1),!
0.157079E+01
*
```

Функция арктангенс вычисляет угол (в радианах), тангенс которого задается как аргумент функции. Аргумент функции арктангенса может принимать любые значения из интервала $-10^{19}$ ... $10^{19}$, а значения функции лежат в интервале $-\pi/2$ ... $\pi/2$ (—90° ... +90°), например:

```
*T %,FATAN(1.0E+19),!
0.157079E+01
*
```

**Функция управления общей шиной FX.** Эта функция используется для организации работы с периферийными устройствами и обращения к ячейкам памяти и имеет следующий синтаксис:

```
<функция управления общей шиной>::=          (3.7)
```

**FX(<КОП>,<адрес ОШ>[,<выражение>]).**

Здесь КОП — код операции, который может принимать значения —1, 0, 1 и определяет тип выполняемой операции (—1—запись значения ВЫРАЖЕНИЯ по АДРЕСУ общей шины (ОШ); функция FX принимает десятичное значение записываемой величины; 0 — операция «ЛОГИЧЕСКОЕ И» над содержимым ячейки памяти по АДРЕСУ ОШ и значением ВЫРАЖЕНИЯ; функция FX принимает десятичное значение результата операции; 1 — чтение содержимого ячейки памяти по АДРЕСУ ОШ; третий аргумент может опускаться; функция FX принимает десятичное значение прочитанной ячейки).

Второй аргумент функции FX — АДРЕС ОШ — представляет собой восьмеричное число или идентификатор переменной и должен быть четным. АДРЕС ОШ указывает адрес ячейки памяти или регистра периферийного устройства, к которому производится обращение.

Третий аргумент используется в операциях типа —1 и 0 и должен быть десятичным числом или арифметическим выражением в диапазоне —32 768 ... 32 767.

Запись в ячейки памяти с адресами 0—$2000_8$ *запрещена*, поскольку они используются интерпретатором для своей работы. При попытке записи в запрещенную область памяти выдается сообщение об ошибке.

*Примеры:*

1)

```
*T FX(1,177662)
  10.0000*
```

выводит содержимое ячейки с адресом $177662_8$ — регистра данных клавиатуры;

2)

```
*X FX(-1,2002,5)
*
```

записывает число пять в ячейку с адресом $2002_8$;

3)

```
*T FX(0,40000,177)
  0.0000*
```

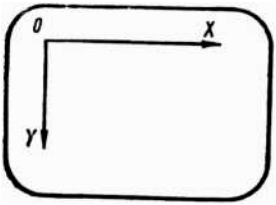



**Рис. 3.2. Система координат перемещений курсора**

выводит содержимое младшего байта ячейки по адресу $40000_8$.

**Функция управления курсором FK.** Эта функция предназначена для установки курсора в точку с координатами (X, Y) на плоскости экрана и имеет следующий формат: FK (X, Y).

Координаты X и Y могут быть арифметическими выражениями, десятичные значения которых должны удовлетворять следующим условиям: —63 ≤ X ≤ 63; —23 ≤ Y ≤ 23.

Изменение положения курсора на экране осуществляется относительно левого верхнего угла экрана, имеющего координаты (0,0) (рис. 3.2), причем положительное направление оси X — слева направо, а оси Y — сверху вниз. Экран имеет 24 позиции по вертикали и 64 — по горизонтали.

*Примеры:*

1) оператор **`*X FK(32,12)`** поместит курсор в центр экрана переместив его на 32 позиции вправо по оси X и на 12 позиций вниз по оси Y относительно левого угла экрана;

2) оператор **`*X FK(-32,-12)`** тоже поместит курсор в центре экрана, причем смещение будет происходить как бы от правого нижнего угла на 32 позиции влево по оси X и на 12 позиций вверх по оси *Y*.

Функция FK принимает десятичное значение координаты Y.

**Функции формирования графических примитивов FT и FV.** Эти функции имеют одинаковый формат:

**`FT(<КОП>,X,Y), FV(<КОП>,X,Y)`**

и в зависимости от значения кода операции КОП формируют или стирают на экране точку или вектор (отрезок прямой) соответственно (КОП = 0 — стирание точки или вектора на экране; КОП = 1 — формирование точки или вектора на экране).

Для *функции формирования точки* FT X и Y — это декартовые десятичные координаты *точки,* а для *функции формирования вектора* FV — это декартовые десятичные координаты *конца вектора.* Координаты отсчитываются от левого верхнего угла экрана, имеющего координаты (0,0) (см. рис. 3.2), и могут задаваться арифметическими выражениями, десятичные значения которых должны удовлетворять следующим условиям: $0 \le X \le 511$; $0 \le Y \le 239$. При построении на экране вектора, координаты его начала либо предварительно задаются с помощью функции FT, либо определяются значением последней выполненной функции FV, т. е. другими словами, при последовательном построении нескольких отрезков прямой начало последующего вектора есть конец предыдущего.

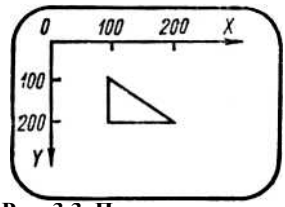


**Рис. 3.3. Построение треугольника**

*Пример.* Построение треугольника:

```
*1.10 T "ПОСТРОЕНИЕ ТРЕУГОЛЬНИКА",!
*1.20 X FT(0,100,100)
*1.30 X FV(1,100,200); X FV(1,200,200)
*1.40 X FV(1,100,100)
```

Оператор в строке 1.20 задает координаты начальной точки, а операторы в строке 1.30 последовательно проводят отрезки a, *b, c* (рис. 3.3).

Функции FT и FV принимают десятичное значение координаты Y.

**Функция случайного числа FRAN.** Эта функция используется для генерации псевдослучайных чисел в интервале —1 ... 1 с равномерным законом распределения и нулевым математическим ожиданием. Формат функции: FRAN ([1]).

Период последовательности этих чисел равен $2^{15} = = 32\,768$. Это значит, что выдав последовательность из $2^{15}$ чисел, функция FRAN начнет повторять ее снова. Если для решения (или отладки) задачи необходимо повторение одной и той же последовательности случайных чисел, то необходимо выполнить функцию FRAN (1)

(например, X FRAN (1)), которая установит начальное значение для этой последовательности. Это объясняется наличием системной переменной, которая изменяется при каждом обращении к функции FRAN вида FRAN ( ), но устанавливается в исходное состояние при обращении вида FRAN (1).

*Пример.* Генерация случайных чисел на интервале (A, B):

```
*1.10 T "ПРОГРАММА ГЕНЕРАЦИИ N СЛУЧАЙНЫХ"
*1.15 T " ЧИСЕЛ НА ОТРЕЗКЕ (A,B)",!
*1.20 A "ЗАДАЙТЕ НИЖНЮЮ ГРАНИЦУ",A
*1.30 A "ЗАДАЙТЕ ВЕРХНЮЮ ГРАНИЦУ",B
*1.40 A "ЗАДАЙТЕ КОЛИЧЕСТВО СЛУЧАЙНЫХ ЧИСЕЛ",N
*1.50 F J=1,N; DO 2; QUIT
*2.10 S X=A+[FRAN()+1]*(B-A)/2
*2.20 T %,X,!
```

В ФОКАЛе—БК реализация функции FRAN имеет два недостатка. О первом уже упоминалось — это короткий период последовательности случайных чисел; второй недостаток — это корреляция последовательных пар случайных чисел. Предположим, нужно сформировать двумерное случайное поле — случайные точки ($x$, $y$), равномерно распределенные на экране в прямоугольнике $0 \le x \le 511$, $0 \le y \le 239$. Для этого формируем два последовательных случайных числа: первое будет координатой $x$, а второе — $y$. Точки с этими координатами будем выводить на экран с помощью следующей программы:

```
*1.10 X FCHR(12); C ОЧИСТКА ЭКРАНА
*1.20 S X=[FRAN()+1]*511
*1.30 S Y=[FRAN()+1]*239
*1.40 X FT(1,X,Y); G 1.2                    (3.8)
```

Вместо ожидаемого равномерного заполнения экрана получается совсем не случайный узор, в чем может убедиться каждый, кто запустит приведенную программу на БК. Несложная модификация программы (3.8) позволяет получить двумерное случайное поле с достаточно равномерным заполнением экрана:

```
1.10 X FCHR(12); C ОЧИСТКА ЭКРАНА
*1.20 F J=1,500; S X(J)=[FRAN()+1]*511
*1.30 F J=1,500; S Y(J)=[FRAN()+1]*239
```

```
*1.40 F J=1,500; X FT(1,X(J),Y(J)); G 1.2
```

Эта программа использует в качестве координат точек не последовательные пары случайных чисел, а пары, отстоящие друг от друга на 500 случайных чисел, что устраняет корреляцию «ближнего порядка».

Существенно лучше работает датчик случайных чисел, описанный в [20]. Выдаваемые им числа равномерно располагаются в промежутке 0 ... 1 в соответствии с алгоритмом:

$$r_n = Cr_{n-1} \bmod m,\ r_0 = 1,\ \mathrm{RND_n} = r_n/m,$$

где RND — текущее случайное число, $C = 65\,539$; $m = 2^{31}$.

Длина последовательности случайных чисел $RND_n$ равна $2^{39}$ и в единичном квадрате этот алгоритм дает равномерное распределение точек.

К сожалению, реализация этого алгоритма на ФОКАЛе не возможна из-за недостаточной точности представления чисел.

**Функция работы с портом ввода—вывода FP.** Порт (16-разрядный программируемый интерфейс) — это техническое устройство компьютера, предназначенное для управления периферийным оборудованием. Порт имеет разъем «УП», находящийся на задней стенке корпуса БК, через который пользователь может подключить требуемое периферийное оборудование.

Разъем «УП» имеет 64 штырьковых контакта, с частью которых связаны два 16-разрядных регистра порта, один из которых (*выходной регистр порта*) доступен только по записи, а другой (*входной регистр порта*) — только по чтению.

Регистр порта — это устройство компьютера, которое позволяет записывать и считывать в (из) него информацию как в ячейку памяти. Все разряды в регистрах пронумерованы от 15 до 0, что схематично можно изобразить следующим образом:

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

(15-й разряд называют старшим; нулевой — младшим, так как в них хранятся старший и младший разряды двоичного числа соответственно). Соответствие между разрядами входного и выходного регистров порта и контактами разъема УП приведено в табл. 3.1.

Таблица 3.1

**Соответствие разрядов входного и выходного регистров контактам разъемного порта**

| Разряд входного регистра | Контакт разъема порта | Разряд выходного регистра | Контакт разъема порта | Разряд входного регистра | Контакт разъема порта | Разряд выходного регистра | Контакт разъема порта |
|---|---|---|---|---|---|---|---|
| 0 | B24 | 0 | A16 | 8 | B31 | 8 | A20 |
| 1 | A24 | 1 | A13 | 9 | A31 | 9 | A28 |
| 2 | B23 | 2 | B12 | 10 | B23 | 10 | A27 |
| 3 | B17 | 3 | B10 | 11 | A32 | 11 | B27 |
| 4 | B20 | 4 | B5 | 12 | B30 | 12 | A26 |
| 5 | A20 | 5 | B7 | 13 | A29 | 13 | B26 |
| 6 | B22 | 6 | B6 | 14 | B29 | 14 | A25 |
| 7 | A23 | 7 | A7 | 15 | A30 | 15 | B25 |

Примечания. 1. Общие контакты разъема УП: A11, B11, A18, B18, A19, B19; остальные контакты использовать нельзя. 2. A — верхний ряд контактов, B — нижний.

Для обмена информацией (сигналами) между устройством пользователя, подключенным через разъем к порту, и компьютером используется функция FP, имеющая формат: FP (<КОП>, <маска>), где КОП — код операции (он может принимать следующие значения: 0 — чтение регистра ввода по маске; 1 — очистка битов регистра вывода по маске; 2 — установка битов регистра вывода по маске; 3 — чтение регистра вывода по маске).

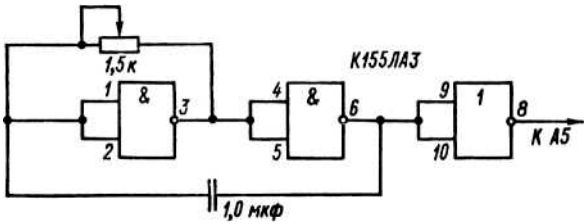



**Рис. 3.4. Схема генератора импульсов**

Маска — *восьмеричное число* в диапазоне 0 ... 177777 или *переменная* с десятичным значением в диапазоне —32768 ... 32767. При операциях чтения с регистра ввода (КОП = 0) и регистра

вывода (КОП = 3) выполняется поразрядная конъюнкция (операция «ЛОГИЧЕСКОЕ И») между считываемой с регистра информацией и маской. Если маска равна 177777, то читаются все 16 разрядов; если маска равна 377, то читается младший байт, а если 177400, то старший байт регистра. При КОП = 1 выполняется очистка (обнуление) битов регистра вывода по маске, т. е. обнуление битов регистра вывода происходит в соответствии с теми битами маски, которые установлены в единицу. Для очистки всего регистра маска должна быть равна 177777; для очистки младшего байта — 377, а старшего байта — 177400. При КОП = 2 выполняется установка битов регистра вывода по маске. Установка битов регистра вывода происходит в соответствии с теми битами маски, которые установлены в единицу. Для установки всех битов регистра вывода маска должна быть равна 177777; для установки в единицу битов младшего байта — 377, а старшего — 177400.

Сама функция FP принимает десятичное значение считанного или записанного слова, например:

```
*X FP(2,177777); T FP(3,40)
32.0000*
```

Первый оператор директивной строки выполняет установку всех битов регистра вывода порта в единицу. Второй оператор считывает бит 4 в регистре вывода порта и так как он был установлен в единицу, то функция принимает десятичное значение 32 (= $40_8$).

**Функция измерения времени FCLK.** В число допустимых функций ФОКАЛа—БК входит функция FCLK ( ), позволяющая проводить измерение интервалов времени, выполняя обработку прерываний от некоторого генератора синхроимпульсов — таймера. Однако реальное использование функции FCLK сдерживается отсутствием в архитектуре БК такого устройства. Применение функции FCLK становится возможным, если подключить к БК какой-нибудь генератор импульсов. На рис. 3.4 приведена схема простейшего генератора и его подключения к БК. Импульсы, подаваемые генератором на вход *A5* системной магистрали (например, с частотой 50 Гц) (см. табл. 6.4) будут вызывать прерывание работы процессора по вектору прерывания $100_8$ каждые 20 мс; при этом ФОКАЛ—БК ведет подсчет возникших прерываний, а обращение к функции позволяет

получить их число.

Программа, приведенная ниже, позволяет определить частоту работы генератора-таймера и в случае необходимости помогает настроить его на требуемую частоту;

```
*1.10 T !,"ВКЛЮЧИТЕ ТАЙМЕР И НАЖМИТЕ ЛЮБУЮ "
*1.15 T "КЛАВИШУ."
*1.20 T !,"ПО ИСТЕЧЕНИИ 30 СЕК НАЖМИТЕ КЛАВИШУ"
*1.25 T " ПРОБЕЛ."
*1.30 X FCHR(-1); S T1=FCLK()
*1.40 I [FX(1,177662)-32]1.4,1.5,1.4
*1.50 T !,"ВАШ ТАЙМЕР РАБОТАЕТ С ЧАСТОТОЙ "
*1.55 T (FCLK()-T1)/30," ГЦ"
```

## 3.6. ДИАГНОСТИКА ОШИБОК

Если при выполнении программы или директивной строки интерпретатор ФОКАЛа—БК обнаруживает ошибку, то он выводит сообщение об ошибке в виде:

```
?ll AT mm.nn
<текстовое сообщение>
```

где *ll* — номер ошибки, *mm.nn* — номер строки программы (или 00 для директивной строки), в которой была обнаружена ошибка.

Текстовое сообщение служит для объяснения ошибки, например:

```
*T FSQT(-4),!

?17 AT 0.00
КОРЕНЬ КВАДРАТНЫЙ ИЗ
ОТРИЦАТЕЛЬНОГО ЧИСЛА
```

Список ошибок с текстовыми сообщениями приведен в приложении 2.

ГЛАВА 4

# МЕТОДОЛОГИЯ ПРОГРАММИРОВАНИЯ НА ФОКАЛе

## 4.1. НЕОБХОДИМОСТЬ ПРОЕКТИРОВАНИЯ ПРОГРАММ

Простота языка ФОКАЛ в сочетании с диалоговым режимом работы на персональном компьютере являются определенной преградой для начинающего программиста. Чаще всего изучение ФОКАЛа ведется непосредственно на компьютере, без какой-либо предварительной подготовки. Новые конструкции или операторы могут быть опробованы в прямом режиме для понимания их работы и определения возможных ограничений. Интерпретатор позволяет быстро выявлять и исправлять ошибки, особенно в маленьких и простых программах. Все эти удобства имеют и отрицательную сторону, так как очень часто программист создает программу непосредственно на экране дисплея, без ее предварительного обдумывания. Когда программа состоит из 20—30 строк, а программист имеет определенный опыт работы, то это вполне допустимо, особенно если алгоритм достаточно прост и понятен. Для более длинной и сложной программы такой подход неприемлем даже для опытных программистов. Составление такой программы затрудняется, в частности, из-за ограниченного размера экрана: невозможность просмотра всего текста оказывается реальным источником ошибок. Создание программы превращается в процесс непрерывного внесения поправок и вставок. В результате возникает запутанная и ненадежная программа, которая часто не соответствует исходному заданию. Это неизбежно приводит к обязательному требованию — разрабатывать (проектировать) программу точно так же, как проектируется любая машина или устройство. При проектировании конечный результат (программа) получается не сразу, а как итог последовательных этапов разработки.

Первый этап в решении любой задачи (не только в программировании) — это ее постановка, т. е. точное словесное описание проблемы; второй этап — выбор и последовательное уточнение алгоритма, который реализует требуемые функции. При

этом решая сложную задачу, необходимо разбить ее на подзадачи (модули), а затем выбрать способы и средства для решения выделенных подзадач. На следующем этапе происходит реализация алгоритма на выбранном языке программирования, т. е. создание пригодной к выполнению программы. В идеальном случае работа программиста на этом и завершается, так как выполнение этой программы на компьютере должно привести к требуемым результатам. Фактически, из-за сложности задач и ограниченности человеческих возможностей, первоначально написанная программа, как правило, содержит ошибки. Поэтому следующими этапами являются отладка (устранение ошибок, допущенных при программировании) и тестирование (проверка на правильность работы программы в соответствии с ее постановкой). Последние два этапа могут выполняться итеративно (выявленные при очередном тестировании ошибки исправляются и тестирование происходит вновь) до тех пор, пока не возникнет уверенность в том, что программа работает правильно.

Уточним некоторые понятия, которые будут использоваться в дальнейшем изложении.

*Постановка* — словесное описание работы программы, формата входных и выходных данных, возможных режимов ее работы, начиная от защиты от ошибок в задании данных до требований к удобству работы оператора.

*Промежуточная форма описания* (программы) — форма, в которой описывается функционирование программы, полученной на некотором этапе проектирования. Такой формой может быть, например, блок-схема или описание программы на русском языке или на каком-либо промежуточном языке. В частности может быть и описание на языке ФОКАЛ, в некоторых местах которого операторы языка заменены комментариями на русском языке. Такую форму иногда называют псевдокодом.

*Псевдокод* — программа, частично написанная на ФОКАЛе, в которой некоторые части написаны на обычном разговорном языке или каком-либо вспомогательном, отличном от ФОКАЛа. Описание разных частей (например, подпрограмм) может быть различным — от совсем краткого объяснения функционирования до развернутого алгоритма в словесной форме.

*Модуль* — часть программы, которая может быть спроектирована, отлажена и тестирована независимо от других частей (модулей). Хотя модули часто оформляются как подпрограммы, знак равенства между ними ставить нельзя.

Общий подход к проектированию и разработке программ можно сформулировать следующим образом.

1. Продвигайтесь небольшими шагами. Не пытайтесь сделать слишком много за один шаг.

2. Разбивайте большие задачи на небольшие, логически независимые подзадачи, с тем, чтобы: 1) их можно было проверять и отлаживать отдельно и 2) изменения, которые делаются в одной из них, как можно меньше влияли на другие.

3. Организуйте поток управления как можно проще, чтобы облегчить нахождение ошибок при отладке и тестировании.

4. Используйте по-возможности наглядное или графическое описание. Оно удобнее для восприятия, чем словесное описание. В этом состоит существенное преимущество блок-схем.

5. Придавайте особое значение простоте и ясности. Производительность программы можно увеличить (если это необходимо) после того, как она начнет функционировать.

6. Не испытывайте судьбу. Либо не применяйте те методы, в которых вы не уверены, либо используйте их очень осторожно. Обращайте внимание на ситуации, которые могут привести к путанице, и вносите в них ясность насколько возможно.

7. Помните, что программа должна быть отлажена, проверена и должна сопровождаться. Планируйте эти завершающие этапы.

8. Применяйте простые и последовательные приемы и методы. При разработке программ повторяемость в той же мере не является недостатком, в какой сложность не является достоинством.

9. Прежде чем начинать запись программы в окончательном виде полностью завершите ее проектирование. Не поддавайтесь соблазну скорее начать запись инструкций: это имеет не больше смысла, чем разводка печатных плат до того, как вы будете точно знать, что будет в системе.

10. Будьте особенно внимательны к факторам, которые могут изменяться. Выполняйте реализацию так, чтобы было легко вносить изменения.

## 4.2. ТЕХНИКА ПРОГРАММИРОВАНИЯ

Техника программирования включает те приемы, которые облегчают работу программиста и обеспечивают возможность достижения высокого качества конечного программного продукта. Эти приемы включают следующие моменты: 1) постановку задачи — задание спецификаций программы; 2) промежуточные формы описания — словесную, блок-схемы, псевдокод и др.; 3) методы программирования — нисходящее, восходящее, структурное и модульное программирование и т. д.; 4) стиль программирования — выбор имен переменных, использование комментариев, достижение читаемости программы.

Прежде чем переходить к подробному обсуждению этих приемов, рассмотрим сквозной пример для иллюстрации и сопоставления рассматриваемых методов.

**Сквозной пример.** Как уже отмечалось, ФОКАЛ — интерпретатор, т. е. система, ориентированная на взаимодействие (диалог) с человеком. При разработке программ, требующих вмешательства пользователя, следует учитывать следующие факторы: 1) какие процедуры ввода наиболее естественны для пользователя; 2) может ли пользователь легко определить начало, продолжение и конец операции ввода; 3) как пользователь узнает о процедурных ошибках и сбоях оборудования; 4) какие ошибки вероятнее всего допустит пользователь; 5) как пользователь узнает, что данные были введены правильно; 6) какой должна быть форма вывода, чтобы пользователь мог легко воспринимать и понимать ее; 7) адекватна ли реакция программы действиям пользователя; 8) удобна ли программа в работе; 9) предусмотрены ли обучающие возможности для неопытного пользователя и приемлемые возможности для опытного пользователя; 10) всегда ли пользователь может определить или восстановить состояние программы после прерываний или сбоев.

Воспользуемся этим подходом при описании сквозного примера. На рис. 4.1 показана простая система, в которой источником ввода является единственный ключ. В ответ на замыкание ключа процессор должен включить на одну секунду светодиодный индикатор.

*Ввод в системе «ключ/индикатор»*: вводимые данные — это один бит, значение которого может быть равно либо «0» (ключ замкнут), либо «1» (ключ открыт); вводимые данные не надо запрашивать, они доступны, по крайней мере, через несколько

миллисекунд после замыкания ключа; ввод будет изменяться, как правило, не чаще, чем один раз в несколько секунд; процессор должен опрашивать ключ, чтобы определить замыкание; очевидными ошибками ввода являются поломка ключа, разрыв во входной цепи и попытка пользователя замкнуть ключ раньше, чем пройдет некоторое время; вводимые данные не зависят от какого-либо другого ввода или вывода.


**Рис. 4.1. Система «ключ — индикатор»**

*Вывод в системе «ключ/индикатор»*: выводимые данные — это один бит, который должен иметь значение «О», чтобы индикатор включился, или «1», чтобы он погас; данные должны измениться на противоположные через одну секунду; возможными ошибками вывода являются поломка индикатора и разрыв в выходной цепи; вывод зависит только от ввода с ключа и времени.

*Исполнительная часть* программы реализуется просто. Как только ввод от ключа становится равным логическому нулю, программа включает индикатор (выводит логический нуль) на одну секунду.

Рассмотрим возможные ошибки и сбои: нажатие ключа до истечения одной секунды; поломка ключа, индикатора или компьютера.

Наиболее вероятна первая ошибка. Простейшее решение состоит в том, чтобы программа игнорировала нажатие ключа, пока не истечет одна секунда. Этот промежуток столь незначителен, что пользователь его не заметит. Более того, игнорирование нажатия ключа во время этого интервала означает, что не нужно никакой аппаратной или программной обработки «дребезга».

Перейдем к рассмотрению техники программирования.

**Постановка задачи.** В задании на проектирование программы точно так же, как в задании на проектирование какой-то машины или устройства, необходимо ясно, точно и недвусмысленно ответить на вопрос, какие требования предъявляются к этой программе. Составление правильного и точного задания не такая уж простая задача.

Прежде всего необходимо описать поведение программы при всевозможных сочетаниях входных данных. При этом понятие входные данные нужно воспринимать в самом широком смысле. Например, для программы, которая вычисляет и выводит на печать значение функции, входными данными могут быть начальное и конечное значения и шаг изменения аргумента. Наиболее просто описать поведение программы, которая работает без входных данных. Такая программа называется *автономной*. Автономные программы редко используются отдельно, обычно они являются составной частью более сложной программы. При определении поведения программы, которая работает со входными данными, следует обращать особое внимание на всевозможные критические ситуации и ошибки во входных данных. Одними из разновидностей критических ситуаций являются неопределенности арифметического (математического) типа. Вообще говоря, это не те случаи, когда некоторая функция (в частном случае, арифметическое действие) неопределена при данном значении аргумента. Типичными арифметическими неопределенностями являются, например, переполнение (превышение разрядности получаемого результата размера разрядной сетки компьютера), деление на ноль, извлечение квадратного корня из отрицательного числа и т. д. Некоторые из этих неопределенностей обусловлены чисто машинными ограничениями (например, переполнение), другие же возникают в результате выполнения математических операций. Независимо от причин, вызвавших возникновение подобной ситуации, если только не предприняты специальные меры, компьютер реагирует стандартным образом — прекращает выполнение программы и выводит сообщение об ошибке, например (если в строке 1.5 происходит извлечение квадратного корня из отрицательного числа):

```
?17 AT 01.05
КОРЕНЬ КВАДРАТНЫЙ ИЗ
ОТРИЦАТЕЛЬНОГО ЧИСЛА
```

Неудобство состоит в том, что выполнение программы прекращается и интерпретатор переходит в диалоговый режим. Желательно, чтобы программа выявляла и анализировала критические ситуации. Для этого могут использоваться различные приемы. В нашем примере можно предварительно вычислить

подкоренное выражение и проверить, неотрицательно ли оно. В этом случае программа тоже может вывести сообщение, но уже без прерывания. Из этого примера можно сделать вывод, что *требуется защищать программу от неправильного задания данных и неопределенностей* и это необходимо предусматривать при постановке задачи.

Важным элементом постановки задачи является обеспечение пользователя *максимальным удобством при работе с программой.* Диалоговый режим и развитые возможности языка ФОКАЛ для работы символьной информацией и графикой позволяют предоставить пользователю практически любые требуемые условия.

Исходя из этого рекомендуется: 1) принимать во внимание подготовку пользователя (имеет он или нет опыт работы с компьютером); 2) независимо от подготовки предполагаемого пользователя отдавать предпочтение таким средствам общения с программой, которые не требуют специальной подготовки или, что еще лучше, не требуют вообще никакой подготовки; 3) желательно, чтобы программа сама инструктировала пользователя о виде данных, которые требуется ввести, и о действиях, которые необходимо выполнить для продолжения или завершения работы с данной программой.

Поскольку длинные инструкции значительно увеличивают объем программы, *необходимо стремиться к максимальной лаконичности пояснительного текста.*

Организация диалога между пользователем и программой должна быть предусмотрена при постановке задачи. Существуют несколько стандартных приемов организации диалога. Все их легко реализовать средствами ФОКАЛа. К ним относятся: 1) выбор из меню; 2) ответ «да» или «нет»; 3) ответ числовой или символьной информацией; 4) использование формального языка с заданным синтаксисом; 5) использование подмножества естественного (разговорного) языка.

**Выбор из меню.** В этом случае компьютер выводит на экран список возможных режимов (иногда их называют *опциями),* которые реализованы в программе. Например, в некоторой вычислительной программе может появиться следующее меню:

1. ВЫЧИСЛЕНИЕ ТРИГОНОМЕТРИЧЕСКИХ ФУНКЦИЙ
2. ВЫЧИСЛЕНИЕ ЛОГАРИФМА И ЭКСПОНЕНТЫ
3. ВЫЧИСЛЕНИЕ СТАТИСТИЧЕСКИХ ПАРАМЕТРОВ
4. ПОЯСНЕНИЯ

Чтобы выбрать один из желаемых режимов, пользователь нажимает клавишу с одной из цифр 1—4. Можно организовать иерархию меню, при которой выбор какого- то режима в первом меню приводит к появлению списка возможных услуг (меню) выбранного режима.

**Ответ «да» или «нет».** В некоторых случаях компьютер может задавать вопрос (обычно о выборе некоторого режима), на который можно ответить «да» или «нет». Для ответа могут использоваться как клавиши *Y* (от *YES)* и *N* (от *NO),* так и соответственно Д и Н. Например: нужны ли пояснения? (Д/Н):

**Ответ числовой или символьной информацией.** При «разговоре» с компьютером не на всякий вопрос достаточно дать ответ, содержащий всего один бит информации (т. е. «да» или «нет»). Часто в программу нужно вводить числовые и (или) символьные данные. Например:

```
ВВЕДИТЕ ВАШЕ ИМЯ(НЕ БОЛЕЕ 15 СИМВОЛОВ):
ВВЕДИТЕ ДАТУ(ДД.ММ.ГГ):
ВВЕДИТЕ ЧИСЛО ИЗМЕРЕНИЙ(ДЕСЯТИЧНОЕ ЧИСЛО):
```

**Использование формального языка с заданным синтаксисом.** Его начинают использовать, когда данные описывают объекты со сложной структурой и большим числом параметров. Типичными примерами программ с такими входами являются трансляторы, интерпретаторы и т. п. Формальный язык может использоваться в системах моделирования и автоматизированного проектирования. При таком подходе требуется лексический, синтаксический и семантический анализ введенной конструкции. Этот подход труден для реализации и доступен только опытным программистам, знающим теоретические аспекты системного программирования. Другой недостаток этого подхода состоит в том, что потребителю необходимо изучать синтаксис и семантику входного языка.

**Использование подмножества естественного языка.** Этот подход остается пока в сфере исследовательских разработок, в которых ставится задача обучения компьютера языку, близкому к естественному (русскому, английскому) с ограничениями в предметной (семантической) области и грамматике. Но даже при очень ограниченном словаре (50—100 слов) и конкретной предметной области (например, управление роботами)

соответствующее программное обеспечение очень сложно.

**Промежуточная форма описания. Блок-схемы программ.** Использование блок-схем — наиболее широко известный метод разработки программ. Во многих книгах по программированию можно прочесть, что программисты должны сначала нарисовать блок-схему, а затем начинать писать реальную программу. В действительности, так работают очень немногие программисты, а остальным они кажутся скорее помехой, нежели полезным методом программирования. Мы опишем как достоинства, так и недостатки блок-схем, и покажем диапазон применимости этого подхода при разработке программ.

Основное преимущество блок-схемы — наглядное представление. Многим такие представления кажутся более понятными, чем текстовые описания. Программист может изобразить всю программу и увидеть связи между ее различными частями. Кроме того, блок-схемы: 1) используют стандартные символы (ГОСТ 19.003—80; ГОСТ 19.002—80) (рис. 4.2); 2) понятны без текста программ; 3) могут применяться для разбивки задачи (проблемы) на подзадачи; 4) показывают последовательность операций и, следовательно, могут помочь в обнаружении источника ошибок при отладке реальной программы; 5) широко используются в других областях, а не только в программировании.

Отметим, что существуют средства, помогающие упростить

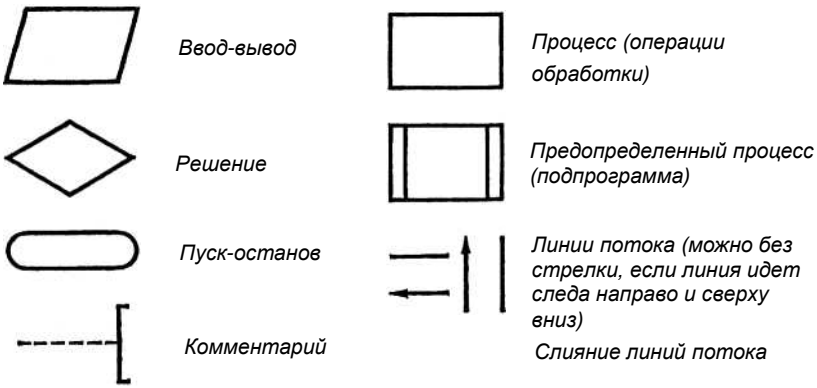



**Рис. 4.2. Стандартные символы блок-схемы**

вычерчивание блок-схемы (лекала для программистов,

автоматические пакеты пририсовок и т. п.). Перечисленные достоинства блок-схем гарантируют их дальнейшее распространение. Однако использование блок- схем как метода разработки программ имеет и недостатки, например: 1) за исключением простейших ситуаций они трудоемки для разработки, рисования и особенно для внесения изменений; 2) нет простого способа отлаживать или проверять их; 3) для больших программ они становятся очень запутанными. (Разработчикам трудно находить «золотую середину» между числом деталей, необходимых, чтобы блок-схема была полезной, и тем их количеством, которое уподобляет по восприятию блок-схему листингу программы); 4) они показывают только организацию программы и не отражают организацию данных или структуру модулей ввода—вывода; 5) их сложно использовать в задачах, связанных с обслуживанием аппаратуры или временных запросов; 6) они допускают неструктурированное программирование. Линии и стрелки возвратов управления и циклы во всей схеме являются прямой противоположностью принципам структурного программирования, хотя и предпринимались попытки избавить блок-схемы от этого недостатка 151.

Таким образом, хотя блок-схемы безусловно полезны, не следует пытаться применять их слишком широко. Они удобны в качестве программной документации, так как состоят из стандартных символов и понятны непрограммистам. Однако как средство проектирования блок-схемы не могут обеспечить больше, чем начальный эскиз; программист не может отлаживать подробную блок-схему, а разработка блок-схемы может оказаться более трудоемкой, чем создание программы.

*Пример. Блок-схема системы «ключ—индикатор».* Эту задачу легко изобразить с помощью блок-схемы (рис. 4.3). Структура данных здесь настолько проста, что ее можно вообще игнорировать. Небольшая трудность возникает при решении о числе требуемых деталей. Блок-схема дает простое изображение понятных процедур.

В этом подходе важно, что блок-схема может игнорировать программные переменные и непосредственно задавать вопросы. Часто бывает необходим компромисс. Так, полезно иметь две версии блок- схемы — одну на естественном языке, которая будет понятна всем, а другую — в терминах программных переменных, которая полезна только программистам.

**Псевдокод.** Основное достоинство блок-схем состоит в наглядности, с которой они представляют управление вычислительным процессом. Во всех других отношениях они примитивны и несовершенны. В последнее время существует тенденция использовать в качестве промежуточной формы не блок-схемы, а псевдокод, который в максимальной степени позволяет использовать средства языка высокого уровня, на котором должна быть написана программа, *Псевдокод — это смесь конструкций языка высокого уровня и разговорного языка.* На последнем описываются те элементы алгоритма, которые еще недостаточно уточнены. По мере уточнения алгоритма происходит замена описательной части псевдокода на конструкции формального языка программирования.

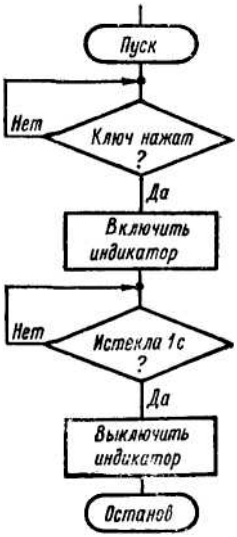



**Рис. 4.3. Блок-схема системы «ключ —индикатор»**

Преимущества псевдокода состоят в его ясности и понятности, а также в быстроте и легкости перехода к программе на языке высокого уровня и очень сильно зависят от используемого языка программирования. Наиболее подходящими для этой цели являются структурные языки программирования (Паскаль, Модула-2, Алгол-68 и т. п.), но и на ФОКАЛе можно создать программы с помощью псевдокода. Для работы с псевдокодом на ФОКАЛе можно руководствоваться следующими правилами и приемами работы.

1. *Включайте текст на разговорном языке.* Часть псевдокода, написанная на разговорном языке, может быть включена в программу вместо эквивалентной конструкции ФОКАЛа или как ее дополнение в качестве комментария. Чтобы отличать их от конструкций языка, можно использовать, например, фигурные скобки:

```
IF {D=0 или D*R<0}
DO {то печать сообщения об ошибке};
DO {нахождение максимального элемента};
SET R={новое значение R};
```

...

2. Используйте *псевдометки.* Нумерация строк в ФОКАЛе, с одной стороны, упрощает редактирование и отладку, но, с другой стороны, затрудняет проектирование программы. Главным образом это относится к ситуациям, когда оператор языка (или псевдокода) ссылается на еще ненаписанный оператор, т. е. происходит *ссылка вперед.* В руководствах по программированию на ФОКАЛе рекомендуется нумеровать строки не подряд, а с некоторым шагом, но если программа большая, а логика алгоритма разветвленная, то ошибки все равно неизбежны.

Наиболее правильный и рациональный подход состоит в том, чтобы на этапе проектирования программы вообще не нумеровать строки! Вместо номера удобно использовать алфавитно-цифровые или символьные метки (как в АЛГОЛе, ФОРТРАНе и др.), помещаемые только перед теми строками, к которым происходит обращение. Эти метки будут указываться в операторах IF, GOTO, DO и других вместо истинных номеров строк. Дополнительное преимущество такого подхода состоит в том, что метки могут нести смысловую нагрузку, объясняя назначение той или иной строки или части программы. При переходе к реальной программе строки последовательно нумеруются (например, с шагом 0.1), а метки в операторах заменяются на номера по простой и очевидной методике. Проиллюстрируем ее на следующем примере:

```
            COMMENT ПСЕВДОКОД
            COMMENT СОРТИРОВКА МАССИВА ПО
            COMMENT ВОЗРАСТАНИЮ
            FOR I=0,10; ASK "A(",I,)=",A(I),!
            SET I=0
{СРАВНЕНИЕ} IF [A(I+1)-A(I)] {РАЗМЕЩЕНИЕ}
            SET I=I+1
            IF (I-10){СРАВНЕНИЕ},{ВЫВОД}
{РАЗМЕЩЕНИЕ}SET V=A(I)
            SET A(I)=A(I+1)
            SET A(I+1)=V
            IF (I){СРАВНЕНИЕ},{СРАВНЕНИЕ}
            SET I=I-1
            GOTO {СРАВНЕНИЕ}
{ВЫВОД}     FOR I=0,10; TYPE A(I),!
```

После нумерации строк (начиная, например, с номера строки 1.10 с шагом 0.05) соответствие меток и номеров будет следующим:

```
{СРАВНЕНИЕ}  = 1.25
{РАЗМЕЩЕНИЕ} = 1.40
{ВЫВОД}      = 1.65
```

Теперь заменяем метки в операторах на соответствующие номера, например, оператор с меткой {СРАВНЕНИЕ} примет вид

```
1.25 F [A(I+1)-A(I)]1.40;
```

3. *Пишите программы-модули на псевдокоде.* Часто используемые подпрограммы можно записать на псевдокоде с метками вместо номеров. Такая подпрограмма перемещаема, т. е. достаточно требуемым образом пронумеровать строки и заменить метки соответственными номерами, чтобы включить ее в любую программу. При этом нужно быть внимательным, чтобы избежать конфликта имен переменных в подпрограмме с именами переменных в основной программе, поскольку в языке ФОКАЛ все имена переменных глобальны.

В заключение отметим, что невозможно отдать безоговорочное предпочтение какой-либо одной форме промежуточного описания — тут многое зависит от типа задачи и привычек программиста; более того, при проектировании программы в разных ее частях могут использоваться различные промежуточные формы.

**Технология разработки программ. Модульное программирование.** Когда программы становятся большими и сложными, блок-схемы перестают быть удовлетворительным средством разработки программ. Однако постановка задачи и ее блок-схема могут прояснить вопрос о том, как разбить задачу на разумные подзадачи. *Разбиение всей программы на подзадачи или модули называется «модульным программированием».*

Проблемы, с которыми разработчик сталкивается в модульном программировании, состоят в том, как разбивать задачу на модули и как объединять модули.

Преимущества модульного программирования достаточно очевидны: 1) единичный модуль легче написать, отладить и проверить, чем всю программу; 2) если модуль выполняет некоторые общие часто встречающиеся операции, то он, вероятно,

окажется полезен и для других программ (таким образом можно сформировать библиотеку стандартных модулей); 3) оно позволяет программисту производить декомпозицию задачи и использовать ранее написанные программы; 4) изменения в программе могут затрагивать только один модуль, а не всю систему; 5) в единичном модуле легче изолировать и выявить ошибки; 6) оно дает некоторое представление о том, насколько продвинулась разработка и что осталось сделать, т. е. помогает планировать работу.

Идеи модульного программирования настолько очевидны и привлекательны, что его недостатки часто игнорируются, тогда как: 1) существенной проблемой может стать объединение модулей, особенно если их писали разные люди; 2) модули требуют тщательного документирования, так как они могут влиять на другие детали программы, такие как структуры данных, используемые всеми модулями; 3) раздельная отладка и проверка модулей трудоемка, так как другие модули могут создавать данные, используемые отлаживаемым модулем, и, кроме того, другие модули могут использовать генерируемые им результаты (можно написать специальные программы именно для порождения требуемых данных и проверки программ, но они требуют дополнительных ресурсов, по сути ничего не добавляя к системе); 4) некоторые задачи очень трудно разбивать на модули (если это все- таки делается, то почти все ошибки и изменения будут затрагивать несколько модулей); 5) модульные программы часто дольше выполняются и занимают большую память, поскольку разные модули могут дублировать некоторые функции.

Разбивая программы на модули, полезно придерживаться следующих принципов: 1) модули, которые ссылаются на общие данные, должны быть частями некоторого охватывающего модуля; 2) два модуля, один из которых использует второй или зависит от него, а второй не зависит и не использует первый, должны быть разделены; 3) модуль, который используется несколькими другими модулями, не должен быть частью того общего модуля, в который они входят; 4) два модуля, один из которых используется многими модулями, *а* второй — только несколькими, должны быть раздельными; 5) если два модуля часто используются, но выполняют существенно разные функции, то они должны быть раздельными; 6) структура или организация связанных с модулем данных должна быть скрыта в этом модуле.

*Пример. Разбиение системы «ключ—индикатор» на модули.* Эта

простая программа может быть разбита на два модуля.

**Модуль 1** ожидает нажатия ключа и в ответ включает индикатор.

**Модуль 2** обеспечивает односекундную задержку.

Модуль 1 будет, вероятно, ориентирован на конкретную систему, так как он зависит от того, какой ключ и индикатор используются. Модуль 2 будет более общим, так как во многих задачах требуются задержки. Ясно, что удобно иметь стандартный модуль задержки, который обеспечивает задержку требуемой длительности. Модуль потребует тщательного документирования, чтобы было понятно, как вызывать модуль и как задавать длительность задержки.

Этот пример слишком прост, но при разработке больших программных систем важно, чтобы каждое проектное решение (такое как скорость передачи в битах, формат сообщения или процедура проверки ошибок) реализовывалось только в одном модуле. Другие модули должны писаться так, чтобы они совершенно не знали те значения, которые выбирались, или те методы, которые использовались при реализации этого модуля. Эта важная концепция известна как *«принцип скрытия информации».* По этому принципу модули разделяют только ту информацию, которая абсолютно существенна для выполнения задачи. Остальная информация скрывается в модулях.

Использование этого принципа полезно при обработке ошибок. В тех случаях, когда модуль обнаруживает неисправимую ошибку, он не должен вызывать процедуры восстановления. Вместо этого он должен передать информацию об ошибке назад, в вызвавший его модуль, который и примет решение об обработке ошибки. Такой подход объясняется тем, что модуль низкого уровня часто не располагает информацией, требуемой для принятия верного решения о том, какие процедуры необходимы для восстановления. Например предположим, что у нас есть модуль, который получает числа, вводимые извне. Этот модуль завершает свою работу правильно, когда строка цифр заканчивается, например, возвратом каретки. Ввод любого нецифрового символа приводит к ненормальному завершению работы модуля. Поскольку модуль не знает, в каком контексте он используется (т. е. является ли он частью ассемблера, интерактивного редактора или системы управления файлами), он не может принять верное решение о том, что он должен предпринять, когда встречается отличный от цифры или возврата каретки символ. Если же в модуле реализован конкретный метод обработки ошибок, то он теряет общность и может использоваться только в тех ситуациях, когда применяется именно

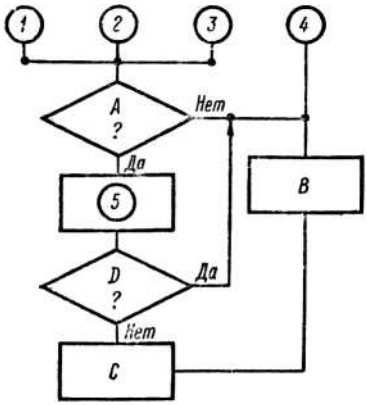



**Рис. 4.4. Блок-схема неструктурированной программы**

этот метод обработки ошибок.

В заключение приведем несколько правил, которых следует придерживаться при модульном программировании:

1) используйте модули, содержащие от 20 до 50 строк (более короткие модули, как правило, бесполезны, а более длинные редко бывают общими и могут быть трудны для подключения);

2) старайтесь делать модули достаточно общими;

3) уделяйте больше внимания тем модулям, которые могут оказаться полезными в других разработках или используются в нескольких местах данной программы;

4) старайтесь, чтобы модули были настолько разными и логически несвязанными, насколько это возможно;

5) не пытайтесь разбивать на модули простые задачи.

**Структурное программирование.** Для программирования модулей полезно применять приемы, облегчающие составление программ, их понимание, отладку, документирование и модификацию, и известные как *структурное программирование*. Этот подход характеризуется тем, что любой модуль (подзадача) проектируется с использованием ограниченного набора программных структур (или просто структур), причем каждая структура имеет единственный вход и единственный выход. Зачем это нужно? Блок-схема неструктурированной программы приведена на рис. 4.4. Если ошибка возникла, например, в блоке B, то ее причиной могут оказаться пять источников. При ее исправлении нужно не только проверить все последовательности взаимодействия частей программы, но и быть уверенным, что любые изменения, сделанные для исправления ошибки, не повлияют на другие части программы и их взаимодействие. Отладка в этой ситуации превращается в борьбу с лернейской гидрой — пока вы отсекаете одну голову, вырастает несколько новых.

Чтобы избежать этого, надо иметь ясную последовательность операций, облегчающую локализацию ошибок. Именно такую

последовательность обеспечивают структуры с одним входом и одним выходом.

Теоретически было доказано, что любую программу можно создать на основе всего трех базовых структур: следования, условия (если ... то ... иначе) и цикла (пока .... выполнять).

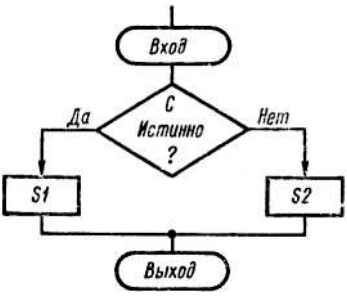



**Рис. 4.5. Блок-схема структуры условия**

*Следование* — это линейная структура, в которой операторы языка или другие структуры выполняются последовательно. Структура следования дает возможность разделить задачу на подзадачи и при их последовательном выполнении решить основную задачу.

*Структура условия* в общем случае имеет вид «если С то S1 иначе S2» и выполняет одну из двух инструкций (или подзадач) S1 и S2 в зависимости от истинности или ложности сформулированного условия С. На рис. 4.5 показана логика этой структуры. Отметим, что структура имеет единственный вход и единственный выход, так что невозможно войти или выйти в (из) подзадачи S1, S2 иначе, чем через структуру.

*Структура цикла* имеет вид «пока С выполнять S» и повторяет некоторую последовательность действий S (называемую *телом цикла*) до тех пор, пока сформулированное условие С истинно. Эта структура, изображенная на рис. 4.6, имеет один вход и один выход. Компьютер никогда не выполнит тело цикла S, если с самого начала условие С ложно, так как значение С проверяется до

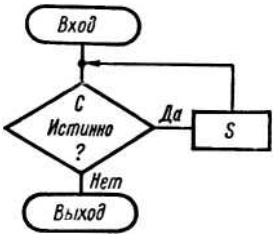



**Рис. 4.6. Блок-схема структуры цикла с предусловием**

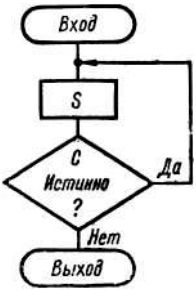



**Рис. 4.7. Блок-схема структуры цикла с постусловием**

выполнения S (поэтому эта структура иногда называется *циклом с предусловием).*

В ряде так называемых *структурных языков программирования* (таких как Паскаль, Алгол-68, Модула-2), ориентированных на разработку *структурированных программ,* кроме описанных базовых структур, существуют и другие структуры.

*Структура цикла с постусловием* имеет вид «выполнять S пока C». Тело цикла S выполнится по крайней мере один раз, так как условие C проверяется после выполнения S (рис. 4.7).

Отметим, что в ФОКАЛе цикл FOR может быть записан через любую из структур цикла.

Последней структурой, которую мы рассмотрим, будет *структура варианта*. Эта структура имеет вид «вариант J $S_0$, $S_1$, ..., $S_{n-1}$, где J — индекс варианта ($0 \le J \le n - 1$), а SJ — операторы или

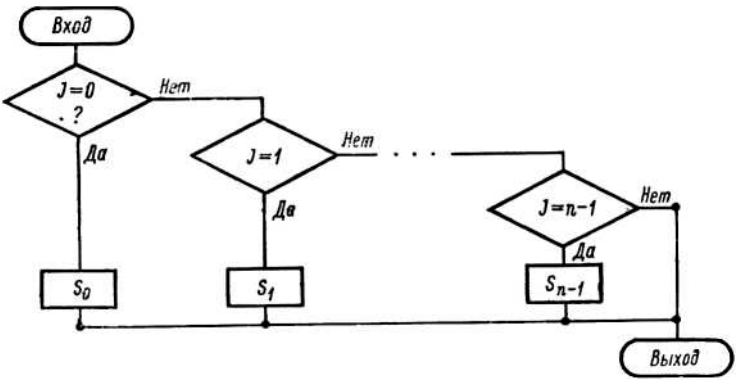



**Рис. 4.8. Блок-схема структуры варианта**

структуры. При работе этой структуры выполнится только одна из структур $S_K$, а именно та, индекс которой равен значению индекса варианта (K = J). Если $J \ge n$, то ни одна из структур $S_K$ не выполнится. Эта структура показана на рис. 4.8.

Проектирование программы с помощью описанных структур делает ее *структурированной.* Правда в большинстве языков программирования (в том числе и в ФОКАЛе) потребуются операторы GOTO, но их использование в программе следует ограничить переходами к определенным точкам — началу и концу задачи или подзадачи (модуля).

К достоинствам структурного программирования можно отнести следующее: 1) последовательность операций проста для

просмотра, что облегчает отладку и проверку программ (модулей); 2) количество структур ограничено, а технология стандартизирована; 3) структуры могут быть легко преобразованы в модули; 4) теоретически доказано, что данный набор структур полон, т. е. все программы могут быть записаны с помощью трех базовых структур; 5) структурированная версия программы оказывается в значительной степени самодокументированной и довольно легко читаемой; 6) структурированные программы легко описать с помощью блок-схемы; 7) структурное программирование увеличивает производительность труда программистов.

Структурное программирование по сравнению с модульным требует от программистов гораздо большей дисциплины и аккуратности. В результате получаются более систематические и лучше организованные программы. Вместе с тем, структурное программирование не лишено недостатков. Перечислим основные из них.

1. Только в нескольких языках высокого уровня (Паскаль, Модула-2, Ада и др.) непосредственно присутствуют структуры. Следовательно, программист, не использующий эти языки, должен затратить дополнительные усилия для преобразования структур (как промежуточной формы описания) в конструкции применяемого им языка. Тем не менее, структурная версия программы часто полезна как документация.

2. Структурированные программы выполняются, как правило, более медленно и используют большую память, чем неструктурированные программы.

3. Ограничение числа структур до трех-четырех базовых конструкций делает некоторые задачи очень трудными для программирования. Полнота набора структур означает, что любая программа может быть реализована с их помощью, хотя не всегда эффективно.

4. Базовые структуры довольно часто запутаны. Например, вложенные структуры «если ... то ... иначе» могут быть очень сложны для восприятия, так как не всегда есть явное указание на то, где заканчивается внутренняя структура. То же относится к вложенным циклам «пока ... выполнять».

5. В структурированных программах рассматриваются только последовательности программных операций, а не потоки данных. Следовательно, управление данными может быть реализовано не самым лучшим образом.

Некоторые программисты привыкают к структурному

программированию, тогда как другие считают стандартные структуры неудобными и ограниченными. Мы не призываем и не отговариваем от использования структурного программирования. Это один из подходов к систематизированной разработке программ. Вообще говоря, структурное программирование наиболее эффективно в следующих ситуациях: 1) большие программы, возможно, превышающие 1000 строк; 2) приложения, в которых размер памяти и быстродействие не являются критичными; 3) приложения, в которых стоимость разработки программного обеспечения (особенно проверка и отладка) является существенным фактором; 4) приложения, включающие обработку символьных строк, управление процессами или другие сложные алгоритмы, а не просто числовые расчеты.

Так как стоимость машинной памяти неуклонно уменьшается, а объем и стоимость разработки программного обеспечения возрастают, то методы, аналогичные структурному программированию, которые уменьшают стоимость разработки больших программ (пусть даже с использованием большей памяти), получают все большее распространение. Безусловно, структурное программирование не панацея, но оно в значительной степени упорядочивает процесс разработки программ, который иначе мог бы быть хаотическим. Структурированная программа служит также средством отладки, проверки и документирования.

Можно рекомендовать следующий подход к разработке структурированных программ: 1) начинайте с основной блок-схемы, которая поможет определить логику программы; 2) старайтесь использовать, в основном, базовые конструкции «следование», «если ... то ... иначе» и «пока ... выполнять» (известно, что они являются полным множеством, т. е. любая программа может быть записана в терминах этих структур); 3) отделяйте каждый новый уровень отступами из нескольких пробелов от предыдущего уровня, с тем чтобы программу было удобнее читать; 4) используйте ограничители для каждой структуры, например, «конец» для «пока ... выполнять» и «все» для «если ... то ... иначе» (ограничители плюс отступы делают программу ясной и понятной); 5) придавайте особое значение простоте и читаемости программы: вставляйте пробелы, используйте значащие идентификаторы и делайте выражения как можно более понятными; не пытайтесь оптимизировать логику программы в ущерб ясности; 6) вставляйте в программу комментарии только там, где это необходимо; 7) проверяйте логику

структур и операторов, проверяйте все граничные варианты или специальные условия и несколько простых вариантов.

*Пример. Структурная версия системы «ключ—индикатор».* Структурная версия этого примера имеет следующий вид:

```
КЛЮЧ := ВЫКЛ
пока КЛЮЧ = ВЫКЛ выполнять
    Читать КЛЮЧ
конец
ИНДИКАТОР := ВКЛ
Задержка 1
ИНДИКАТОР := ВЫКЛ
```

Значения ВКЛ (включить) и ВЫКЛ (выключить) должны быть соответствующим образом определены для ключа и индикатора. Предполагается, что «Задержка» — это модуль, который обеспечивает задержку, задаваемую параметром в секундах.

Оператор в структурированной программе (такой как «Читать») в действительности может быть подпрограммой. Однако, чтобы не нарушать правила структурного программирования, подпрограмма должна возвращать управление главной программе.

Так как конструкция «пока ... выполнять» проверяет условие до выполнения цикла, то переменной КЛЮЧ присваивается значение ВЫКЛ. Структурированная программа последовательна, читабельна и легко проверяется вручную. Однако она, вероятно, займет большую память и будет выполняться дольше, чем неструктурированная программа, в которой будет не нужно инициализировать переменную КЛЮЧ, а процедуры чтения и проверки условия будут объединены.

**Программирование на ФОКАЛе на основе алгоритмического языка.** Хотя концепции структурного программирования обычно связываются с языками высокого уровня, синтаксис которых включает различные типы структур (Паскаль, Модула-2 и т. п.), это не означает, что структурное программирование нельзя применять к программированию на языке ФОКАЛ или других языках, безусловно плохо приспособленных к этому подходу.

В работе [28] предлагается метод программирования на основе алгоритмического языка (Е-языка) с построчным кодированием на ФОКАЛе. В основу метода положены следующие принципы: 1) единство основных компонентов разработки программ (постановки, алгоритмизации, реализации алгоритма и отладки); 2) структурный подход к программированию; 3) базовый язык

(псевдокод).

Идея метода основана на том, что ФОКАЛ допускает запись комментариев в одной строке с выполняемым оператором (правее его) или в отдельной строке. Это позволяет ввести в программу и сохранить в качестве комментариев ее текст на псевдокоде (алгоритмическом языке). Кроме того, поскольку ФОКАЛ допускает однобуквенную запись имен операторов, то это позволяет сократить в тексте программы ту ее часть (коды), которая мешает естественному восприятию логики программы.

Проиллюстрируем этот подход на нескольких простых примерах. Вычисление функции

$$y = \begin{cases} x \text{ при } x < 0 \\ 2x \text{ при } x \geq 0 \end{cases}$$

может быть выполнено с помощью следующей программы:

```
1.10 C                          АЛГ ФУНКЦИЯ
1.20 C                          НАЧ ВЕЩ X,Y
1.30 ;                              ASK "X=",X
1.40 IF (X)1.50,1.80,1.80; C        ЕСЛИ X < 0
1.50 C                                ТО
1.60 S                                  Y=X
1.70 G 1.90; C                        ИНАЧЕ
1.80 S                                  Y=2*X
1.90 C                              ВСЕ
2.10 ;                              TYPE "Y=",Y
2.20 Q                          КОН
```

Обратите внимание что текст программы на псевдокоде вводится в компьютер с отступом от левого края строки, например, с 30-й колонки. Записанные в строках 1.10 и 1.20 заголовки алгоритма и описания переменных предваряются в ФОКАЛе буквой C (COMMENT), т. е. рассматриваются как комментарии. Команды ввода ASK и вывода TYPE, включенные в алгоритмический язык из ФОКАЛа, также как операторы S (SET), предваряются «;» для сохранения отступа от левого края строки. Кроме оператора IF для кодирования структуры условия «если ... то ... иначе» используется оператор GOTO, который кодирует слово ИНАЧЕ для обхода второй альтернативы. Слово КОН кодируется оператором Q (QUIT) — останов программы. Во вспомогательных алгоритмах

(подпрограммах) это слово кодируется оператором R (RETURN), как будет видно из следующего примера, демонстрирующего применение описываемого метода при организации подпрограмм.

Вызов вспомогательного алгоритма в Е-языке соответствует (в терминах языков программирования высокого уровня) обращению к процедуре с передачей параметров по значению. Однако дело в том, что ФОКАЛ не поддерживает вызов процедуры с параметрами. Поэтому при записи подпрограммы на псевдокоде необходимо использовать уникальные имена для параметров и промежуточных переменных. Присваивая аргументам вспомогательного алгоритма значения фактических параметров при входе в него, а при выходе передавая результаты, мы моделируем вызов процедуры с параметрами.

Рассмотрим кодирование вспомогательного алгоритма на примере программы вычисления следующей функции:

$$u\,(x, y) = \max\,(x, y)/\max\,(0{,}5, x - y)$$

```
1.10 C                              АЛГ ДРОБЬ
1.20 C                              НАЧ ВЕЩ X,Y,U,A,B
1.30 ;                                 ASK X,Y
1.40 S M=X;S N=Y;D 2;S A=K;C           MAX(X,Y,A)
1.50 S M=0.5;S N=X-Y;D 2;S B=K;C       MAX(0.5,X-Y,B)
1.60 S                                 U=A/B
1.70 ;                                  TYPE "U=",U
1.80 Q                               КОН
1.90 C-------------------------------------
2.01 C                              АЛГ MAX(ВЕЩ M,N,K)
2.05 C                                 АРГ M,N
2.10 C                                 РЕЗ K
2.15 C                              НАЧ
2.20 IF (M-N)2.25,2.25,2.40;C          ЕСЛИ M-N
2.25 C                                   ТО
2.30 S                                     K=N
2.35 G 2.45;C                            ИНАЧЕ
2.40 S                                     K=M
2.45 C                                 ВСЕ
2.50 R                              КОН
```

В строках 1.40—1.50 содержатся два вызова вспомогательного алгоритма (подпрограммы) MAX с разными значениями фактических параметров. Выход из подпрограммы происходит с помощью оператора R

(RETURN) на операторы, следующие за оператором DO в этих строках. С помощью этих операторов присваивания результат вычислений (значение переменной К) присваивается фактическим параметрам А и В соответственно.

Рассмотренный метод программирования на основе алгоритмического языка крайне полезен как простое и доступное средство обучения основам программирования на микрокомпьютере «Электроника БК-0010». Существенно, что на всех этапах разработки программы программист мыслит в терминах алгоритмического языка, овладевая тем самым структурным программированием — систематическим методом создания правильных программ.

**Проектирование «сверху вниз».** Мы пока не рассматривали вопрос о том, как независимо проверять модули (подзадачи) и объединять их вместе. Ясно, что необходим метод, который допускает проверку и отладку в реальной программной среде и обеспечивает модульный подход к разработке программных систем.

Таким методом является проектирование «сверху вниз» или, как его иногда называют, нисходящее проектирование. При этом подходе начинают с записи общей управляющей программы. Те ее части (подзадачи), которые пока недостаточно определены или реализацию которых приходится на время отложить, заменяются программными «заглушками», т. е. временными программами, которые либо имитируют (с той или иной степенью приближения) работу соответствующей подзадачи, либо просто пусты. После этого можно проверить логику управляющей программы, чтобы убедиться в ее правильности.

На следующем шаге происходит расширение заглушек. Как правило, каждая заглушка, в свою очередь, будет содержать подзадачи, которые временно будут заменяться заглушками. Этот процесс расширения, отладки и проверки продолжается до тех пор, пока все заглушки не заменятся работающими программами.

Отметим, что проверка и включение отлаженных подзадач происходит на каждом уровне, а не только в конце. При этом отпадает необходимость в разработке вспомогательных программ для генерации данных.

Проектирование сверху вниз предполагает модульное программирование и совместимо также со структурным программированием.

Как и всякий метод, проектирование сверху вниз не лишено

недостатков, а именно: 1) общий подход не учитывает конкретные особенности аппаратуры; 2) заглушки могут оказаться трудными для разработки (в частности, если они должны правильно функционировать в различных местах программы); 3) ошибки, допущенные на верхнем уровне, могут оказать катастрофическое влияние на всю последующую разработку.

Проектирование сверху вниз — полезное средство, но не панацея. Оно обеспечивает систематический метод для расширения блок-схемы или постановки задачи до уровня, необходимого для реального написания программы. Вместе со структурным программированием оно образует полное множество средств разработки.

Можно рекомендовать следующий подход к проектированию сверху вниз: 1) начинайте с основной блок-схемы;

2)делайте заглушки настолько полными и независимыми, насколько это возможно; 3) аккуратно определяйте все возможные выходы из каждой заглушки и подготавливайте соответствующий набор тестов; 4) проверяйте каждый уровень тщательно и систематически; 5) используйте структуры из структурного программирования; 6) внимательно следите за общей задачей и структурами данных; 7) выполняйте отладку и проверку после каждого расширения заглушек и не пытайтесь сделать слишком много за один шаг.

Альтернативой методу проектирования сверху вниз служит *проектирование снизу вверх* (*восходящее проектирование*). В то время, как при проектировании сверху вниз, исходная задача разбивается на подзадачи, при проектировании снизу вверх сначала проектируются модули нижнего уровня. Основной недостаток восходящего проектирования связан с тем, что интеграция модулей одного уровня с помощью модуля более высокого уровня может быть чрезвычайно сложна, поскольку спроектированные раздельно нижние модули могут иметь несогласованные интерфейсы.

В действительности при разработке программ проектирование сверху вниз и снизу вверх взаимно дополняют друг друга. Например, с помощью проектирования снизу вверх разрабатываются процедуры общего назначения — подпрограммы ввода—вывода, управления структурами данных и т. п. Дальнейшее проектирование ведется нисходящим методом.

*Пример. Нисходящее проектирование системы «ключ—индикатор».* Пример структурного программирования этой системы полностью совпадает с первым шагом разработки сверху вниз (см. структурную версию

системы). В действительности используемые операторы есть ни что иное, как заглушки, так как ни один из них полностью не определен. Например, что означает оператор Читать КЛЮЧ? Если КЛЮЧ является одним битом порта ввода ПОРТ, то этот оператор, в действительности, принимает вид КЛЮЧ: = ПОРТ /\ МАСКА, где /\ — операция «логическое И», а константа МАСКА имеет бит, равный единице в соответствующей позиции.

Аналогично «Задержка 1» означает в действительности (если процессор сам обеспечивает задержку)

```
РЕГ := СЧЕТЧИК
пока РЕГ # 0 выполнять
    РЕГ := РЕГ - 1
конец
```

Здесь СЧЕТЧИК — соответствующее число для обеспечения односекундной задержки. Теперь расширенная версия программы принимает вид

```
КЛЮЧ := 0
пока КЛЮЧ = 0 выполнять
    КЛЮЧ := ПОРТ /\ МАСКА
конец
ИНДИКАТОР := ВКЛ
РЕГ := СЧЕТЧИК
пока СЧЕТЧИК # 0 выполнять
    РЕГ := РЕГ - 1
конец
ИНДИКАТОР := ВЫКЛ
```

Безусловно, это программа более ясная и теперь ее легче записать фактическими инструкциями или операторами.

**Заключение.** Обратите внимание, что на протяжении всего раздела практически не упоминалось ни о конкретном компьютере, ни о конкретном языке программирования. Несмотря на распространенное заблуждение, что написание машинных кодов или операторов языка программирования является ключевой частью разработки программного обеспечения, это, в действительности, оказывается наиболее простым этапом.

После написания нескольких программ этап кодирования становится для программиста достаточно простым, так как он запоминает инструкции или операторы и приобретает необходимый навык. В то же время сложность других стадий

проектирования программного обеспечения не уменьшается.

На этапе постановки задачи необходимо определить все характеристики системы: ввод—вывод, обработку, ограничения по памяти и времени, обработку ошибок и тому подобное.

На стадии разработки программ существенную помощь оказывают методы программирования. *Модульное программирование* побуждает программиста разбивать всю программу на небольшие отдельные модули. *Структурное программирование* обеспечивает систематический подход к определению логики этих модулей, в то время как *проектирование сверху вниз* служит систематическим методом их объединения и проверки. Все эти методы обеспечивают унифицированный подход к разработке программного обеспечения и их следует рассматривать как основу, на базе которой должна вестись его разработка.

**Стиль программирования.** Многие программисты считают, что не имеет никакого значения, как написана программа, достаточно, чтобы она работала согласно заданным спецификациям. Это ошибочное мнение, особенно когда речь идет о тех программах, которые имеют большой жизненный цикл. Такая программа, как правило, используется и поддерживается не ее разработчиком, а другими пользователями, которые должны иметь возможность вносить в нее изменения и исправлять выявленные в процессе эксплуатации ошибки. Отсюда возникает еще одна важная характеристика программы — ее *читаемость* или наглядность.

Хорошо структурированная программа во всех случаях более понятна, чем неструктурное нагромождение операторов. Но кроме структурного подхода к разработке программ, у программиста есть и дополнительные средства, владение которыми и создает *стиль программирования*.

Перечислим основные элементы стиля программирования.

1. Выбор имен переменных, массивов и других программных объектов, которые обозначаются идентификаторами. Желательно, чтобы эти имена имели смысловое значение. Хотя в ФОКАЛе идентификаторы различаются по первым двум символам, интерпретатор ФОКАЛа допускает использование идентификаторов произвольной длины, что повышает читаемость программы и облегчает ее понимание. При этом нужно следить, чтобы у двух разных идентификаторов не совпадали первые два символа.

2. Расстановка комментариев в соответствующих местах

программы. Хорошо документированная программа включает комментарии, расположенные только там, где это необходимо, бессмысленно комментировать очевидные действия, например:

```
1.10 SET K=K+1; COMMENT K УВЕЛИЧИВАЕТСЯ НА 1
```

Хотя расстановка комментариев во многом зависит от индивидуальности программиста, существуют бесспорные ситуации, когда это желательно, например:

1) чтобы выделить начало и конец некоторой части программы, выполняющей важные функции:

```
COMMENT НАЧАЛО ФОРМИРОВАНИЯ МАССИВОВ A,B
...
COMMENT КОНЕЦ ФОРМИРОВАНИЯ A,B
```

2) чтобы отметить начало и конец подпрограммы с объяснением ее функций;

3) чтобы при вызове подпрограммы объяснить ее действия:

```
DO 5;COMMENT ПОДПРОГРАММА ВЫЧИСЛЕНИЯ СРЕДНЕГО
```

4) чтобы пояснить функции некоторой части программы, которые не очевидны или используют специальные средства.

3. Структурирование (форматирование) текста программы. Расположение программных структур друг относительно друга, показывающее их вложенность и подчиненность, делает программу более ясной и удобной для отладки. Использование этого подхода, к сожалению, существенно зависит от языка программирования. Для некоторых из них он, если можно так выразиться, естественен (Паскаль, Модула-2 и др.), тогда как в ФОКАЛе его можно использовать только для рукописного оформления программы.

В заключение отметим, что некоторые особенности ФОКАЛа (как интерпретатора) не благоприятствуют хорошему стилю программирования. Например, использование длинных имен и комментариев увеличивают как объем памяти, занимаемый программой, так и время ее выполнения. Поэтому некоторые программисты разрабатывают и сохраняют два варианта

программы. Первый, снабженный комментариями, используется для документирования и сопровождения программы; второй вариант максимально сокращен (исключены комментарии, идентификаторы сокращены до одного-двух символов, в каждой строке записано максимально возможное количество операторов языка) для увеличения быстродействия и минимизации занимаемой памяти.

## 4.3. ОТЛАДКА И ТЕСТИРОВАНИЕ ПРОГРАММ

В идеальном случае работа программиста завершается составлением программы, поскольку выполнение этой программы на компьютере должно привести к требуемым результатам. Фактически первоначально написанная программа, как правило, содержит ошибки.

Стремление уменьшить количество ошибок при разработке программ было одной из главных причин, которые привели к созданию современной методологии программирования. Систематическое нахождение и исправление ошибок еще на этапе проектирования особенно эффективно, поскольку не расходуется машинное время. Не выявленные на этом этапе ошибки могут быть обнаружены и устранены при выполнении программы.

Интерпретатор ФОКАЛа удобное и мощное средство для отладки программ. Недостатки ФОКАЛа как языка проектирования (по сравнению, скажем, с Паскалем или Модулой-2) компенсируются удобством быстрой и эффективной отладки программ.

**Отладка программы.** Отладкой обычно называют процесс устранения ошибок, допущенных при разработке программы. Есть только один способ выяснить правильность программы — исполнить программу с подходящими для проверки наборами данных, т. е. выполнить тестирование программы. Следовательно тестирование является неотъемлемой частью отладки программы. Если программа работает неверно, то требуется локализировать ошибку (ошибки), установить ее характер и исправить ее. После этого необходимо снова тестировать программу. Этот процесс продолжается до тех пор, пока не будут устранены все ошибки и тестирование не выполнится успешно.

Отладка большой программы целиком чрезвычайно трудная задача. Она занимает много времени, а результат часто оказывается плачевным. Очевидно, что гораздо легче отладить небольшую

программу (20—30 операторов). Поэтому известную стратегию «разделяй и властвуй» необходимо применять и при отладке программ.

Вспомним, что программисты могут использовать предварительно написанные и отлаженные модули (наиболее часто модули оформляют как подпрограммы).

В отличии от проектирования при отладке имеет смысл использовать восходящий подход, при котором прежде всего отлаживаются подпрограммы самого низкого уровня, а в последнюю очередь — главная программа. Далеко не всегда каждый модуль нужно оформлять как подпрограмму. Если данная подпрограмма вызывается в одном месте, то после отладки имеет смысл включить ее тело в это место программы, что сэкономит и память и время, поскольку будут исключены операторы DO и RETURN.

**Тестирование программы.** Тестирование — это выполнение программы со специально подобранными наборами данных (тестовые наборы, тесты) в целях выяснения наличия ошибок. Как правило, одного тестового набора недостаточно для выявления всех возможных ошибок. Тестовые наборы должны учитывать как объем и структуру входных данных, так и организацию и структуру самой программы.

Составление тестов для конкретной программы совсем не простая задача. Одна из причин, по которой данный тест не выявляет все ошибки, кроется в разветвленной структуре алгоритма. При тестировании программы с некоторым тестовым набором выполняется только часть операторов, реализующих соответствующую ветвь алгоритма. Для проверки всех ветвей алгоритма тесты необходимо составлять так, чтобы выполнились все ветвления (например, в операторе IF) программы. Программа со сложной логикой обычно требует большого количества тестов.

Отладка и тестирование охватывают большой круг проблем, наиболее важными из которых являются следующие: 1) как составлять тестовые наборы данных; 2) каким образом по неверному выполнению теста определять место и вид ошибки (ошибок), если интерпретатор не выдавал конкретного сообщения о ней; 3) каким образом наиболее рационально исправить ошибку; 4) каким должен быть тест, который покажет, что ошибка устранена?

Не существует формальной теории, которая давала бы ответ на все эти (и многие другие) вопросы. Стратегия отладки во многом

зависит от используемого языка программирования и от конкретной задачи. Существующие интерпретаторы ФОКАЛа имеют средства, которые поддерживают работу по отладке программ.

**Виды тестов и их составление.** Любой набор данных может быть тестовым набором для программы или просто тестом. С точки зрения программы входные наборы могут быть разделены на два класса.

1. Допустимые входные наборы. Это такие наборы данных, значения которых находятся в определенных пределах и подчиняются определенным ограничениям. Программа принимает эти данные и на их основе создает некоторый другой набор данных (входной), если работает правильно.

2. Недопустимые входные наборы. Такие данные имеют значение или структуру, которая не подчиняется заданным ограничениям. При работе с такими данными программа не может сформировать «осмысленные» результаты. Поэтому правильно написанная программа должна анализировать и распознавать недопустимые входные наборы и либо сообщать об этом, либо игнорировать их. Правильность работы программы в таких ситуациях закладывается еще при постановке задачи.

На начальном этапе тестирование программы должно производиться только допустимыми наборами данных, так как использование недопустимых усложнит и без того непростой процесс локализации ошибок. Но после того, как программа отлажена на допустимых наборах данных, желательно проверить ее «реакцию» на недопустимые наборы. Для программ вычислительного характера наибольший интерес и трудность представляет проверка *граничных условий*, когда значения данных лежат на границах допустимых интервалов.

**Составление теста.** Формирование определенного входного набора еще не означает, что тест подготовлен. Необходимо определить ожидаемую правильную реакцию программы на данный набор, вплоть до проверки работы программы с этим набором «вручную». Например, некоторые задачи вычислительного характера можно проверить с помощью электронного калькулятора. Ясно, что составление тестов может занять много времени, но на этом нельзя экономить.

**Размеры теста.** Некоторые программы могут работать с различными по величине наборами данных. Например, в программе умножения матриц размеры матриц могут изменяться в

широких границах.

Было бы неверно начинать отладку программы с теста, который имеет большие размеры. В лучшем случае составление такого теста и определение правильности работы программы займет много времени, в худшем — может привести к ошибкам, аналогичным тем, которые типичны для граничных условий.

**Диагностические возможности теста.** В простейшем случае тест может зафиксировать наличие ошибки. Если же тест составлен таким образом, что позволяет определить место и характер ошибки, то такой тест называют диагностическим.

**Нахождение ошибок.** Следует свыкнуться с мыслью, что ошибки при программировании — неизбежное зло. Любой программист допускает ошибки, но опытный программист умеет их находить, тогда как неопытный зачастую оказывается перед ними беспомощным.

*Виды ошибок*. Ошибки могут быть очень разнообразны и классифицироваться по различным признакам. Например, по этапам проектирования, на которых они допускаются, выделяют следующие ошибки: в постановке задачи, в алгоритме, в реализации алгоритма в программе, синтаксические, семантические и др.

Понятие *синтаксическая ошибка* означает несоблюдение синтаксических правил при записи операторов языка. Например, синтаксическими будут ошибки: 1.10 SET X = = A*—B (нельзя записывать два знака операции подряд); 1.50 SET X = (A + ( — D) (несоответствие числа открывающих и закрывающих скобок).

Как правило, все синтаксические ошибки выявляются интерпретатором ФОКАЛа, поэтому они наименее опасны.

По мере повышения квалификации программиста количество таких ошибок уменьшается.

Под *семантическими ошибками* понимают неправильное толкование (программистом) смысла оператора. Например, если строки 1.60 нет, а программист передал на нее управление (GOTO 1.60), то это будет семантической ошибкой, поскольку синтаксис самого оператора правильный. Следует иметь в виду и учитывать при отладке, что интерпретатор выявляет далеко не все семантические ошибки.

Ошибки могут разделяться на *логические* и *механические.* Логические ошибки возникают при проектировании программы. Механические допускаются, как правило, по невнимательности при записи программы.

Некоторые логические ошибки обнаруживаются интерпретатором, когда они создают ненормальную ситуацию типа деления на нуль, переполнения, извлечения квадратного корня из отрицательного числа и т. п. Но в общем случае такие ошибки не выявляются интерпретатором, который не может знать, что используется одна переменная вместо другой, натуральный логарифм вместо десятичного и т. д. Наиболее трудно находить и исправлять ошибки, допущенные на раннем этапе проектирования (анализ, составление алгоритма). В отдельных случаях исправление таких ошибок приводит к разработке новой программы.

*Стратегия поиска ошибок*. Если «поведение» программы отличается от ожидаемого при составлении теста, а ошибка не очевидна, то необходимо выбрать какую-то стратегию поиска ошибок. Она зависит от поведения программы и состоит в следующем: 1) в зависимости от поведения программы выдвигается гипотеза о причине ошибки; 2) проверяют эту гипотезу; если она оказывается неверной, то нужно придумать новую и т. д., до тех пор, пока не будет установлена истинная причина.

Следовательно, для отладки необходимо располагать: средствами для анализа поведения программы; знаниями о наиболее вероятных ошибках при различных видах поведения; способами определения ошибок по поведению.

Интерпретатор ФОКАЛа имеет набор стандартных средств анализа поведения. К ним, например, относятся: пуск программы с начала (GO) или с произвольной строки (например, GO 2.05); трассировка (прослеживание) программы; расстановка контрольных остановов (QUIT) в определенных точках; возможность вывода промежуточных значений переменных.

*Трассировка*. Это наиболее удобное средство получения диагностической информации о ходе выполнения программы без изменения ее логики или структуры. Режим трассировки устанавливается в программе на ФОКАЛе с помощью символа «?», который можно помещать в любое место программы. При выполнении программы текст строки (или строк), заключенный между двумя символами «?» или между символами «?» и ВК в последней строке программы, выводится на печать.

Оператор печати TYPE и функция FCHR выполняются обычным образом. Например, при следующей организации программы:

```
1.10 S A=1; S B=5; S C=3
1.20 T %1,?A+B-C?,!
1.30 T ?B+C/A,!
1.40 F I=1,3; T I+A,!
```

на печать будет выводиться такая информация

```
*G
A+B-C    3
B+C/A    8!
F I=1,3; T I+A,   2!
T I+A,    3!
T I+A,    4!
*
```

Здесь вывод символьной строки A + B — C обусловлен двумя символами «?» в строке 1.20, а затем оператор TYPE выводит значение этого выражения; после этого выводится информация между символом «?» в строке 1.30 и концом последней строки, причем та часть строки 1.40, которая выполняется в цикле, распечатывается при каждой итерации.

При запуске программы с помощью оператора GO? выполняется трассировка всей программы, как если бы символ «?» был первым символом первой строки программы. Например,

```
1.10 S X=1
1.30 T %1,X,!
*GO?
1.10 S X=1
1.30 T %1,X,    1!
*
```

Трассировка является незаменимым средством анализа в следующих случаях: программа зациклена; программа завершилась преждевременно, не напечатав результаты совсем или частично; не выполнилась некоторая подпрограмма или не произошло нормальное возвращение к главной программе и т. п

**Контрольная печать.** Трассировка не показывает всего, что происходит при выполнении программы, а именно — значений

переменных. Если в какой-то части программы необходимо знать эти значения, то следует поместить оператор печати в это место.

Например, если с помощью трассировки было выяснено, что оператор

```
1.65 IF (A)2.20,2.30,2.40
```

не передал управление на требуемую строку, то полезно непосредственно перед строкой 1.65 поместить контрольную печать

```
1.60 TYPE "A=",A,!
```

Она позволит узнать значение переменной А, от которой зависит переход.

*Контрольные остановы*. Чтобы при тестировании программы исключить ошибочные переходы на те ее ветви, которые не должны проверяться при данном тесте, можно использовать оператор останова QUIT, «запирающий» эти ветви программы. Чтобы программист знал, в каком месте программы произошел останов, перед оператором QUIT следует поместить оператор TYPE *nn.mm,* где *nn. mm* — номер соответствующей строки.

Другой способ организации контрольного останова — использование, например, оператора GO с несуществующим номером строки. В случае выполнения этого оператора интерпретатор ФОКАЛа сообщит об ошибке и выведет номер строки, в которой он находится (выполнение программы при этом, естественно, прекратится).

*Наиболее часто встречающиеся ошибки*. Программирование предоставляет «бездну» возможностей для того, чтобы допустить ошибку, и невозможно описать все виды ошибок и их влияние на работу программы. Более того, ошибка одного вида может по-разному проявляться не только в различных программах, но и в разных местах одной программы. Тем не менее, из опыта известно, что существуют типы ошибок, которые встречаются наиболее часто.

Рассмотрим некоторые из этих типов ошибок и обсудим их влияние на работу программы.

1. Не выполнена инициализация переменных. В большинстве реализаций интерпретатор ФОКАЛа считает, что переменная, впервые встретившаяся в правой части оператора присваивания или в качестве аргумента функции, имеет нулевое значение. Если значение этой переменной должно было быть отличным от нуля, то

это отразится на работе программы.

Чтобы избежать ошибок такого рода, рекомендуется всегда устанавливать начальное значение переменной (даже если оно равно нулю) с помощью оператора присваивания SET.

2. Побочное влияние на значение переменной. Программисты часто используют одну и ту же переменную для нескольких различных целей. «Любимыми» переменными являются A, B, C, I, J, K и др. В результате значение переменной может измениться, когда это не ожидается. Например,

```
1.65 SET A=5
1.70 DO 3
1.75 SET B=A+3
3.05 SET A=0
3.10 FOR I=1,15; S A=A+I
3.20 S A=A/15
```

Программист считает, что в строке 1.75 переменная A имеет значение 5, а оно изменилось потому что эта переменная использовалась и в подпрограмме (группе строк) 3. В подпрограмме надо было использовать другую переменную, например, A1.

3. Ошибки в циклах. Такие ошибки наиболее часто допускаются в циклах, организованных без использования оператора FOR. Из-за них возникает одно из следующих состояний программы: зацикливание (бесконечное выполнение цикла); цикл выполняется один раз или вообще не выполняется, а должен был выполниться многократно; вместо K раз цикл выполняется K— 1 или K + 1 раз.

Причина этих ошибок чаще всего кроется в операторе IF, проверяющем условие выхода из цикла.

В цикле, организованном с помощью оператора FOR, наиболее типичной ошибкой является модификация переменной цикла в теле цикла; в результате изменяется число повторений, неправильно выбираются индексированные переменные.

4. Ошибки арифметического характера. Здесь прежде всего следует упомянуть такие ошибки, как переполнение, деление на нуль, извлечение квадратного корня из отрицательного числа, потерю значимости и т. п. Как правило, эти ошибки обнаруживаются интерпретатором и не вызывают проблем.

Большую трудность представляют ошибки, которые возникают из-за неточного выполнения операций с вещественными числами,

что типично для компьютеров (отметим, что ФОКАЛ оперирует только с вещественными числами).

Ошибки из-за неточности возникают тогда, когда программист думает, что результат точный, хотя, в сущности, это не так. Например, может случиться так, что произведение $10^{-6} \cdot 10^{6}$ (записанное в виде 1E—6*1E6) не будет в точности равно единице из-за погрешности при преобразовании чисел из внешнего десятичного во внутреннее двоичное представление. По той же причине цикл FOR K = 1(E — 04, 1E — 04,1; ... будет выполнен не 10 000 раз, а 9998 раз, так что последнее значение К не будет в точности равно единице.

Если в той же программе используется условие вида IF (К — 1) 1.20, 1.40, 1.20, то понятно, что управление на строку 1.40 никогда не будет передаваться и программа будет работать неправильно.

Это типичный пример того, как небольшие погрешности приводят к существенным ошибкам. Опытный программист будет проверять выполнение такого условия с учетом возможного отклонения абсолютной величины К от единицы.

5. Ошибки в операторах условия. Тут часто допускаются такие банальные ошибки, как: условие, которое проверяется в операторе IF, сформулировано неправильно; управление передается не туда, куда нужно (неверен номер строки), и т. п.

6. Ошибки при исправлении программы. Нередко при исправлении одной ошибки в программу вносятся новые. Одна из наиболее распространенных ситуаций, когда при изменении имени переменной новое имя вносится не везде, где используется эта переменная. Во избежание этой, да и других подобных ситуаций, программисту следует быть внимательным при внесении исправлений в программу (а не только при исправлении ошибок), особенно если в изменяемой (исправляемой) части программы используются операторы GOTO.

## 4.4. ЭФФЕКТИВНОСТЬ ПРОГРАММ

Программа считается эффективной, когда она занимает *минимальную память* и выполняется за *минимальное время*. Стремление к эффективности оправдано, когда программа большая и с трудом помещается в память микрокомпьютера.

Быстродействие, например, может оказаться ограничивающим фактором при управлении процессами, в какой-нибудь компьютерной игре и вообще при работе в реальном времени.

Увеличение быстродействия такой программы означает уменьшение времени реакции на входные сигналы.

Средства создания быстрой и компактной программы на ФОКАЛе в известном смысле вступают в противоречие с хорошим стилем программирования. Эффективные программы трудно понимать и модифицировать. Для достижения эффективности программы можно рекомендовать следующие приемы: 1) используйте имена переменных, состоящие из одного символа; 2) не включайте в программу комментарии; 3) заменяйте полное имя оператора его первой буквой; 4) используйте переменные вместо констант, а значения этим переменным присваивайте в начале программы; 5) помещайте в строку несколько операторов, разделенных точками с запятой.

Необходимо понимать, что все эти приемы являются следствием специфической реализации ФОКАЛа как интерпретатора

Усилия, которые необходимо затратить, чтобы написать короткую и быстродействующую программу, не всегда оправданы. Например, в каких случаях стремиться к эффективности не следует: когда программа имеет короткий жизненный цикл; когда не возникает ограничений по памяти и программа выполняется за «приемлемое» время; когда усилия по созданию эффективной программы чрезмерны.

Вывод из всего сказанного состоит в том, что стремиться к эффективности программы надо только тогда, когда это оправдано.

ГЛАВА 5

# НАУЧНЫЕ, ОБУЧАЮЩИЕ И ИГРОВЫЕ ПРОГРАММЫ НА ФОКАЛе

## 5.1. ПРОГРАММЫ ДЛЯ НАУЧНЫХ РАСЧЕТОВ

**Определение направления обхода многоугольника.** Составная часть геометрических задач автоматизации проектирования больших интегральных схем — процедура идентификации направления обхода многоугольников.

Приведенная программа (пример 1) реализует простой алгоритм определения направления обхода многоугольника, заданного последовательным перечислением координат (X (I), Y (I), I = 1, K) его угловых точек.

**Вычисление параметров окружности.** Приведенная программа (пример 2) вычисляет координаты центра и радиус окружности, проходящей через заданные три точки (координаты вводимых точек X1, Y1; X2, Y2; X3, Y3 могут быть в общем случае и отрицательными, поэтому квадрат значения переменной вычисляется в программе как X1 * X1, а не X1 ^ 2).

**Быстрая сортировка.** Программа (пример 3) реализует алгоритм QUICKSORT — быстрой сортировки данных

**ПРИМЕР 1**

```
C: ФОКАЛ-БК0010
1.01 C ПРОГРАММА ОПРЕДЕЛЕНИЯ НАПРАВЛЕНИЯ
1.05 C ОБХОДА ПРОИЗВОЛЬНОГО МНОГОУГОЛЬНИКА
1.10 A "ВВЕДИТЕ КОЛИЧЕСТВО УГЛОВ
       МНОГОУГОЛЬНИКА =",N
1.15 T "ВВЕДИТЕ КООРДИНАТЫ УГЛОВЫХ ТОЧЕК
       МНОГОУГОЛЬНИКА",!
1.20 F I=1,N;T "X(",%1,I,")=";A x(I);
     T "Y(",I,")=";A Y(I)
1.25 S XMIN=1.E+30;F I=1,N;D 2.0
1.30 I (IX-1)1.40,1.35,1.40
1.35 S I2=N;S I1=2;G 1.55
1.40 I (IX-N)1.50,1.45,1.50
1.45 S I2=N-1;S I1=1;G 1.55
1.50 S I1=IX+1;S I2=IX-1
```

```
1.55 S DX=X(I1)-X(IX);S DY=Y(I1)-Y(IX);
     S SP=DY/FSQT(DX*DX+DY*DY)
1.60 S DX=X(I2)-X(IX);S DY=Y(I2)-Y(IX);
     S SN=DY/FSQT(DX*DX+DY*DY)
1.65 I (SN-SP)1.70,1.70;T "ОБХОД ПРОТИВ
       ЧАСОВОЙ СТРЕЛКИ";G 1.75
1.70 T "ОБХОД ПО ЧАСОВОЙ СТРЕЛКЕ"
1.75 Q
2.01 I [X(I)-XMIN]2.05;R
2.05 S XMIN=X(I);S IX=I
```

в возрастающем порядке. В качестве входных данных в программе используется линейный массив A длины N случайных чисел с равномерным законом распределения на промежутке.

**Численные методы.** Программы вычисления факториала, вычисления интеграла от функции на заданном интервале по модифицированному методу Симпсона и вычисления значения корня заданной функции по методу Ньютона (примеры 4—6) демонстрируют возможность использования ФОКАЛа для достаточно сложных инженерных расчетов.

**ПРИМЕР 2**

```
C: ФОКАЛ-БК0010
  1.05 T "ВВЕДИТЕ КООРДИНАТУ ТРЕХ ТОЧЕК";
       A "X1=",X1, "Y1=", Y1,! "X2=",X2,"Y2=",Y2,
       !"X3=",X3,"Y3=",Y3,!
  1.10 S D=X1*(Y2-Y3)+X2*(Y3-Y1)+X3*(Y1-Y2)
  1.15 S OX=[(X1*X1+Y1*Y1)*(Y2-Y3)+(X2*X2+
       Y2*Y2)*(Y3-Y1)+(X3*X3+Y3*Y3)*(Y1-Y2)]/
       2/D
  1.20 S OY=[(X1*X1+Y1*Y1)*(X3-X2)+(X2*X2+Y2*
       Y2)*(X1-X3)+(X3*X3+Y3*Y3)*(X2-X1)]/2/D
  1.30 S R=FSQR[(OX-X1)*(OX-X1)+(OY-Y1)*(OY-Y1)]
  1.40 T "РАДИУС КРУГА=",R,
       "КООРДИНАТЫ ЦЕНТРА КРУГА ","OX=",OX,
       "OY=",OY
*
```

**ПРИМЕР 3**

```
C: ФОКАЛ-БК0010
  1.01 A "N=",N
  1.05 F I=1,N;S A(I)=N*(FRAN()+1)
  2.05 S K=1;S S(1,1)=1;S S(1,2)=N
  2.20 S L=S(K,1);S M=S(K,2);S K=K-1
  2.35 S I=L;S J=M
  2.45 S R=FITR((L+M)/2);S X=A(R)
  2.50 I (X-A(I))2.65,2.65;S I=I+1;G 2.50
  2.65 I (A(J)-X)2.75,2.75;S J=J-1;G 2.65
  2.75 I (J-I)2.90
  2.78 S R=A(I);S A(I)=A(J);S A(J)=R;S I=I+1;
       S J=J-1
  2.85 I (I-J)2.50,2.50
  2.90 I (M-I)2.94,2.94;S K=K+1;S S(K,1)=I;
       S S(K,2)=M
  2.94 S M=J;I (L-M)2.35;I (K)2.20,2.98,2.20
  2.98 F I=1,N;DO 3.0
  2.99 Q
  3.01 T A(I);R
```

**ПРИМЕР 4**

```
C: ФОКАЛ-БК0010
  1.01 X FCHR(12);T !!"    ВЫЧИСЛЕНИЕ
       ФАКТОРИАЛА"
  1.10 A !,"N=",N;S ST=0;S IN=-1
  1.15 D 5
  1.20 I <IN>1.25;T !,%6.0,N,"! =",%,B,%6.0;
       D 2;G  1.1
  1.25 T !,%6.0,N,"! =",B
  1.30 G 1.1
  2.20 F I=1,3;X FCHR(8)
  2.30 X FCHR(23);T "("; F I=1,5;X FCHR(25)
```

```
  2.40 T "+",ST,")"
  5.10 S B=1
  5.20 F I=1,N;D 6;S B=B*I
  5.22 R
  6.10 I <1.E+30-B>6.2;R
  6.20 S IN=0;S B=B/1.E+30;S ST=ST+30;R
*

ПРИМЕР 5

C: ФОКАЛ-БК0010
  1.10 C ПРОГРАММА ВЫЧ. ИНТЕГРАЛА ПО СИМПСОНУ
       С ПОПРАВКОЙ ПО РУНГЕ
  1.02 C АВТОР: ЭЙГИН ФЕДОР, ТЕЛ. 183-07-97
  1.50 X FCHR(12,155);T !,!,!,"   ВЫ УЖЕ ВВЕЛИ
       ФУНКЦИЮ?";I (FCHR(-1)-68)1.6,1.8,1.6
  1.60 X FCHR(12);T !,!,!,"   ОРГАНИЗУЙТЕ ВАШУ
       ФУНКЦИЮ   ",!
  1.62 T "   В ОПЕРАТОРАХ ГРУППЫ 9   ",!
  1.64 T " ПРИМЕР:",!,!," 9.01 S &=FCOS(&+&^5);
       R",!;X FCHR(155);Q
  1.80 X FCHR(12);T !,!,"   ВВЕДИТЕ:",!,!,
       "   НИЖНИЙ ПРЕДЕЛ?";A A
  1.82 T !,"   ВЕРХНИЙ ПРЕДЕЛ?";A B;T !,
       "   ЧИСЛО РАЗБИЕНИЙ (ЧЕТНОЕ)?"; A N
  1.90 D 4;S Y1=Y;S N=N*2;D 4;S Y2=Y;
       S ER=(Y2-Y1)/15;S Y=Y2+ER
  1.95 T !,!,%,"   ИНТЕГРАЛ=",Y,!,"   ОШИБКА=",
       ER,!,!,"   IN=",Y1,!,"   I2N=",Y2,!
  1.97 X FCHR(155);Q
  4.02 S H=(B-A)/N;S Y=0;S X=A
  4.05 F I=2,2,N;S Y=Y+FSBR(9,X)+4*FSBR(9,X+H)+
       FSBR(9,X+2*H);S X=X+2*H
  4.06 S Y=Y*H/3;R
  9.01 S &=FCOS(&+&^5);R
*
```

**ПРИМЕР 6**

```
C: ФОКАЛ-БК0010
  1.01 C ПРОГР. МЕТОД НЬЮТОНА. АВТОР ЭЙГИН ФЕДОР.
       ТЕЛ. 183-07-97
  1.03 C РЕШЕНИЕ УР-ИЯ F(X)=0
  1.50 X FCHR(12,155);T !,!,!,"   ОРГАНИЗУЙТЕ
       ВАШУ ФУНКЦИЮ   ",!
  1.52 T " И ЕЕ ПРОИЗВОДНУЮ В ГРУППАХ 8,9.", !,!,!
  1.54 T "   ОРГАНИЗОВАЛИ?";I (FCHR(-1)-68)1.99,
  1.6,1.99
  1.60 X FCHR(12);T !,!," ВВЕДИТЕ:",!,!;
       A " ПРИБЛИЖЕНИЕ КОРНЯ",X0;T !
  1.61 A " КОЛ-ВО ДОПУСТИМ. ИТЕРАЦИЙ",N;T !
  1.62 A " ТОЧНОСТЬ",EPS;T !;D 4
  1.99 Q
  4.02 S D0=1E12;S XS=X0;S I=1
  4.03 S V=FSBR(9,XS);S XN=XS-FSBR(8,XS)/V;
       S DEL=FABS(XN-XS)
  4.04 I (DEL-EPS)4.07;I (DEL-D0)4.05,4.05;
       T !,"ИТЕРАЦИИ НЕ СХОДЯТСЯ К КОРНЮ!";Q
  4.05 I (N-I)4.06;S I=I+1;S XS=XN; S D0=DEL;
       G 4.03
  4.06 T !,"ЧИСЛО ИТЕРАЦИЙ ВЫШЕ ДОПУСТИМОГО!";Q
  4.07 T !,!,!,"   КОРЕНЬ X=",%8.04,XN,!,!,
       %4.00,I," ИТЕРАЦИИ.";X FCHR(155);T !;Q
  8.01 S &=10*&-FEXP(&);R
  9.01 S &=10-FEXP(&);R
*
```

## 5.2. КОМПЬЮТЕРНАЯ ГРАФИКА

Приводимые в этом разделе программы демонстрируют возможности как цветной, так и монохромной поточечной графики на БК (примеры 7—9). Среди них программы, реализующие быстродействующие алгоритмы (модификация метода Безенхема)

качественного формирования базовых графических растровых примитивов: линии (пример 10) и окружности (пример 11), программа для создания в диалоговом режиме простейших графических рисунков на экране устройства отображения БК (пример 12), демонстрационная программа формирования десяти окружностей, аппроксимированных отрезками прямых линий, положение и размер каждой из которых на экране устройства отображения БК — случайные величины с равномерным законом распределения (пример 13). Для устранения ошибок при выполнении программа содержит проверку условий высвечивания точки за пределами экрана:

$$0 \le X < XM,\ 0 \le Y \le YM, \qquad (5.1)$$

где XM — максимально допустимое значение горизонтальной координаты; YM — максимальное допустимое значение вертикальной координаты.

В программе показано использование неравенства (5.1) для экрана размером 512X240 точек. В случае нарушения неравенства (5.1) по соответствующей координате X или Y значению координаты высвечиваемой точки присваивается одно из предельно допустимых значений изменения координаты.

**ПРИМЕР 7**

```
C: ФОКАЛ-БК0010

1.02 C
1.03 C G 1.1 - ЦИКЛИЧЕСКИЙ ВЫЗОВ
1.04 C
1.05 C ПРОГРАММА ДЕМОНСТРАЦИИ ЦВЕТНОЙ ГРАФИКИ
1.06 C
1.07 C Е.А. РУДОМЕТОВ    Г.ЛЕНИНГРАД 25.07.86
1.08 C
1.09 E;G 1.11
1.10 E;S СК=10
1.11 X FCHR(12,148,153,154)
1.12 X FCHR(145);T "   ПРОГРАММА 'ГРАФИКА'"!!
1.14 X FCHR(146);T !!!"НЕКОТОРЫЕ ВОЗМОЖНОСТИ"
1.16 T !"МИКРО-ЭВМ";X FCHR(145);
```

```
     T "'ЭЛЕКТРОНИКА БК0010'"
1.17 X FCHR(147);T !!!!"ЦВЕТНАЯ ГРАФИКА";
     X FCHR(145)
1.18 F I=1,4000;
1.19 X FCHR(12);D 5;D 6;D 7;X FCHR(155)
1.20 X FCHR(12);X FT(0,127,115);F I=1,220;D 20
1.30 X FCHR(155)
1.32 X FCHR(146);S X=127;X FCHR(12);D 15
1.34 X FCHR(12,145);D 17;F I=1,1000;
1.36 X FCHR(157);F I=1,1000;
1.38 X FCHR(157)
1.40 X FCHR(12);F I=1,150;D 10
1.50 X FCHR(12);F I=1,400;X FT(1,128,115);D 10
1.60 X FCHR(12);X FT(1,127,115);F I=1,300;D 60
1.70 X FCHR(12);D 70
1.75 C
1.80 X FCHR(154);I <CK-5>1.9;G 1.1
1.90 X FCHR(12,148,158,145)
1.92 T !!!!,"ВВОД СЛЕДУЮЩЕЙ ПРОГРАММЫ С МЛ";R
2.05 C
5.08 X FCHR(158)
5.10 F I=0,16,255;X FT(1,I,0);X FV(1,I,255)
5.12 X FT(1,255,0);X FV(1,255,239)
5.20 F I=0,16,239;X FT(1,0,I);X FV(1,255,I)
5.25 X FT(1,0,239);X FV(1,255,239)
5.27 F I=112,1,144;X FT(1,I,112);X FV(1,I,128)
5.30 X FCHR(148,158)
6.10 D 8;X FCHR(146);D 5
7.10 D 8;X FCHR(147);D 5;D 8
8.10 F I=1,2000;
10.10 S X=FITR(128*(FRAN()+1)+.5);
      S Y=FITR(115*(FRAN()+1)+.5)
10.20 S S=FRAN()+1
10.30 I <S-.5>10.5;I <S-1>10.6;I <S-1.5>10.7;
10.40 X FCHR(145);G 10.8
10.50 X FCHR(146);G 10.8
10.60 X FCHR(147);G 10.8
10.70 X FCHR(148)
10.80 X FV(1,X,Y)
15.20 F I=-127,2,0;S S=X+I;S R=X-I;D 16.1
15.25 F I=1,2000;
```

```
15.30 X FCHR(12)
15.40 F I=0,5,127;S S=X+I;S R=X-I;D 16.1
15.50 X FCHR(12)
15.60 F I=-125,2,0;S S=X+I;S R=X-I;D 16.2;
      D 16.3
15.65 D 15.25
16.10 X FT(1,R,R);X FV(1,S,R);X FV(1,S,S);
      X FV(1,R,S);X FV(1,R,R)
16.20 X FT(1,R,R);X FCHR(145);X FV(1,S,R);
      X FCHR(146);X FV(1,S,S)
16.30 X FCHR(147);X FM(1,R,S);X FCHR(146);
      X FV(1,R,R)
17.10 F K=1,3;D 18
18.10 F I=1,16;D 19
18.20 F I=1,16;D 19.2;D 19.1
19.10 F J=1,4;X FCHR(127)
19.20 F J=1,4;X FCHR(32)
20.20 X FV(1,128*FITR(FRAN()+1.5)+127,
            115*FITR(FRAN()+1.5))
22.20 X FV(1,127*FITR(FRAN()+1.5)+254,
            115*FITR(FRAN()+1.5))
60.20 S S=FRAN()+1;I <S-.66>60.4;
      I <S-1.32>60.5;
60.30 X FCHR(145);G 60.8
60.40 X FCHR(146);G 60.8
60.50 X FCHR(147)
60.80 X FV(1,127*FITR(FRAN()+1.5),
            115*FITR(FRAN()+1.5))
70.10 S L=120;S X=150;S Y=125
70.17 S K0=0.75
70.18 F C=1,2;D 70.24;F T=1,200;S TR=X*Y
70.19 X FCHR(145);R
70.24 X FCHR(12);F XR=-L,5,L;D 80
70.30 X FCHR(145)
70.40 R
80.10 X FT(0,X*K0,Y)
80.15 X FCHR(145)
80.20 X FV(1,(X+XR)*K0,Y+FSQT(L^2-XR^2))
80.22 I (C-2)80.25
80.23 X FT(0,X*K0,Y)
80.25 X FCHR(147)
```

```
  80.30 X FV(1,(X+XR)*K0,Y-FSQT(L^2-XR^2))
*
```

ПРИМЕР 8

```
C: ФОКАЛ-БК0010
  1.10 X FCHR(12,148,158,155,154,145)
  1.12 X FCHR(155);T "    ПРОГРАММА 'НЕБО'"!!
  1.14 X FCHR(146);T !!!"НЕКОТОРЫЕ ВОЗМОЖНОСТИ"
  1.16 T !"МИКРО-ЭВМ";X FCHR(145);
       T "'ЭЛЕКТРОНИКА БК0010'"
  1.17 X FCHR(147);T !!!!"ЦВЕТНАЯ ГРАФИКА";
       F I=1,4000;
  1.20 X FCHR(12);S X=128;S Y=127;S L=2
  10.20 F I=1,5;S R(I)=FRAN()
  10.30 S S=R(1);I <S+.05>10.5;I <S>10.6;
        I <S-.5>10.7
  10.40 X FCHR(145);G 10.8
  10.50 X FCHR(146);G 10.8
  10.60 X FCHR(147);G 10.8
  10.70 X FCHR(148)
  10.80 X FT(1,X*R(2)/(R(4)+L)+X,
          Y*R(3)/(R(5)+L)+Y);G 10.2
*
```

ПРИМЕР 9

```
C: ФОКАЛ-БК0010
  8.03 C
  8.05 C ЧАСЫ С ТАЙМЕРОМ
  8.06 C
  8.07 C Е.А. РУДОМЕТОВ  Г.ЛЕНИНГРАД  12.08.86
  8.08 C
  8.09 C
  8.10 X FCHR(12,155,148,158,145);X FP(2,177777)
  8.20 S PI=3.1415926;S RR=50;S RC=30
  8.22 A "ЧАСТОТА ЗАДАЮЩЕГО ГЕНЕРАТОРА",!,TG
  8.25 T !!!!,"ИДЕТ ПОДГОТОВКА ДАННЫХ"
  8.30 F I=0,59;S AR=PI*(1-I/15)/2;S MX(I)=125+
         FCOS(AR)*RR;S MY=125-FSIN(AR)*RR
```

```
8.40 F I=0,47;S AR=PI*(1-I/6)/2;S CX(I)=125+
       FCOS(AR)*RC;S CY(I)=125-FSIN(AR)*RC
9.10 X FCHR(12);T "УСТАНОВКА ЧАСОВ"
10.01 A !,"ЧАСЫ  ",TC,"МИНУТЫ  ",TM,"СЕКУНДЫ",
        TS;S CC=TC*2;S PC=CC
10.02 T !,"УСТАНОВКА БУДИЛЬНИКА"
10.03 A !,"ЧАСЫ  ",CB,"МИНУТЫ  ",MB,"СЕКУНДЫ",
        SB;X FCHR(12,154,155)
10.04 X FK(0,21);T %2.00"БУДИЛЬНИК"!,CB,
        "  ЧАСОВ",MB,"  МИНУТ",SB,"  СЕКУНД"
10.05 S T=FCLK();S T=T-TS*TG;X FCHR(155);D 60
10.06 F I=0,2,22;X FT(1,CX(I)*2-125,
        CY(I)*2-125)
10.08 S NN=0;D 20.2
10.10 S TS=FITR(<FCLK()-T>/TG+0.5)
10.11 I <TS-60>10.12;S NM=0;D 40.2;S TM=TM+1;
      S T=FCLK();S TS=TS-60
10.12 I <TM-60>10.15;I <TC-23>10.13;S TC=-1
10.13 S NC=0;D 30.2;S TM=0;S TC=TC+1;S CC=TC*2;
      S PC=CC
10.15 X FK(0,0);T %2.00,TC,"  ЧАСОВ",!,TM,
        "  МИНУТ",TS,"  СЕКУНД"
10.16 X FCHR(147);D 20
10.17 X FCHR(145);D 40
10.18 I <TM-30>10.19;I <TS>10.2;I <2-TS>10.19;
      S NC=0;D 30.2;S PC=CC+1
10.19 D 30
10.20 I <TC-CB>10.8,10.22,10.8
10.22 I <TM-MB>10.8,10.24.10.8
10.24 I <TS-SB>10.8,10.30,10.3
10.30 F I=1,3;D 10.4;F J=1,5;D 10.5;
       F K=1,50;X FCHR(7)
10.40 F C=1,300;
10.50 F C=1,50»
10.80 G 10.08
20.10 S NN=1
20.20 X FT(1,125,125);X FV(NN,MX(TS),MY(TS))
30.10 S NC=1
30.20 X FT(1,125,125);X FV(NC,CX(PC),CY(PC))
40.10 S NM=1
```

```
  40.20 X FT(1,125,125);X FV(NM,MX(TM),MY(TM))
  60.20 X FK(15,5);T "12"
  60.30 X FK(6,12);T "9";X FK(24,12);T "3"
  60.40 X FK(15,19);T ”6"
*
```

**ПРИМЕР 10**

```
C: ФОКАЛ-БК0010
  1.01 X FCHR(12);X FT(1,0,239);X FV(1,200,239);
       X FV(1,200,96);X FV(1,0,96)
  1.05 X FV(1,0.239)
  2.05 A X0,Y0,IX,IY
  2.10 S N1=1;S N2=1;S LF=0;S X=0;S Y=0
  2.15 I (-IX)2.20,2.20;S N1=-1
  2.20 I (-IY)2.25,2.25;S N2=-1
  2.25 S IX=FABS(IX);S IY=FABS(IY);
       I (IX-IY)2.30,2.35,2.35
  2.30 S LF=1;S R=IX;S IX=IY;S IY=R
  2.35 S D=FITR(IY-IX/2);F I=1,IX;D 3.05
  2.40 Q
  3.05 I (D)3.10,3.10;S X=X+N1;S Y=Y+N2;
       S D=D+IY-IX;G 3.25
  3.10 I (LF)3.15,3.15;S Y=Y+N2;G 3.20
  3.15 S X=X+N1
  3.20 S D=D+IY
  3.25 S Y1=Y/1.4;X FT(1,X0+X,239-Y0-Y1)
*
```

**ПРИМЕР 11**

```
C: ФОКАЛ-БК0010
  1.03 X FCHR(12)
  1.05 A X0,Y0,R
  1.10 S X=R;S Y=0;S D=5/4-R
  1.15 S I=X;S J=Y
  1.20 F II=1,2;D 2.0
  1.25 I (FABS(X)-FABS(Y))1.05,1.05,1.30
  1.30 I (D)1.35,1.40,1.40
  1.35 S Y=Y+1;S D=D+2*Y+1;G 1.15
  1.40 S X=X-1;S Y=Y+1;S D=D-2*X+2*Y+1;G 1.15
```

```
  2.01 S J1=FITR(J/1.7);X FT(1,X0+I,Y0+J1);
       X FT(1,X0-I,Y0+J1)
  2.05 X FT(1,X0-I,Y0-J1);X FT(1,X0+I,Y0-J1)
  2.10 S I=J;S J=X;R
*
```

**ПРИМЕР 12**

**С: ФОКАЛ-БК0010**

```
  1.01 X FCHR(12);T "    **РИСУНОК НА ЭКРАНЕ**"
  1.02 T !,"ДИРЕКТИВЫ:          -ПЕРЕМЕЩАЕТ
           ТОЧКУ ПО СТРЕЛКЕ"
  1.04 T !,"T-ФИКСИРУЕТ ТОЧКУ"
  1.05 T !,"L-ЧЕРТИТ ЛИНИЮ ОТ T ДО НОВОЙ ТОЧКИ"
  1.06 T !,"C-СТИРАЕТ ТОЧКУ ИЛИ ЛИНИЮ"
  1.07 T !,"ДЛЯ АВТ. ПЕРЕМЕЩЕНИЯ ТОЧКИ-НАЖАТЬ
           ПОВТ"
  1.08 T !,"(ДЛЯ ПРОДОЛЖЕНИЯ-НАЖАТЬ ЛЮБУЮ
           КЛАВИШУ)"
  1.09 X FCHR(-1)
  1.10 X FCHR(12)
  1.20 X FK(0,0);T "X=";X FK(10,0);T "Y="
  1.30 S N=1;S K=0;S X=256;S Y=128
  1.40 X FT(N,X,Y)
  1.50 X FK(2,0);T %3.00,X;X FK(12,0);T %3.00,Y
  1.60 S B=FCHR(-1)
  1.70 D B;G 1.40
  67.10 X FT[0,X1,Y1];X FV[0,X,Y];S N=1;G 1.60
  76.10 X FT[1,X1,Y1];X FV[1,X,Y]
  77.10 S N=K;D 1.40;S Y=Y+1;S N=1;S K=0
  81.10 S N=K;D 1.40;S X=X-1;S N=1;S K=0
  83.10 S N=K;D 1.40;S Y=Y-1;S N=1;S K=0
  84.10 S X1=X;S Y1=Y;S K=1;G 1.60
  94.10 S N=K;D 1.40;S X=X+1;S N=1;S K=0
*
```

**ПРИМЕР 13**

**С: ФОКАЛ-БК0010**

```
  1.10 S XM=511;S YM=239
  1.15 F I=1,10;D 1.20
```

```
  1.17 Q
  1.20 X FCHR(12);G 1.25
  1.25 F J=1,10;D 2
  1.30 F K=0,100;R
  2.01 S R=10+FITR(40*(FRAN()+1)/2)
  2.03 S X=FITR(XM*(FRAN()+1)/2)
  2.05 S Y=FITR(YM*(FRAN()+1)/2)
  2.10 S DT=2*3.14/R;S TH=0
  2.15 S X1=X+FITR(R*FCOS(TH))
  2.20 S Y1=Y+FITR(R*FSIN(TH))
  2.30 I (X1-XM)2.35,2.35:S X1=XM
  2.35 I (-X1)2.40,2.40;S X1=0
  2.40 I (Y1-YM)2.45,2.45;S Y1=YM
  2.45 I (-Y1)2.50,2.50;S Y1=0
  2.50 X FT(1,X1,Y1)
  2.55 F K=0,R;D 3
  2.60 R
  3.01 S TH=TH+DT
  3.05 S X2=X+FITR(R*FCOS(TH));
       S Y2=Y+FITR(R*FSIN(TH))
  3.10 I (-X2)3.15;S X2=0;G 3.20
  3.15 I (X2-XM)3.20,3.20;S X2=XM
  3.20 I (-Y2)3.25;S Y2=0;G 3.30
  3.25 I (Y2-YM)3.30,3.30;S Y2=YM
  3.30 X FV(1,X2,Y2)
*
```

## 5.3. ИГРОВЫЕ И ОБУЧАЮЩИЕ ПРОГРАММЫ

Приведенные учебные программы обучают некоторым приемам работы с БК в режимах редактирования и управления поточечной графикой (пример 14) и работе с клавиатурой БК (пример 15). Отметим, что для функционирования последней требуется подключение к БК генератора импульсов. Кроме того, в раздел включена небольшая игровая программа «Питон» (пример 16). В этой программе змея «Питон» может перемещаться по экрану в горизонтальном и вертикальном направлении. Движением змеи можно управлять с клавиатуры. Цель игры заключается в поедании денег (целей). Каждый раз после съедания монетки змея становится длиннее. Если голова змеи выходит за границу экрана или пересекает собственное тело, то она погибает, а игра заканчивается.

**ПРИМЕР 14**

```
C: ФОКАЛ-БК0010
  1.03 T !,"ЭТА ПРОГРАММА ДЕМОНСТРИРУЕТ
       НЕКОТОРЫЕ НЕ"
  1.04 T !,"ЛЕЖАЩИЕ НА ПОВЕРХНОСТИ ВОЗМОЖНОСТИ
       ФОКАЛА"
  1.05 T !!,"РАССМОТРИМ ТАКУЮ ЗАПИСЬ СУММЫ
       ПО J:"
  1.08 T !!,"              "
  1.10 T "Σ                "
  1.20 T " J "
  1.30 C
  1.40 T "A(I,J)*X(J)",!!
  1.50 T !,"УПРАЖНЕНИЕ: НАЙДИТЕ НА КЛАВИАТУРЕ
       БУКВУ";D 1.10
  1.51 T ".";T !!;D 12
  1.55 T !
  1.60 T !,"ЕЕ К СОЖАЛЕНИЮ, НЕТ. В ЭТОЙ СИТУАЦИИ"
  1.61 T !,"УДОБНО ИСПОЛЬЗОВАТЬ ГРАФИЧЕСКИЙ
       РЕЖИМ."
  1.62 T !,"ЧТОБЫ ЗАПИСАТЬ ОПЕРАТОР";T !!;
       W 1.10
  1.65 T !,"НАДО НАЖАТЬ КЛАВИШИ:"
  1.70 T !,"Т ПРОБЕЛ КАВЫЧКА РЕД ГРАФ →→→"
```

```
  1.80 T !,"→→→ ↓ЗАП↑ ←←←←←← "
  1.90 X FCHR(132,32,30,32,30,320,30,32,31,32,
       31,32,31,132)
  2.01 T !,"↓→→→→→→↑↑ГРАФ → РЕД КАВЫЧКА"
  2.10 T !!;D 12
  2.15 T !,"ОПЕРАТОР В СТРОКЕ 1.20 СОДЕРЖИТ
ПОСЛЕДОВАТЕЛЬНОСТЬ"
  2.20 T !,"КАВЫЧКА РЕД ";X FCHR(132,31,132)
  2.22 T "J ↑ РЕД КАВЫЧКА, ПРИЧЕМ"
  2.25 T !,"J РАСПОЛОЖЕН В ТЕКСТЕ НА СЛЕДУЮЩЕЙ
       СТРОЧКЕ:"
  2.30 D 11
  2.35 T !,"В ОПЕРАТОРЕ",!!;W 1.90
  2.40 T !,"ДЛЯ РИСОВАНИЯ КОСЫХ СТРЕЛОЧЕК
ИСПОЛЬЗУЕТСЯ"
  2.45 T !,"КЛАВИША  БЛОК РЕД  (КОД132)"
  2.47 T !!;D 12
  3.10 T !,"ВОТ ЕЩЕ ОДНО ПОЛЕЗНОЕ ПРИМЕНЕНИЕ
       КЛАВИШИ"
  3.20 T !,"БЛОК РЕД. КОГДА МЫ РАСПЕЧАТЫВАЕМ
       ПРОГРАММУ"
  3.30 T !,"ОПЕРАТОР W, В СТРОКАХ 1.10,1.20
       МЫ ВИДИМ"
  3.40 T !,"ОКОНЧАТЕЛЬНЫЙ РЕЗУЛЬТАТ ДЕЙСТВИЯ
       ПОСЛЕДОВАТЕЛЬ-"
  3.50 T !,"НОСТИ УПРАВЛЯЮЩИХ КЛАВИШ, А САМА
       ЭТА"
  3.60 T !,"ПОСЛЕДОВАТЕЛЬНОСТЬ ОТ НАС СКРЫТА:";
       D 11
  3.70 T !,"ЕСЛИ ЖЕ ПЕРЕД ОПЕРАТОРОМ W НАЖАТЬ
       БЛОК РЕД"
  3.80 T !,"ТО В ЭТИХ СТРОКАХ МЫ УВИДИМ ИСКОМУЮ"
  3.90 T !,"ПОСЛЕДОВАТЕЛЬНОСТЬ:"
  3.95 X FCHR(132);D 11;X FCHR(132);T !;D 12
  4.10 T !,"ВАЖНО, ЧТО ИСПОЛЬЗОВАНИЕ  БЛОК РЕД
       ПОЗВОЛЯЕТ"
  4.20 T !,"РЕДАКТИРОВАТЬ СТРОКИ ТИПА 1.10, Т.К."
  4.30 T !,"ВСЕ УПРАВЛЯЮЩИЕ СИМВОЛЫ ВИДНЫ"
  4.35 T !,"И ИХ ЛЕГКО МЕНЯТЬ"
  4.55 T !!;D 12;D 9;T !!;D 12;
  4.70 T !,"ДАЛЬНЕЙШЕГО ПРОГРЕССА МОЖНО ДОСТИЧЬ,
```

```
      ЕСЛИ"
4.80 T !,"ГАСИТЬ КУРСОР НА ВРЕМЯ РИСОВАНИЯ
      ГРАФИ-"
4.90 T !,"ЧЕСКИХ СИМВОЛОВ, ИСПОЛЬЗУЯ КЛАВИШУ"
5.10 T !,"КУРСОР  (В НИЖНЕМ РЕГИСТРЕ).
      ЕЕ КОД - 154"
5.20 T !,"СРАВНИТЕ:"
5.30 T !!;W 9.3;D 9.30;T !!;W 5.60
5.60 X FCHR(154);D 9.3;X FCHR(154)
5.70 T !!;D 12
5.80 T !,"К СОЖАЛЕНИЮ, В РЕЖИМЕ ГРАФ"
5.85 T !,"НАМ НЕ УДАСТСЯ ПОМЕСТИТЬ СИМВОЛ"
6.10 T !,"ГАШЕНИЯ КУРСОРА ПРЯМО В ОПЕРАТОР
      TYPE"
6.20 T !,"ОДНАКО МЫ ЕЩЕ НЕ ИСЧЕРПАЛИ ВСЕХ
      ВОЗМОЖ-"
6.30 T !,"НОСТЕЙ РИСОВАНИЯ"
6.40 D 1.10;T "!";T !!;D 12
6.50 T !,"УСОВЕРШЕНСТВОВАТЬ"
6.60 T !!;X FCHR(132);W 1.10;X FCHR(132)
6.70 T !,"МОЖНО, ЕСЛИ ВОСПОЛЬЗОВАТЬСЯ ОДНИМ
      ОСОБЫМ"
6.80 T !,"СВОЙСТВОМ ГРАФИЧЕСКОГО РЕЖИМА:
      ВМЕСТО →→→→"
6.90 T !,"МОЖНО НАПИСАТЬ ПРОСТО  4→.
      ОПЕРАТОР"
7.10 T !!;X FCHR(132);W 1.10;X FCHR(132)
7.20 T !,"МОЖНО, СЛЕДОВАТЕЛЬНО, ЗАПИСАТЬ
      В ВИДЕ"
7.30 T !!;X FCHR(132);W 13.10;X FCHR(132)
7.40 T !;D 12
7.45 T !,"ОПЕРАТОР ПОЛУЧАЕТСЯ КОМПАКТНЕЕ,
      ХОТЯ"
7.50 T !,"СКОРОСТЬ ЗАМЕТНО НЕ УВЕЛИЧИВАЕТСЯ:"
7.60 T !!,"1.10   "
7.75 X FCHR(154)
7.80 F I=1,20;D 1.10
7.85 T !!,"13.10   "
7.90 F I=1,20;D 13.10
7.95 X FCHR(154)
8.05 T !!;D 12
```

```
  8.10 T !,"НА ЭТОМ КОНЕЦ"
  8.80 T !!;Q
  9.05 T !
  9.10 T !,"ГРУППА ОПЕРАТОРОВ"
  9.12 T !!;W 10
  9.15 T !,"ПО СВОЕМУ ДЕЙСТВИЮ ЭКВИВАЛЕНТНА"
  9.20 T !,"ОПЕРАТОРУ";T !;W 1.10
  9.25 T !,"ОДНАКО 1.10 РАБОТАЕТ БЫСТРЕЕ,
         ЗАНИМАЕТ МЕНЬШЕ"
  9.27 T !,"МЕСТА В ПАМЯТИ И БОЛЕЕ НАГЛЯДЕН.",
         !;D 12
  9.28 T !!;W 9.30
  9.30 T !;F I=1,20;D 1.10
  9.35 T !!;W 9.40
  9.40 T !;F I=1,20;D 10
  10.10 X FCHR(149,25,25,25,25,25,25,27,150)
  10.20 X FCHR(26,8,8,8,8,8,8,27,30,30,30,31,
         31,31,27)
  10.30 X FCHR(25,25,25,25,25,25,26,26,150,
         149,25)
  11.10 T !!;W 1.10;W 1.20;W 1.30;W 1.40
  12.10 T "        НАЖМИТЕ ВВОД   ";X FCHR(-1)
  12.20 X FCHR(24,24,24,24,24,24,24,24,24,24,
         24,24)
  13.10 T "Σ "
*
```

ПРИМЕР 15

```
C:  ФОКАЛ-БК0010
  2.15 C ПРОГРАММА ПОМОГАЕТ ИЗУЧИТЬ КЛАВИАТУРУ.
  2.30 C ВЕДЕТСЯ АНАЛИЗ ВРЕМЕНИ ОТВЕТОВ,
  2.35 C ПРЕДЛАГАЕТСЯ СИМВОЛ, НА КОТОРЫЙ
  2.37 C ПОТРАЧЕНО БОЛЬШЕ ВРЕМЕНИ.
  2.45 C ИМЕЕТСЯ ДВА СПЕЦИАЛЬНЫХ ОТВЕТА:
  2.50 C   ВС - ПЕЧАТЬ ПОЛНОГО ЧИСЛА ПРИМЕРОВ И
  2.52 C        ОШИБОК
  2.55 C   ГТ - ПЕЧАТЬ МАССИВОВ:
  2.65 C        1.ВРЕМЕНИ ОТВЕТОВ НА СИМВОЛЫ,
  2.70 C        2.ИЗУЧАЕМЫХ СИМВОЛОВ.
  2.75 C
```

```
2.80 C РУДОМЕТОВ Е.А. 6.11.1986 Г.ЛЕНИНГРАД
2.94 C
4.10 X FCHR(155,154,148,158,12);E
5.10 X FCHR(147);T !"ЭТА ПРОГРАММА ПОМОГАЕТ
       ВЫУЧИТЬ
5.11 X FCHR(145);T "КЛАВИАТУРУ";X FCHR(147)
5.12 T !!"КЛАВИШИ ЗАДАЮТСЯ ВЫБОРОМ РЕЖИМА:"
5.13 T !"С-ЦИФРЫ"!"L-ЛАТИНСКИЕ БУКВЫ"!
       "R-РУССКИЕ БУКВЫ"
5.14 T !!"ПРОГРАММА АНАЛИЗИРУЕТ ВРЕМЕНА
       ОТВЕТОВ И ПРЕДЛАГАЕТ СИМВОЛ,"
5.16 T " НА ВЫБОР КОТОРОГО ЗАТРАЧЕНО
       НАИБОЛЬШЕЕ ВРЕМЯ"
5.20 T !!"ПРОГРАММА РАБОТАЕТ С ТАЙМЕРОМ,  "
5.22 T "ДИСКРЕТНОСТЬ КОТОРОГО ЗАДАЕТСЯ
       ВНЕШНИМ  ГЕНЕРАТОРОМ"
6.20 X FCHR(145,154);A !!!"ВВЕДИТЕ ЧАСТОТУ"!
      "ВНЕШНЕГО ГЕНЕРАТОРА",TG,!
6.30 X FCHR(154,147);X FK(1,10);F K=1,3;D 80;
     F I=1,5;X FCHR(19)
8.20 T !"ПРОГРАММА ИМЕЕТ 2 СЛУЖЕБНЫХ РЕЖИМА"
8.30 T !"ОТВЕТ ВС-ИНДИКАЦИЯ КОЛИЧЕСТВ
       ""ПРИМЕРОВ И ОШИБОК"
8.40 T !"ОТВЕТ ГТ-ВЫВОД МАССИВОВ: ВРЕМЕНИ
       ОТВЕТОВ И СИМВОЛОВ"
8.45 F K=1,3500;
8.50 X FK(1,10);F K=1,7;X FCHR(19)
40.10 X FP(2,177777);X FCHR(154,145)
40.20 A !"РЕЖИМ",R
40.22 I <R-0C>40.2,40.24;I <R-0L>40.2,40.26;
      I <R-0R>40.2,40.28,40.2
40.24 S L1=48;S M1=57;G 40.3
40.26 S L1=65;S M1=90;G 40.3
40.28 S L1=224;S M1=254;T !,"НАЖМИТЕ 'РУС'"
40.30 S DB=2*TG/(M1-L1+1)
40.40 X FCHR(154)
50.20 F I=L1,M1;S A(I)=I;
      S TS(I)=10*(FRAN()+2)*TG;S TH(I)=0
50.30 C
60.20 S MX=TS(L1);F I=L1,M1;D 74
60.30 S I=IA;D 70
```

```
60.40 F I=L1,M1;S TS(I)=TS(I)+DB
60.50 G 60.2
60.80 C
70.25 X FCHR(148,158,145,12);S TN=FCLK()
70.30 X FK(10,7);X FCHR(153,145);
      X FCHR(A(I));T "  "
70.35 X FCHR(154);S CM=FCHR(FCHR(-1));
      X FCHR(154)
70.40 I <CM-19>70.44,70.8;
70.42 I <CM-20>70.44,70.9,70.44
70.44 X FK(0,17);X FCHR(153)
70.45 X FK(8,16);X FCHR(153);
      I <CM-A(I)>70.5,70.7,70.5
70.48 C
70.50 X FCHR(146);T "НЕТ,  "
70.52 X FCHR(145);X FCHR(CM)
70.54 X FCHR(146);T "  ОШИБКА  ";S H=1;G 70.3
70.56 C
70.70 S TH(I)=FCLK()-TN;S TS(I)=(TH(I)+TS(I))/2
70.72 C
70.75 X FCHR(147);T "ВЕРНО, МОЛОДЕЦ!";S W=W+1;
      S O=O+H;S H=0
70.78 R
70.80 X FCHR(155);X FK(0,0)
70.82 T "ВСЕГО  ",W,!
70.84 T "ОШИБКИ ",O
70.86 X FCHR(155);F P=1,1000;
70.88 S TN=FCLK();G 70.3
70.90 D 99;X FCHR(-1,12);G 70.25
70.95 R
74.30 I <TS(I)-MX>74.4;S MX=TS(I);S IA=I
74.40 R
80.10 F J=1,600;
80.20 C
99.07 X FCHR(12);S VV=0;S VV1=0
99.08 F I9=L1,1,M1;D 108
99.09 S VV=VV/VV1
99.10 T !!"ВРЕМЯ",%3.01,VV;
      X FK(5,2);T "  ",!
99.15 F I9=L1,5,M1;F I5=0,4;D 106;D 100
99.18 X FK(0,13)
```

```
  99.20 T "СИМВОЛЫ ";X FCHR(155)
  99.25 T "/ПОРЯДОК СООТВЕТСТВУЕТ КОИ-7/"!;
        X FCHR(155)
  99.30 F I9=L1,5,M1;F I5=0,4;D 103;
        X FCHR(A(I5+I9));D 100
  99.40 R
  100.1 I <I5+I9-M1>100.2;S I5=100
  100.2 R
  101.1 X FK(7*I5-1,FITR((I9-L1)/5+4))
  103.1 X FK(7*I5+2,FITR((I9-L1)/5+15))
  106.1 I <I5>106.2,106.2;D 101;
        T %2.01,TH(I5+I9)/TG;R
  106.2 D 106.3;T %2.01,TH(I5+I9)/TG;D 106.3;
        X FCHR(22)
  106.3 X FK(0,FITR((I9-L1)/5+4))
  108.1 C
  108.4 I <TH(I9)-.001>108.8,108.8;
        S VV=VV+(TH(I9)+.0001)/TG;S VV1=VV1+1
  108.8 R
*
```

ПРИМЕР 16

```
C: ФОКАЛ-БК0010
  1.01 C ПРОГРАММА PITON2
  1.02 E;X FCHR(12,155,148,155,146);
       A "УРОВЕНЬ СЛОЖНОСТИ",K
  1.05 X FCHR(12,154);X FK(13,1);T "ПИТОН";
       X FCHR(155,145)
  1.06 X FT(1,2,28);X FV(1,506,28);
       X FV(1,506,230);X FV(1,2,230);
       X FV(1,2,28)
  1.07 S P=1;F I=1,60;S X=FITR<FABS[FRAN()]*60>+
       3;D 1.08
  1.08 S Y=FITR<FABS[FRAN()]*20>+3;S T(X,Y)=1;
       X FK(X,Y);T "■"
  1.10 S DX=1;S DY=0;S N=0
  1.11 F I=1,K;S X=FITR<FABS[FRAN()]*60>+3;
       S Y=FITR<FABS[FRAN()]*20>+3;D 1.12
  1.12 S T(X,Y)=2;X FK(X,Y);T "M"
  1.15 S X=32;S Y=12
```

```
  1.20 S K=FITR[FX(1,104)/256];X FK[X(S),Y(S)];
       T " ";I (K-K1)1.3,3.1,1.3
  1.30 I (K-8)3.1,1.5;I (K-18)3.1,1.6;
       I (K-26)1.7,1.8,3.1
  1.50 S DX=-1;S DY=0;G 3.1
  1.60 S DX=0;S DY=1;G 3.1
  1.70 S DX=1;S DY=0;G 3.1
  1.80 S DX=0;S DY=-1;G 3.1
  3.10 S X=X+DX;I [31-FABS(X-31.5)]4.1
  3.30 S Y=Y+DY;I [10-FABS(Y-12.5)]4.1
  3.35 S [T(X,Y)-1]3.5,3.4,4.1
  3.40 S P=P+1
  3.50 X FK(X,Y);T "O"
  3.60 S K1=K;S J=J+1;S X(J)=X;S Y(J)=Y;S S=J-P;
       S T(X,Y)=2;S T[X(S),Y(S)]=0;G 1.2
  4.10 X=FCHR(12,155);X FK(5,7);
       T "ЭТО ВЫ ЧТО-ТО СЪЕЛИ"
  4.20 X FK(5,8);T "ВАША ДЛИНА - ",%2,P;
       X FCHR(155,154)
*
```

ГЛАВА 6

# АРХИТЕКТУРА И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ БЫТОВОГО ПЕРСОНАЛЬНОГО КОМПЬЮТЕРА «ЭЛЕКТРОНИКА БК-0010»

## 6.1. ОБЩАЯ ХАРАКТЕРИСТИКА БК

«Электроника БК-0010» (в дальнейшем просто БК) — первый в СССР серийно выпускаемый бытовой персональный компьютер. БК предназначен для: организации гибких обучающих и информационно-справочных систем; обучения основам информатики и вычислительной техники; повышения эффективности учебного процесса в школах, ПТУ и ВУЗах; разработки и использования программ развлекательного игрового характера; управления бытовыми приборами и устройствами; включения в состав технологического и измерительного оборудования в качестве управляющей микроЭВМ; создания диагностических и прогнозирующих систем в здравоохранении и т. п.

**Основные технические характеристики.** В настоящее время БК выпускается в трех модификациях: «Электроника БК-0010Ш», «Электроника БК-0010» и «Электроника БК-0010-01».

«Электроника БК-0010Ш» (школьный) отличается наличием интерфейса для радиального подключения устройств с последовательной передачей информации (так называемый ИРПС или телеграфный канал).

«Электроника БК-0010» и «Электроника БК-0010-01» отличаются внешним видом клавиатуры, количеством клавиш, а также тем, что в состав БК-0010-01 входит блок тестовой диагностики МСТД. Кроме того, БК-0010 работает с интерпретатором языка ФОКАЛ, а БК-0010-01 — с интерпретатором языка БЕЙСИК.

Питание компьютера осуществляется напряжением 5 ± 0,15 В при токе 1,5 А через выносной стабилизированный блок питания от однофазной сети переменного тока напряжением 220 В и частотой 50 Гц. Блок питания входит в комплект поставки БК. К БК подключаются периферийные устройства: отображения информации и внешнее запоминающее. В качестве устройства отображения информации используется телевизионный приемник, имеющий вход «ВИДЕО»; в качестве внешнего запоминающего устройства — кассетный магнитофон. Остальные технические параметры БК сведены в табл. 6.1.

**Конструкция БК.** Конструктивно БК состоит из двух отдельных блоков: микроЭВМ, размещенной в одном корпусе с клавиатурой, и блока питания. Габаритные размеры пластмассового корпуса — 364x205x56 мм, а блока питания — 203x82x85 мм.

Размещение в одном корпусе микроЭВМ, выполненной на одной двусторонней плате, и клавиатуры обеспечивает уменьшение габаритных размеров, более высокую надежность и технологичность БК по сравнению с многоблочными конструкциями. Плата клавиатуры представляет собой коммутационную матрицу и не содержит радиоэлектронных компонентов.

На верхней лицевой панели корпуса БК-0010 на цветной шильд клавиатуры нанесена маркировка клавиш, причем отдельные функциональные группы клавиш выделены разными цветами. Клавиатура БК-0010-01 удобнее в работе, чем клавиатура БК-0010 и содержит 74, а не 93 клавиши.

На БК-0010 слева от клавиатуры расположен отсек пользователя, в котором расположены розетки типа РС-24 для установки съемных ПЗУ и переключатель «стоп— пуск» для перезапуска микроЭВМ. Перезапуском микроЭВМ пользуются, если с помощью клавиши «СТОП» на клавиатуре не удается прервать выполнение программы и передать управление системной программе — интерпретатору ФОКАЛа или БЕЙСИКа. Перезапуск микроЭВМ БК-0010-01 производится переключателем на блоке питания.

На задней панели корпуса компьютера расположены разъемы для подключения к микроЭВМ блока питания (БП), кассетного магнитофона (МГ), периферийных устройств (УП), цветного или черно-белого телевизора (ЦТВ и ТВ) и т. д.

БК формирует полный телевизионный сигнал, подаваемый на разъем ТВ и содержащий все необходимые для работы сигналы синхронизации черно-белого телевизора. Полный видеосигнал не несет информации о цвете, поэтому па выходе разъема ЦТВ БК вырабатывает сигналы цветности — красный, зеленый, синий (R, G, B), которые через пятипроводный кабель подаются непосредственно на соответствующие усилители цветного телевизора и управляют интенсивностью каждого из трех основных цветов (табл. 6.2).

Таблица 6.1

**Основные технические характеристики БК**

| Наименование параметра | Значение |
|---|---|
| Разрядность, бит | 16 |
| Тактовая частота, мГЦ | 3 |
| Быстродействие выполнения операций типа «регистр—регистр», тыс. оп./с | 300 |
| Число регистров общего назначения | 8 |
| Объем адресного пространства, Кбайт | 64 |
| Объем оперативной памяти (ОЗУ), Кбайт: | 32 |
| из нее экранной памяти | 16 (4) |
| Объем ПЗУ, Кбайт: | 32 |
| из него ПЗУ с резидентным программным обеспечением в БК-0010 (БК-0010Ш) в блоке МСТД в БК-0010-01 | 24<br>16 |
| Число воспроизводимых символов в КОИ-7 | 96 |
| Число клавиш клавиатуры: | |
| БК-0010 (БК-0010Ш)<br>БК-0010-01 | 93<br>74 |

| | |
|---|---|
| Число интерфейсов | 3 |
| Разрядность магистрального параллельного интерфейса (МПИ), бит | 16 |
| Разрядность интерфейса для радиального подключения устройств с последовательной передачей информации (ИРПС), бит | 1 |
| Скорость обмена по ИРПС, бит/с | 50-9600 |
| Разрядность параллельного программируемого интерфейса, бит | 16 |
| Скорость обмена информацией с накопителем на кассетной магнитной ленте, бит/с | 1200 |
| Емкость одной кассеты, Кбайт | 512 |
| Число отображаемых цветов на устройстве отображения информации | 3 |
| Число информационных строк, высвечиваемых на экране телевизора (из них служебных) | 25 (1) |
| Число символов в строке при формате символа: | |
| 10 x 8 точек | 64 |
| 10 x 16 | 32 |
| Число точек в строке | 512 |
| » строк развертки | 256 |
| Потребляемая мощность (не более), Вт | 35 |

Таблица 6.2

**Разъемы подключения телевизора**

| Номер контакта разъема | Обозначения сигнала | Примечание |
|---|---|---|
| | Разъем «ТВ», розетка ОНЦ-КГ-5/16Р | |
| *1* | — | — |
| *2* | Общий | — |
| *3* | — | — |
| *4* | Видео | Амплитуда видеоимпульсов 0,7В—1,4В (на $R_н$ = 75 Ом) |
| *5* | — | — |
| | Разъем «ЦТВ», розетка ОНЦ-КГ-5/16Р | |
| *1* | Синхронизация | Амплитуда импульсов отрицательной полярности (0,7—1,0) В на $R_н$ = = 75 Ом |
| *2* | Общий | — |
| *3* | Зеленый (*G*) | Амплитуда видеоимпульсов положительной полярности 0,9В—1,1В на $R_н$ = 75 Ом |
| *4* | Синий (*B*) | |
| *5* | Красный (*R*) | |

Из-за отсутствия в отечественных телевизорах входа для приема разделенных сигналов цветности (R, G, B), разъем ЦТВ может быть не выведен на заднюю панель БК.

## 6.2. АРХИТЕКТУРА БК

**Структура и состав БК.** Структурная схема БК показана на рис. 6.1. Основу БК составляет одноплатная микроЭВМ на базе однокристального микропроцессора К1801ВМ1. Кроме микропроцессора микроЭВМ включает следующие функциональные узлы: оперативное запоминающее устройство (ОЗУ) емкостью 32 Кбайта; постоянное запоминающее устройство (ПЗУ) емкостью 32 Кбайта; клавиатуру с контроллером клавиатуры; параллельный программируемый интерфейс (порт) ввода—вывода; контроллеры кассетного магнитофона и телеграфного канала; видеоконтроллер устройства отображения информации; внутреннюю системную магистраль (типа Q-bus), через которую все перечисленные узлы микроЭВМ связаны с микропроцессором. Функциональная схема микроЭВМ приведена на рис. 6.2.

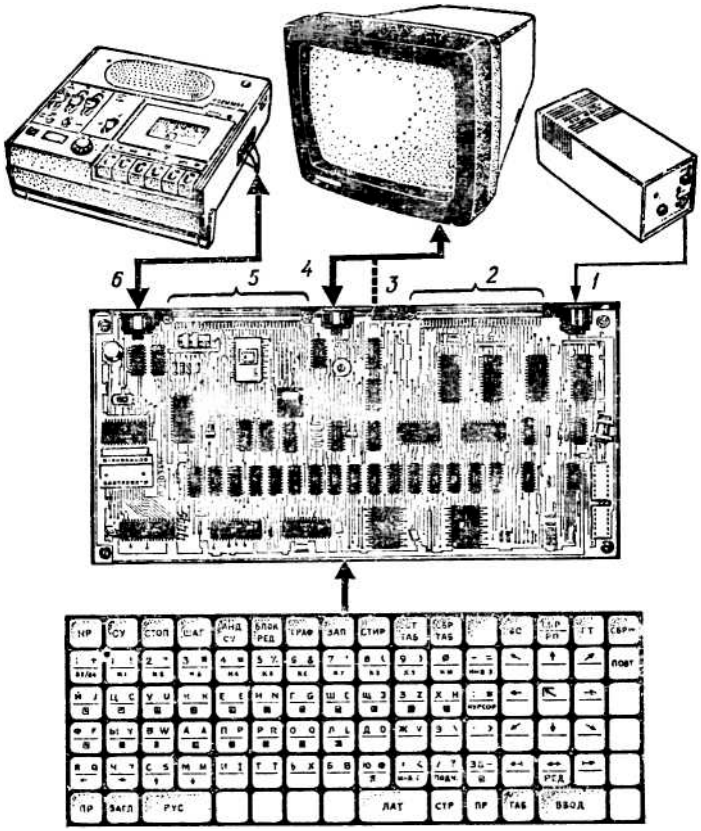



**Рис. 6.1. Структурная схема БК: *1* — разъем для подключения блока питания; *2* — разъем порта ввода—вывода; *3* — разъем для подключения цветного телевизора; *4* — разъем для подключения черно-белого телевизора; *5* — разъем системной магистрали микроЭВМ; *6* — разъем для подключения магнитофона**

Управление внешними устройствами БК. выполняет с помощью управляющих программ (драйверов), размещенных в системном ПЗУ (К1801РЕ1) объемом 8 Кбайт. Обращение к драйверам осуществляется с помощью системных запросов (команд EMT с заданным аргументом). Для настройки и тестирования микроЭВМ на внешний разъем выведена системная магистраль (разъем *5* на рис. 6.1). Однако вследствие малой нагрузочной способности магистрали подключение к ней внешних устройств не рекомендуется.

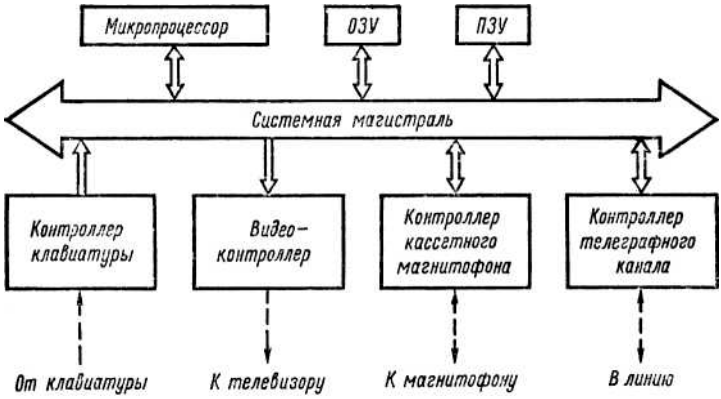



**Рис. 6.2. Функциональная схема микроЭВМ**

Входной язык БК-0010 — ФОКАЛ, а БК-0010-01 — ФОКАЛ или БЕЙСИК. Интерпретатор языка ФОКАЛ занимает объем памяти 8 Кбайтов и размещается в одной микросхеме съемного ПЗУ типа К1801РЕ2. Интерпретатор языка БЕЙСИК занимает объем памяти 24 Кбайта и размещается в трех микросхемах съемного ПЗУ типа К1801РЕ1.

Кроме того, БК-0010 содержит в отсеке пользователя колодку для подключения еще одной микросхемы съемного ПЗУ (объемом 8 Кбайтов) для размещения в ней программ пользователя. Обращение к этим программам может осуществляться с помощью системных запросов, для чего в системном ПЗУ зарезервировано 16 переходов по адресам, входящим в адресное пространство третьего сменного ПЗУ.

**Микропроцессор К1801ВМ1.** Процессор микроЭВМ выполнен на базе однокристального 16-разрядного микропроцессора К1801ВМ1 и предназначен для выполнения следующих функций: вычисления адресов операндов и команд; обмена информацией с другими устройствами, подключенными к системной магистрали; обработки операндов; обработки прерываний от клавиатуры и устройств пользователя, подключенных к разъему порта ввода—вывода. Процессор является единственным активным устройством микроЭВМ, управляющим циклами обращения к системной магистрали и обрабатывающим прерывания от пассивных устройств, которые могут посылать или принимать информацию только под управлением процессора.

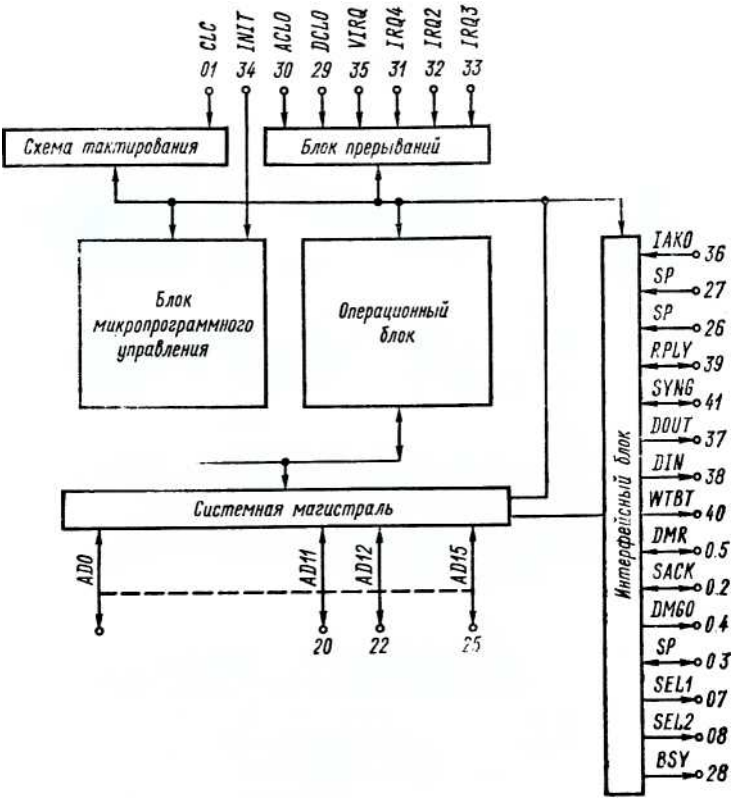



**Рис. 6.3. Структурная организация микропроцессора К1801ВМ1**

Микропроцессор К1801ВМ1 работает в БК с тактовой частотой 3 мГц и содержит следующие основные функциональные блоки (рис. 6.3).

1. *16-разрядный операционный блок*, выполняющий операции формирования адресов команд и операндов, логические и арифметические операции, хранение операндов и результатов.

2. *Блок микропрограммного управления*, вырабатывающий последовательность микрокоманд на основе кода принятой команды. Этот блок выполнен на базе программируемой логической матрицы (ПЛМ), содержащей 250 логических произведений. В ПЛМ закодирован полный набор микрокоманд для всех типов команд.

3. *Блок, прерываний,* организующий приоритетную систему

прерываний. Этот блок выполняет прием и предварительную обработку внешних и внутренних запросов на прерывание вычислительного процесса.

4. *Интерфейсный блок*, отвечающий за обмен информацией между микропроцессором и устройствами, расположенными на системной магистрали. Этот блок осуществляет арбитраж при операциях прямого доступа к памяти, формирует последовательность управляющих сигналов.

5. *Блок системной магистрали*, связывающий внутреннюю магистраль однокристального микропроцессора с внешней. Этот блок управляет усилителями приема и выдачи информации на совмещенные выводы адресов и данных.

6. *Схема тактирования*, обеспечивающая синхронизацию работы внутренних блоков микропроцессора.

Система команд, реализованная в ПЛМ блока микропрограммного управления микропроцессора, совпадает с системой команд наиболее распространенных отечественных мини- и микроЭВМ типа «Электроники-60» (ДВК-2, 3, 4 и т. п.) и приведена в приложении 4. Помимо этих команд однокристальный микропроцессор имеет в своем составе специальные команды, предназначенные для работы с системным ПЗУ К1801РЕ1.

Сигналы AD0 ... AD15 (рис. 6.3) определяют адреса и данные, которые передаются по совмещенной системной магистрали. Передача адресов и данных по одним и тем же выводам обеспечивается разделением во времени.

Группа сигналов SYNC, DIN, DOUT, WTBT, RPLY управляет передачей информации по системной магистрали. Сигнал SYNC, вырабатываемый процессором, означает, что адрес находится на выводах системной магистрали. Этот сигнал сохраняет активный уровень до окончания текущего обмена информацией.

Сигнал RPLY означает, что данные приняты или установлены на информационных выводах. Этот сигнал вырабатывается пассивным устройством в ответ на сигналы DIN и DOUT. Сигнал DIN предназначен для организации двух процедур обмена информацией по магистрали: *ввода данных* (если микропроцессор готов принять данные от пассивного устройства во время действия сигнала SYNC, то он вырабатывает сигнал DIN) и ввода адреса вектора прерывания (сигнал DIN вырабатывается совместно с сигналом IAK0 при пассивном уровне сигнала SYNC). Сигнал DOUT означает, что данные, выдаваемые микропроцессором,

установлены на выводах системной магистрали.

Сигнал WTBT вырабатывается в адресной части цикла для указания о том, что далее следует вывод данных и формируется при выводе данных из микропроцессора для указания о выводе байта.

Сигнал YIRQ вырабатывается внешним устройством для информирования микропроцессора о том, что оно готово передавать адрес вектора прерывания. Если прерывание разрешено, то в ответ на этот сигнал процессор вырабатывает сигналы DIN и IAK0.

Сигнал VRQ1 определяет положение внешнего переключателя «программа—пульт». Низкий уровень этого сигнала означает, что переключатель находится в положении «пульт». Этот сигнал переводит микропроцессор в состояние, которое наступает после выполнения машинной команды HALT.

Сигналы IRQ2 и IRQ3 вызывают прерывание программы, выполняемой процессором, по фиксированным адресам $100_8$ и $270_8$ соответственно. Прерывание возникает при переходе сигналов из высокого уровня в низкий.

Сигнал предоставления прерывания IAK0 процессор вырабатывает в ответ на внешний сигнал VIRQ. Этот сигнал — выходной для микропроцессора и входной для устройства, ближе всего расположенного к микропроцессору на системной магистрали и, следовательно, имеющего более высокий приоритет. Если это устройство не требовало прерывания (не выставляло сигнал VIRQ), то оно транслирует сигнал IAK0 к следующему устройству. Первое из устройств, выставившее сигнал VIRQ, запретит распространение этого сигнала. Сигнал IAK0, последовательно проходя через все устройства, обеспечивает их поочередный опрос и различный приоритет обслуживания.

Сигнал DMR вырабатывает внешнее активное устройство, требующее передачи ему системной магистрали. В ответ на сигнал DMR процессор устанавливает сигнал DMGO, который последовательно проходит через внешние устройства и предоставляет системную магистраль устройству с наивысшим приоритетом, запросившему прямой доступ к памяти. Это устройство прекращает трансляцию сигнала DMGO и выставляет сигнал SACK, который вырабатывается устройством прямого доступа к памяти (ПДП) в ответ на сигнал DMGO. Сигнал SACK означает, что устройство ПДП может производить обмен данными, используя стандартные циклы обращения к системной магистрали.

Низкий уровень сигнала BSY означает, что микропроцессор

начинает обмен по магистрали. Переход сигнала из низкого уровня в высокий указывает окончание обмена.

Сигнал аварии источника питания DCLO вызывает установку микропроцессора в исходное состояние и появление сигнала INIT; сигнал аварии сетевого питания ACLO вызывает переход микропроцессора на обработку прерывания по сбою питания (высокий уровень этого сигнала свидетельствует о том, что сетевое напряжение питания нормально).

Сигнал SEL1 устанавливается при обращении к регистру управления системного внешнего устройства (адрес $177716_8$), а сигнал SEL2 — при обращении к регистру порта ввода—вывода (адрес $177714_8$). Ввод данных в микропроцессор из этих регистров или вывод из него на регистры происходит совместно с сигналами DIN или DOUT соответственно. Выстановление сигнала RPLY от этих регистров не требуется. Длительности сигналов SEL1 и SEL2 совпадают с сигналом BSY.

Сигнал INIT является ответом микропроцессора на сигнал DCLO. Его используют для установки периферийной части системы в исходное состояние. При вводе этого сигнала в процессор происходит сброс триггеров запроса радиальных прерываний и блокирование сигнала DMR. Общие технические характеристики микропроцессора К1801ВМ1 приведены в табл. 6.3.

**Системная магистраль** представляет собой систему сигнальных связей и позволяет адресовать 64 Кбайта. Старшие 8 Кбайт адресного пространства зарезервированы для управления периферийными устройствами и регистрами данных. Магистраль позволяет организовать канал обмена информацией, аналогичный каналу микроЭВМ «Электроника-60» (Q-bus), в котором связь между двумя устройствами осуществляется по принципу «активный— пассивный». Активное устройство управляет прохождением информации по системной магистрали, разрешает прерывания, обеспечивает предоставление прямого доступа к памяти. Пассивное устройство передает информацию только под управлением активного устройства. Кроме микропроцессора, активным устройством на системной магистрали может быть устройство прямого доступа к памяти, которое выполняет адресацию, синхронизацию, вырабатывает управляющие сигналы для организации стандартных циклов обращения к системной магистрали без вмешательства микропроцессора.

Таблица 6.3

**Технические характеристики микропроцессора К1801ВМ1**

| Характеристика | Описание |
|---|---|
| Представление числа | В дополнительном коде с фиксированной запятой |
| Система команд | Безадресная, одноадресная, двухадресная |
| Виды адресации | Регистровая, косвенно-регистровая, автоинкрементная, косвенно-автоинкрементная, автодекрементная, косвенно-автодекрементная, индексная, косвенно-индексная |
| Число регистров общего назначения | 8 |
| Число уровней прерываний | 4 |
| Системная магистраль | Типа Q-bus (МПИ, ОСТ 11.305.903—80) |
| Адресное пространство, Кбайт | 64 |
| Тактовая частота. мГц | До 5 |
| Максимальное быстродействие при выполнении регистровых операций, опер./с | До 500 тыс. |
| Потребляемая мощность, Вт | До 1 |
| Напряжение питания, В Уровни сигналов, В: | +5 (±5%) |
| «логический 0» (активный уровень) | Менее 0,5 |
| «логическая 1» | Более 2.4 |
| Нагрузочная способность по току, мА | 3,2 |
| Емкость нагрузки, пФ | До 100 |
| Технология изготовления | N-МОП |
| Конструкция | 42-выводной планарный металлокерамический корпус |

На задней панели корпуса БК установлен разъем ХТ3 (разъем 5, см. рис. 6.1), на который выведена системная магистраль. Соответствие контактов разъема ХТ3 сигналам магистрали приведено в табл. 6.4.

**Регистры общего назначения.** Эти регистры (R0—R7) используются в качестве индексных и накопительных регистров автоинкрементной и автодекрементной адресации.

**Соответствие контактов разъема ХТ3 сигналам системной магистрали**

| Номер монтажа разъема ХТ3 | Обозначение сигнала в магистрали | Обозначение в соответствии с условным обозначением процессора | Наименование сигнала магистрали |
|---|---|---|---|
| А1 | ОСТ Н | IRQ1 | Останов |
| А2 | Общий | — | Общий |
| А3 | » | — | » |
| А4 | +5 В (контроль) | — | Контроль источника питания + 5 В |
| А5 | ПРТ Н | IRQ2 | Требование прерывания |
| А23 | ВВОД Н | DIN | Ввод данных |
| А25 | ДА13 Н | AD13 | Линия адреса данных |
| А26 | ДА11 Н | AD11 | » » » |
| А27 | ДА09 Н | AD9 | » » » |
| А28 | ДА05 Н | AD5 | » » » |
| А31 | ДА00 Н | AD0 | » » » |
| В2 | Общий | — | Общий |
| В3 | » | — | » |
| В4 | ППР1 Н | IAK1 | Входной сигнал предоставления прерывания |
| В4 | ППР1 Н | IAK0 | Выходной сигнал предоставления прерывания |
| В5 | ТПР Н | VIRQ | Требование прерывания |
| В7 | ДА15 Н | AD15 | Линия адреса данных |
| В11 | БАЙТ Н | WTBT | Вывод байта |
| В19 | СБРОС Н | INIT | Первоначальная установка магистрали |
| В20 | СИП Н | RPLY | Сигнал синхронизации пассивного устройства (ответ) |
| В21 | ВЫВОД Н | DOUT | Вывод данных |
| В22 | СИА Н | SYNC | Сигнал синхронизации активного устройства |
| В23 | ДА14 Н | AD14 | Линия адреса данных |
| В24 | ДА12 Н | AD12 | » » » |
| В25 | ДА10 Н | AD10 | » » » |
| В26 | ДА08 Н | AD8 | » » » |
| В27 | ДА06 Н | AD6 | » » » |
| В28 | ДА04 Н | AD4 | » » » |
| В29 | ДА02 Н | AD2 | » » » |
| В30 | ДА03 Н | AD3 | » » » |
| В31 | ДА01 Н | AD1 | » » » |
| В32 | ДА07 Н | AD7 | » » » |

Из них два регистра R6 и R7 имеют специальное назначение: R6 (SP) используется как системный указатель стека (УС) и содержит адрес последней ячейки стека, а R7 (PC) — счетчик команд (СК) и содержит адрес очередной выполняемой команды.

Кроме регистров общего назначения программно доступным регистром микропроцессора является регистр состояния процессора, содержащий информацию о текущем приоритете, коды условий ветвления программы и состояния «Т-бита», используемого при отладке программы.

**Функциональные узлы микроЭВМ.** *Память* состоит из оперативного запоминающего устройства (ОЗУ) емкостью 32 Кбайт и постоянного запоминающего устройства (ПЗУ) с адресным пространством 32 Кбайта. Блок ОЗУ включает 16 микросхем ОЗУ динамического типа КР565РУ6Г с организацией 1х16 Кбит. ПЗУ представляет собой БИС К1801РЕ1 (К1801РЕ2) емкостью 4 Кбайт 16-разрядных слов и схемой связи с системной магистралью. В БК-0010 предусмотрена возможность установки БИС ПЗУ с программой пользователя в свободную розетку, расположенную в отсеке пользователя.

*Видеоконтроллер* одновременно с управлением и регенерацией ОЗУ динамического типа выполняет функции контроллера бытового телевизора.

Независимая схема регенерации обеспечивает чтение слов из ОЗУ и вывод их на экран. Контроллер обеспечивает формирование и отображение на черно-белом или цветном бытовом телевизоре алфавитно-цифровой и графической информации в трех основных режимах: 1) монохромном, двойного (высокого) разрешения с размером рабочего поля 512 точек по горизонтали и 256 точек по вертикали при двух градациях яркости; 2) цветном, среднего разрешения с прямоугольным растром 256х256 точек и с возможностью окрашивания любой засвеченной точки в один из четырех цветов; 3) монохромном, низкого разрешения (256х256 точек) при четырех градациях по яркости. Формат кадра позволяет формировать 24 информационных и одну служебную строку. В каждой строке в зависимости от режима работы может размещаться 32 символа (матрица 10х16 точек) или 64 символа (матрица 10х8 точек). Контроллер выполнен на БИС К18018ВП1.

*Контроллер клавиатуры* предназначен для формирования и параллельного ввода кодов символов (в КОИ-7) 168

в системную магистраль микроЭВМ. Он устанавливает соответствие между нажатой клавишей и ее семиразрядным кодом.

Контроллер выполнен на БИС К18018П1.

*Контроллер кассетного магнитофона* построен по принципу программно-аппаратного управления. Все основные функции контроллера реализованы на программном уровне. Аппаратная часть состоит из регистра, предназначенного для временного хранения информации, и схемы преобразователей уровня сигналов. Контроллер обеспечивает запись до 256 Кбайт на стандартную магнитофонную кассету типа МК60 со скоростью 1200 бит/с, метод записи на магнитную ленту — разновидность широтно-импульсной модуляции.

*Контроллер телеграфного канала* (*последовательный программируемый интерфейс*) выполнен программно. Он обеспечивает обмен данными с внешними устройствами по протоколу обмена RS-232 в диапазоне скоростей от 50 до 9600 бит/с.

*Порт ввода—вывода* (*параллельный программируемый интерфейс ввода—вывода*) предназначен для подключения устройств пользователя к микроЭВМ через 16 информационных линий ввода и 16 линий вывода. Согласование осуществляется по уровням ТТЛ.

Кроме информационных линий интерфейс содержит две дополнительные линии: СБРОС (для начальной установки устройств пользователя) и ПТР (запрос на прерывание от устройств пользователя по фиксированному вектору $100_8$). Устройство пользователя должно получать питание от отдельного источника.

Соответствие контактов разъема порта УП (разъем *2* см. рис. 6.1) сигналам параллельного программируемого интерфейса приводится в таблице 6.5.

**Распределение адресного пространства.** Постоянное и оперативное запоминающие устройства, в том числе область экранной (растровой) памяти БК, расположены в едином адресном пространстве, объем которого определяется длиной слова микропроцессора К1801ВМ1 (16 разрядов) и равен 64 Кбайтов.

Половину этого адресного пространства (32 Кбайта) занимает ОЗУ с адресами от 0 до $77777_8$ (остальное пространство отведено под ПЗУ и системные регистры микроЭВМ). Распределение адресного пространства показано на рис. 6.4, причем здесь и далее значения адресов даются в восьмеричной системе счисления.

Область ОЗУ с адресами 0—777 отведена под системный стек, переменные драйверов и вектора прерываний. Стек может размещаться в ячейках с адресами от 776 до 300, однако, следует

Таблица 6.5

**Параллельный программируемый регистр ввода—вывода (разъем «УП» — вилка СНП58-64/94Х 9 В-23-В)**

| Номер контакта разъема | Обозначение сигнала | Примечание | Номер контакта разъема | Обозначение сигнала | Примечание |
|---|---|---|---|---|---|
| | Регистр вывода | Управляются программно; | В20 | ВВ04 | $J_{1L} \le 2{,}75$ мА; |
| | | | А20 | ВВ05 | $J_{1H} \le 0{,}01$ мА; |
| А16 | ВД00 | $U_{1L} \le 0{,}4$ В; | В22 | ВВ06 | $J_{0H} \le 1{,}0$ мА; |
| А13 | ВД01 | $U_{1H}$ = | А23 | ВВ07 | $J_{0L} \le 15{,}0$ мА |
| В12 | ВД02 | = (2,4÷5,0) В; | В31 | ВВ08 | |
| В10 | ВД03 | $U_{0H}$ = | В32 | ВВ10 | |
| В5 | ВД04 | = (3,65÷5,0) В; | А32 | ВВ11 | |
| В7 | ВД05 | $U_{0L} \le 0{,}5$ В; | В30 | ВВ12 | |
| В6 | ВД06 | $J_{1L}$ = | А29 | ВВ13 | |
| А7 | ВД07 | ≤ -2,75 мА; | В29 | ВВ14 | |
| А28 | ВД08 | $J_{1H} \le 0{,}01$ мА; | А30 | ВВ15 | |
| В28 | ВД09 | $J_{0H} \le 1{,}0$ мА; | В1 | ПРТ | |
| А27 | ВД10 | $J_{0L} \le 15{,}0$ мА | А1 | СБРОС | |
| В27 | ВД11 | | | Питание | |
| А26 | ВД12 | | А8 | +5 В | $I_{max}$ = 150 мА |
| В26 | ВД13 | | В8 | То же | |
| А25 | ВД14 | | А9 | » | |
| В25 | ВД15 | | В9 | » | |
| | Регистр ввода | Управляются программно; | А11 | Общий | |
| | | | В11 | » | |
| В24 | ВВ00 | $U_{1L} \le 0{,}4$ В; | А18 | » | |
| А24 | ВВ01 | $U_{1H} \ge 2{,}4$ В; | В18 | » | |
| В23 | ВВ02 | $U_{0H} \ge 3{,}65$ В; | А19 | » | |
| В17 | ВВ03 | $U_{0L} \le 0{,}5$ В; | В19 | » | |

иметь в виду, что ячейки с адресами 300— 352 использует драйвер магнитофона. При запуске микроЭВМ монитор устанавливает указатель стека на адрес 1000. Если для работы пользовательской программы необходим стек большего размера, то программа может поместить в указатель стека (УС, адрес 177774) требуемое значение.

Область ОЗУ с адресами 1000—37777—рабочая область памяти, используемая пользовательскими программами (например, ФОКАЛом или БЕЙСИКом). Область ОЗУ с адресами 40000—77777 является экранной (растровой) памятью и служит

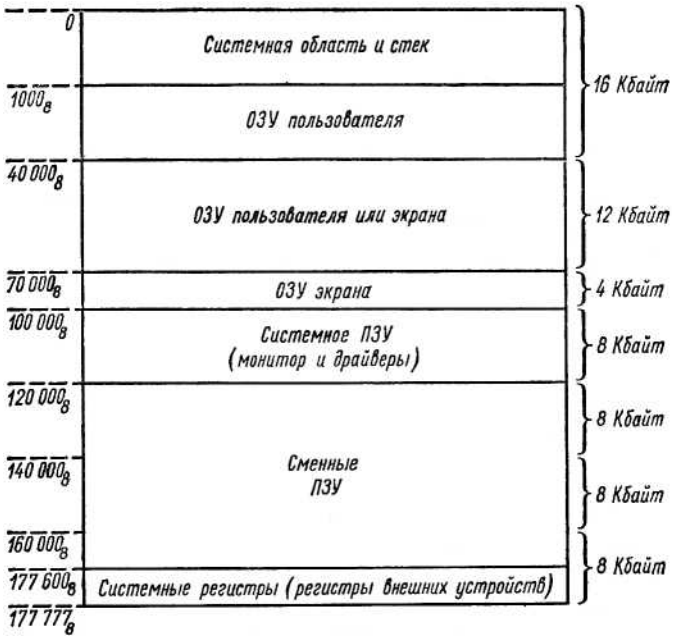



**Рис. 6.4. Распределение адресного пространства микроЭВМ**

для формирования изображения, выводимого на экран телевизора.

МикроЭВМ может работать с расширенным объемом рабочей области ОЗУ. В этом режиме расширенной памяти часть ОЗУ экрана отводится под рабочую область, а формирование изображения осуществляется в области ОЗУ с адресами 70000—77777. Это позволяет увеличить объем рабочей области ОЗУ с 16 до 28 Кбайт. Переключение в режим расширенной памяти (и обратно) осуществляется специальным кодом $214_8$ ($14O_{1o}$), который можно подать с клавиатуры (нажав клавиши «НР-РП» на БК-0010 или «↓ + СБР» на БК-0010-01) или непосредственно из программы XECUTE FCHR (140) на драйвер телевизора.

Оставшиеся 32 Кбайта адресного пространства предназначены для размещения четырех ПЗУ объемом 8 Кбайт каждая с записанными в них системными и пользовательскими программами. Распределение адресного пространства между микросхемами различно для БК-0010 и БК-0010-01 и приведено на

| | |
|---|---|
| $120\ 000_8$ | *1-е сменное ПЗУ - фокал* |
| $140\ 000_8$ | *2-е сменное ПЗУ - резервное* |
| $160\ 000_8$ | *3-е сменное ПЗУ-мониторная система диагностики* |
| $177\ 600_8$ | |

**Рис. 6.5. Распределение сменных ПЗУ для «Электроники БК-0010»**

рис. 6.5 и 6.6. В обеих модификациях БК адреса 100000—117777 занимают ПЗУ с пусковым монитором и драйверами внешних устройств.

Область адресов 177600—177777 отведена под системные регистры микроЭВМ. В БК-0010Ш под системные регистры и РПС отведены также адреса 176560—176566. При подготовке программ для записи в третье сменное ПЗУ следует учесть, что адреса, выделенные под системные регистры, использоваться не могут. Если в комплект БК-0010-01 входит блок ПЗУ с мониторной системой диагностики (блок МСТД), то при подключении блока к разъему системной магистрали микроЭВМ это ПЗУ устанавливается в область с адресами 160000—177600. При этом блокируется работа третьего сменного ПЗУ, установленного на плате микроЭВМ в той же области адресов.

**Системные регистры.** Эти регистры размещаются в области адресов 177600—177777. К ним относятся следующие регистры: клавиатуры, смещения, порта ввода—вывода, управления системными внешними устройствами, общего

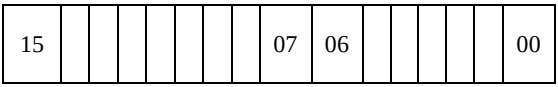
| | |
|---|---|
| $120\ 000_8$ | *1-е сменное ПЗУ - БЕЙСИК* |
| $140\ 000_8$ | *2-е сменное ПЗУ - БЕЙСИК* |
| $160\ 000_8$ | *3-е сменное ПЗУ - БЕЙСИК* |
| $177\ 600_8$ | |

**Рис. 6.6. Распределение сменных ПЗУ для «Электроники БК-0010-01»**

назначения и, для БК-0010Ш, регистры блока ИРПС, расположенные по адресам 176560—176566.

*Регистры клавиатуры* — это регистры состояния и данных клавиатуры.

Регистр состояния клавиатуры расположен по адресу 177660, отображает состояние клавиатуры и имеет следующий формат:

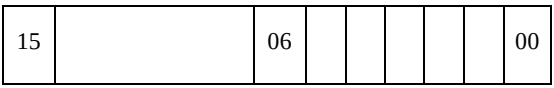



Разряд 6 — бит разрешения прерываний от клавиатуры: «0» — разрешено, «1» — запрещено.

Разряд 7 — флаг состояния клавиатуры (устанавливается в единицу при поступлении в регистр данных клавиатуры нового кода и сбрасывается в нуль при чтении регистра данных клавиатуры). Если разряд 6 установлен в нуль, то при установке разряда 7 в единицу по системной магистрали идет запрос на прерывание от клавиатуры.

Разряд 6 доступен по записи и чтению, а разряд 7 — только по чтению. Остальные разряды не используются.

Регистр данных клавиатуры расположен по адресу 177662, предназначен для записи кодов клавиатуры и имеет следующий формат:




Разряды 0—6 используются для записи кода клавиши клавиатуры. Регистр доступен только по чтению. Запись нового кода в регистр не производится до тех пор, пока не будет прочитан предыдущий код. Разряды 7—15 не используются .

*Регистр порта ввода—вывода* предназначен для записи информации в выходной регистр порта и чтения информации из входного регистра порта; адрес регистра — 177714, а его связь с выходными и входным регистрами порта показана на рис. 6.7 (см. также табл. 6.5).

Все 16 разрядов регистра используются для записи и чтения информации, причем регистр доступен только по записи в выходной регистр порта и только по чтению из входного регистра порта, т. е. отсутствует возможность прочитать содержимое

выходного регистра порта.

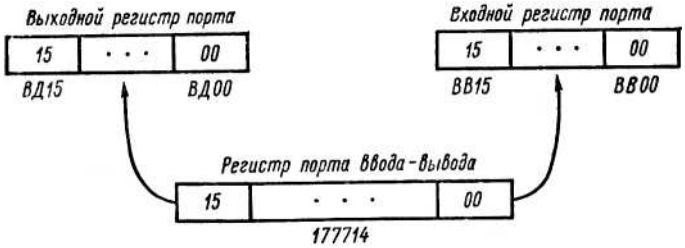



**Рис. 6.7. Работа с портом ввода—вывода**

*Регистр смещения* предназначен для рулонного сдвига информации на экране по вертикали путем задания того адреса растровой памяти, с которого должен начинаться вывод информации на экран.

Регистр смещения имеет адрес 177664, а формат регистра имеет вид

| 15 | | 09 | | 07 | | | | | | | 00 |
|---|---|---|---|---|---|---|---|---|---|---|---|

Разряды 0—7 (младший байт) предназначены для задания адреса растровой памяти. Изменение содержимого младшего байта регистра смещения на 1 соответствует изменению адреса растровой памяти на $100_8$ байт ($512_{10}$ бит, т. е. соответствует длине информационного поля, помещаемого на экране в одной телевизионной строке). Таким образом, изменяя на 1 содержимое регистра смещения можно сместить изображение на экране на 1 телевизионную строку вверх или вниз. При запуске процессора в регистр смещения помещается константа $1330_8$ ($728_{10}$) и видеоконтроллер отображает на экране растровую память, начиная с адреса $40000_8$. Все последующие изменения регистра смещения, обеспечивающие рулонный сдвиг изображения на экране, производятся относительно этой константы.

Разряд 9 используется для управления режимом расширенной памяти; ноль в этом разряде задает режим отображения на экране 1/4 части растровой памяти с адресами 70000—77777 (режим расширенной памяти).

Разряды 8, 10—15 не используются. Регистр доступен по записи и чтению.

*Регистр управления системными внешними устройствами* используется для задания адреса начального пуска процессора, а также для управления внешними устройствами микроЭВМ, и имеет адрес 177716. На самом деле адрес 177716 имеет два регистра, один из которых доступен только по записи (выходной регистр), а другой — только по чтению (входной регистр).

Формат регистра следующий:

| 15 | | 08 | 07 | 06 | 05 | 04 | 03 | | | 00 |
|---|---|---|---|---|---|---|---|---|---|---|

Разряды 8—15 содержат адрес пуска процессора при включении питания, причем содержимое младшего байта считается равным нулю (в БК адрес начального пуска процессора равен $100000_8$). Эти разряды доступны только по чтению.

Разряды 0—3 служат для задания режимов работы процессора и доступны только по чтению.

Разряды 4—7 предназначены для управления системными внешними устройствами микроЭВМ и имеют следующие значения:

*по чтению* (т. е. во входном регистре)

разряд 4 — бит линии радиального интерфейса (0 — прием «логического 0», 1 — прием «логической 1»);

разряд 5 — бит линии чтения информации с магнитной ленты;

разряд 6 — служит индикатором нажатия клавиши на клавиатуре (0 — клавиша нажата, 1 — клавиша не нажата);

разряд 7 — используется для чтения сигнала готовности с линии;

*по записи* (т. е. в выходном регистре)

разряд 4 — бит линии радиального интерфейса (0 — передача «логического 0», 1 — передачи «логической 1» — исходное состояние разряда);

разряд 5 — используется для передачи либо информации на магнитофон, либо сигнала готовности в линию (исходное состояние — «логического 0»); одновременный обмен информацией с магнитофоном и по телеграфному каналу недопустим;

разряд 6 — используется для передачи информации на магнитофон; от содержимого разрядов 5 и 6 зависит уровень

записываемого сигнала;

разряд 7 — используется для управления двигателем магнитофона (0 — соответствует команде «ПУСК», 1 — соответствует команде «СТОП» — исходное состояние),

**Регистры блока ИРПС.** Блок ИРПС входит только в состав аппаратного обеспечения микроЭВМ БК-0010Ш, предназначен для обмена информацией между микроЭВМ и внешними устройствами по интерфейсу для радиального подключения устройств с последовательной передачей информации и содержит регистр состояния (адрес 176560) и буферный регистр приемника (адрес 176562), регистр состояния (адрес 176564) и буферный регистр передатчика (адрес 176566).

*Регистр состояния приемника* имеет следующий формат:

| 15 | | 12 | | | | | 07 | 06 | | | | | | 00 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Разряды 0—5, 8—11 и 13—14 не используются;

разряд 6 — разрешение работы по прерыванию (0 — запрещено, 1 — разрешено); разряд доступен по записи и чтению;

разряд 7 — флаг состояния приемника — устанавливается в «1» при поступлении посылки в буферный регистр приемника; доступен по чтению;

разряд 12 — ошибка переполнения — устанавливается в «1», если в буферный регистр приемника поступает посылка, а предыдущая не была прочитана; при этом независимо от числа поступивших в канал приемника посылок, в буферном регистре сохраняется первая посылка; доступен по чтению;

разряд 15 — ошибка в принятой посылке; доступен по чтению.

*Буферный регистр приемника* доступен по чтению. Младший байт буфера (разряды 0—7) содержит посылку, принятую с линии. В нулевом разряде находится первый бит посылки, в 7-м — восьмой. Разряды 8—15 не используются.

Регистр состояния передатчика имеет следующий формат:

| 15 | | 08 | 07 | 06 | | | | 02 | | 00 |
|---|---|---|---|---|---|---|---|---|---|---|

Разряды 1, 3—5, 8—15 не используются;

разряд 0 — разрыв линии; доступен по записи и чтению;

разряд 2 — проверка работы; доступен по записи и чтению;

разряд 6 — разрешение работы передатчика по прерыванию (0 — запрещено, 1 — разрешено); разряд доступен по записи и чтению;

разряд 7 — флаг состояния передатчика (0 — в буферный регистр передатчика записана информация, 1 — буферный регистр передатчика пуст); установка разряда 7 в «1» происходит в момент выдачи посылки в линию; если запись информации в буферный регистр передатчика произошла во время посылки, то следующая посылка начинается сразу же по окончании предыдущей; доступен по чтению.

*Буферный регистр передатчика* доступен по записи. В линию передается содержимое младшего байта буфера, т. е. разряды 0—7. Нулевой разряд — первый бит посылки, 7-й разряд — восьмой бит посылки; разряды 8—15 не используются.

## 6.3. КЛАВИАТУРА

**Описание клавиатуры.** В состав клавиатуры БК-0010 входят 88 клавиш (у клавиатуры БК-0010-01 — 74 клавиши), которые по функциональному назначению подразделяются на четыре группы (выделенные на шильде БК-0010 разными цветами): регистровые (синие), управляющие (красные), алфавитно-цифровые (зеленые) и редактирующие (желтые). Общий вид клавиатуры БК-0010-01 приведен на рис. 6.8. Из рис. 6.1 видно, что на все алфавитно-цифровые клавиши, большинство редактирующих клавиш и одну управляющую («СБР/РП») нанесены изображения нескольких символов: слева и справа над горизонтальной чертой и под горизонтальной чертой. Для удобства изложения в дальнейшем будем говорить о левых, правых и нижних символах клавиатуры. При нажатии на клавишу формируется код одного из изображенных на ней символов, который в дальнейшем обрабатывается программой —драйвером клавиатуры. Выбор символа и значение кода, генерируемого при нажатии клавиши, зависят от предшествовавших переключений регистров клавиатуры, т. е. от того, какие (или какая) регистровые клавиши были ранее нажаты. Коды и соответствующие им символы клавиатуры БК-0010 приведены в приложении 3.


**Рис. 6.8. Клавиатура «Электроники БК-0010-01»**

*Замечание.* Значительное число символов (строчные буквы английского и русского алфавитов, подчеркивание и вертикальная черта и т. д.) не нанесены на клавиши. Для более детального освоения клавиатуры мы советуем по прочтении этого раздела «вооружиться» приложением 3, занести в БК следующую программную строку:

```
*1.10 T %3,FCHR(FCHR(-1)),!;G
```

и (после задания директивы GOTO) последовательно нажимать клавиши клавиатуры на разных регистрах. При этом, как правило, на экран будет выводиться соответствующий десятичный код и его графический символ (если он есть). Однако при нажатии некоторых клавиш код не вырабатывается, а другие клавиши вызывают изменения в служебной строке (в самой верхней строке экрана) и (или) в информационном поле экрана.

**Регистровые клавиши.** К регистровым относятся следующие клавиши: РУС — включает русский регистр; ЛАТ — включает латинский регистр; ЗАГЛ — включает режим формирования кодов заглавных букв русского или латинского алфавита; СТР — включает режим формирования кодов строчных букв русского или латинского алфавита; НР — включает нижний регистр только в нажатом состоянии (при одновременном нажатии некоторой клавиши и «НР» вырабатывается код нижнего символа этой клавиши); ПР — включает правый регистр только в нажатом состоянии (при одновременном нажатии некоторой клавиши и «ПР» вырабатывается код правого символа этой клавиши); СУ — включает режим формирования управляющих кодов символов только в нажатом состоянии.

Клавиши ЗАГЛ, СТР, НР, ПР и СУ код при нажатии не вырабатывают. Коды клавиш РУС ($17_8$) и ЛАТ ($16_8$) используются только в драйвере клавиатуры и не передаются в рабочую

программу, однако, при передаче этих кодов в драйвер устройства отображения они отображаются в виде соответствующих инвертированных символов латинского алфавита. Режимы РУС (ЛАТ) индицируются в служебной строке.

**Управляющие клавиши.** Они предназначены для формирования кодов, управляющих режимами работы микро- ЭВМ; к ним относятся: СТОП — обеспечивает формирование запроса на прерывание и используется для прерывания выполнения рабочей программы; ШАГ — обеспечивает формирование управляющего кода, который может быть использован для управления режимами работы рабочей программы (нажатие на эту клавишу приостанавливает выполнение программы пользователя; для продолжения работы нужно нажать клавишу ВВОД); ИНД.СУ — устанавливает режим индикации управляющих символов, при котором управляющие символы отображаются на экране в виде негативного изображения соответствующих заглавных букв латинского алфавита (при установке режим индицируется в служебной строке; отмена режима происходит при повторном нажатии клавиши ИНД. СУ); БЛОК РЕД — устанавливает режим блокировки редактирования, при котором блокируется выполнение редактирующих функций, при этом редактирующие коды отображаются на экране в виде символов, соответствующих прорисовке на клавиатуре, а коды клавиш установки режимов (32/64, ИНВ. Э., КУРСОР, УСТ. ИНД., ИНВ.С., ПОДЧ) отображаются в виде инвертированных строчных символов; на коды клавиш ИНД.СУ, БЛОК РЕД, РП режим БЛОК РЕД не действует, а выполнение функций, вызываемых управляющими кодами СУ/*G* (звонок) и СБР, блокируется (при установке режим индицируется в служебной строке; отмена режима происходит при повторном нажатии клавиши БЛОК РЕД); УСТ.ТАБ, СБР.ТАБ — обеспечивают (совместно с клавишей ТАБ) управление механизмом табуляции устройства отображения информации; КТ — обеспечивает формирование управляющего кода 3 (конец текста); в режиме ИНД.СУ отображается на экране в виде инверсного символа *C*; СБР — при передаче кода клавиши СБР в драйвер устройства отображения происходит очистка экрана и перевод курсора в начало экрана (действие клавиши СБР в режиме БЛОК РЕД блокируется; если кроме режима БЛОК РЕД включен режим ИНД.СУ, то код СБР отображается в виде инверсного символа *L;* РП — производит переключение режима расширенной памяти; при включении данного режима объем рабочей области ОЗУ

увеличивается до 28 Кбайт, а для формирования изображения на экране используется 4 Кбайт, что позволяет отображать на экране одну служебную и четыре информационные строки (при установке режим индицируется в служебной строке; отмена режима происходит при повторном нажатии клавши РП); ПОВТОР — предназначена для многократного ввода одного и того же символа с клавиатуры (при нажатии этой клавиши в программу передается код последнего введенного символа); ВВОД — обеспечивает формирование управляющих кодов $12_8$ (перевод строки, ПС) или $15_8$ (возврат каретки, ВК), в зависимости от содержимого ячейки $262_8$; если ячейка содержит 0, то в рабочую программу передается код $12_8$ (ПС), который в режиме ИНД.СУ отображается в виде инверсного символа *J*; если содержание ячейки отлично от нуля, то передается код $15_8$ <ВК>, который в режиме ИНД.СУ отображается в виде инверсного символа М (при инициализации драйвера клавиатуры ячейка $262_8$ обнуляется).

Коды клавиш УСТ.ТАБ, СБР.ТАБ, ТАБ, ПОВТОР используются только в драйвере клавиатуры и не могут быть переданы в рабочую программу.

Ряд нижних символов, расположенных на алфавитно-цифровых клавишах, используется для управления режимами формирования информации на экране устройства отображения; к ним относятся: ИНВ.Э — обеспечивает инверсию поля экрана, т. е. переключение темного фона экрана в светлый, а изображения символов наоборот (при повторном нажатии клавиши происходит обратное переключение); при работе с цветным изображением фон экрана окрашивается в цвет, который был задан для формирования изображения, а изображение окрашивается в тот цвет, какой имел к этому моменту фон; ИНВ.С. — обеспечивает переключение режима инверсии символов; в этом режиме символы выводятся на экран в инвертированном виде (при установке режим индицируется в служебной строке; отмена режима происходит при повторном нажатии клавиши с символом ИНВ.С.); КУРСОР — обеспечивает гашение курсора, а при повторном нажатии — его восстановление на экране; гашение курсора увеличивает скорость вывода информации на экран; УСТ.ИНД. — обеспечивает установку режимов формирования индикаторов в служебной строке; при вводе кода этого символа происходит очистка служебной строки и установка режимов формирования индикаторов в служебной строке в соответствии с режимом, действующим в данный момент на рабочем поле экрана; ПОДЧ — обеспечивает установку режима

подчеркивания выводимых на экран символов (при установке режима индицируется в служебной строке; отмена режима происходит при повторном нажатии клавиши с символом ПОДЧ); 32/64 — обеспечивает последовательное переключение режимов формирования символов обычной (64 символа в строке) и удвоенной (32 символа в строке) ширины; в режиме «64 символа в строке» каждой точке на экране соответствует один бит растровой памяти, а в режиме «32 символа в строке» — два бита.

При вводе с клавиатуры коды ИНД.СУ, БЛОК РЕД, КУРСОР, 32/64, ИНВ.Э., УСТ.ИНД. в рабочую программу не поступают, а передаются из драйвера клавиатуры непосредственно в драйвер устройства отображения.

Еще четыре клавиши могут использоваться для управления работой с полутоновым или цветным изображением. Работа возможна только в режиме «32 символа в строке», при котором на управление (кодировку) яркостью или цветом используются два разряда каждого слова растровой памяти. Таким образом, можно получить четыре градации яркости или четыре цвета, при этом наибольшей яркости соответствует красный цвет, а далее, по мере убывания яркости, — зеленый, синий и черный. Переключение яркостей (цветов) осуществляется с помощью кодов «К» ($221_8$), «З» ($222_8$), «С» ($223_8$) и «Ч» ($224_8$), ввод которых с клавиатуры осуществляется с помощью цифровых клавиш 1, 2, 3, 4 соответственно при одновременно нажатых и удерживаемых клавишах НР и ПР.

**Алфавитно-цифровые клавиши.** Они обеспечивают ввод кодов цифр, заглавных и строчных букв русского и латинского алфавитов, специальных символов, кодов графических символов, а также управление программируемыми ключами К1—К10 (на нижнем регистре цифровых клавиш),

Ввод кодов цифр осуществляется при нажатии на цифровые клавиши.

Ввод кодов специальных символов, расположенных с правой стороны, осуществляется по правому регистру ПР, а ввод заглавных и строчных букв русского и латинского алфавитов — при включении соответствующей комбинации регистров РУС, ЛАТ, ЗАГЛ, СТР. О включении русского или латинского регистров свидетельствует индикатор в служебной строке. Кратковременное включение строчного или заглавного регистров можно выполнять с помощью клавиши ПР. При нажатии клавиши ПР на латинском регистре будет включаться заглавный регистр, а на русском —

строчной. Ввод кодов графических символов и управление программируемыми ключами осуществляется по нижнему регистру.

**Клавиши редактирования и управления режимом поточечной графики.** К редактирующим клавишам относятся следующие:

1) ←, →, ВС, ГТ, СБР↦, ⟷, ↤, ↦;
2) ↖, ↑, ↗, ⇱, ↙, ↓, ↘.

В символьном режиме ввода информации клавиши подгруппы 1 используются для редактирования либо текущей строки, либо строки, заданной оператором *MODIFY,* и описаны в п. 3.4. Нажатие клавиш подгруппы 2 никакого действия в этом режиме не оказывает.

В режиме редактирования, индицируемого в служебной строке символами РЕД, коды клавиш подгруппы 2 и клавиш ←, → вызывают перемещение курсора на одну позицию в направлении, указанном стрелкой (стрелка влево-вверх—перемещение курсора в начало экрана). Клавиши СБР|—>, <—|, |—> и <-/- функционируют как в символьном режиме; ВС вызывает перемещение нижней от курсора части экрана на одну строку вверх, ГТ — перемещение нижней от курсора части экрана на одну строку вниз, а служебный код *U* (СУ/*U*) вызывает перемещение курсора в начало следующей строки. Для перехода в режим редактирования необходимо нажать клавишу с символом РЕД на нижнем регистре. При повторном нажатии этой комбинации клавиш происходит выход из режима редактирования.

В режиме БЛР (индикатор нажатия клавиши БЛОК РЕД) редактирующие коды не вызывают соответствующих действий, а отображаются на экране в виде символов клавиатуры. Для перехода и последующей работы в режиме поточечной графики служат клавиши ГРАФ, ЗАП и СТИР. В этом режиме можно формировать на экране произвольное графическое изображение, используя кроме кодов управления режимом ГРАФ, ЗАП, СТИР цифровые коды и коды управления перемещением курсора (при наличии индикатора РЕД в служебной строке). При нажатии клавиши ГРАФ в служебной строке появляются индикатор ГРАФ, а на месте символьного курсора — графический в виде креста, центр которого указывает на адресуемую точку.

Нажатие клавиши ЗАП приводит к появлению индикатора ЗАП в позиции графического индикатора. В этом режиме происходит

запись точки в текущей позиции, указываемой курсором. При перемещении курсора с помощью редактирующих клавиш на экране остается траектория в виде последовательности записанных точек. Повторное нажатие клавиши ЗАП снова включает режим ГРАФ. При нажатии клавиши СТИР происходит переключение в режим стирания точки, определяемой положением курсора, а в позиции графического индикатора появляется индикатор СТИР. Переход в режим записи обеспечивается нажатием клавиши ЗАП, а возврат в графический режим — повторным нажатием клавиши СТИР.

Перемещение курсора без записи или стирания возможно только в графическом режиме, о котором свидетельствует индикатор ГРАФ в служебной строке. Для выхода из графического режима необходимо повторно нажать клавишу ГРАФ. При этом отменяется действовавший в данный момент режим (ГРАФ, ЗАП или СТИР), а в служебной строке гаснет индикатор и формируется символьный курсор Для редактирования информации, введенной в режиме поточечной графики, необходимо в символьном режиме включить режим БЛР, после чего вывести на экран графическую информацию, которая распечатается в виде последовательности символов. При этом коду ГРАФ соответствует негативное изображение буквы Г, ЗАП — З, СТИР — С. После выключения режима БЛР полученный текст можно редактировать обычным образом.

Режим редактирования и управления поточечной графикой — один из наиболее трудных для понимания и практического освоения. В гл. 5 приводится учебная программа, объясняющая некоторые из возможных приемов работы в этих режимах.

**Функционирование клавиатуры.** Клавиатура БК выполнена в виде печатной платы, на которой смонтированы клавиши, объединенные в координатную матрицу. Схема управления клавиатурой обеспечивает фиксацию координат нажатой клавиши, преобразование координат в код соответствующего символа, запись кода в регистр данных клавиатуры, выставление бита готовности в регистре состояния клавиатуры и формирование запроса на прерывание, если оно было разрешено (т. е. был выставлен бит 6 в регистре состояния клавиатуры). Клавиши НР, СУ, СТОП, ПОВТ, ПР, ЗАГЛ, СТР, РУС и ЛАТ в координатную матрицу не входят.

Клавиатура имеет два вектора прерывания с адресами $60_8$ и $274_8$. Это позволяет из 128 семиразрядных кодов, вырабатываемых

схемой управления клавиатуры, получить полный набор восьмиразрядных кодов, используемых в микроЭВМ. По вектору 274 обрабатываются коды, формируемые по нижнему регистру, а также некоторые коды, вырабатываемые группой управляющих клавиш. Остальные коды обрабатываются по вектору 60.

Обработка кодов, помещенных в регистр данных клавиатуры, выполняется драйвером клавиатуры, который читает код с регистра данных и либо передает его рабочей программе или драйверу устройства отображения, либо обрабатывает его сам.

Передача кода производится либо при поступлении запроса на чтение кода от рабочей программы, либо путем прерывания рабочей программы, в зависимости от значения бита 6 регистра состояния клавиатуры (1 — прерывание разрешено, 0 — запрещено).

Признак передачи кода по прерыванию — ненулевое содержимое ячейки $260_8$, которое рассматривается как адрес, по которому должно передаваться управление при обработке прерывания от клавиатуры. Получив управление, программа может прочитать код, выдав запрос на чтение (EMT 6), выполнить необходимые действия и выйти из прерывания (по команде RTS PC). Если содержимое ячейки $260_8$ нулевое, то передача кода осуществляется только по запросу рабочей программы на чтение кода.

Установка содержимого ячейки 260 выполняется рабочей программой (например, интерпретатором ФОКАЛа). При инициализации драйвера клавиатуры ячейка 260 обнуляется. Драйвер предоставляет возможность пользоваться аппаратом горизонтальной табуляции. Для установки табуляции в любой из 64 позиций в строке н обходимо подвести курсор в данную позицию и нажать клавишу УСТ.ТАБ. При этом в данной позиции под чертой, отделяющей служебную строку от рабочего поля экрана, появится соответствующая метка. При нажатии на клавишу ТАБ курсор переместится от текущей позиции до первой встретившейся затабулированной позиции, причем в рабочую программу будет передано соответствующее количество пробелов. Для сброса табуляции нужно подвести курсор в требуемую позицию и нажать клавишу СБР. ТАБ. При этом будет удалена соответствующая метка. Для ввода с клавиатуры отдельных часто употребляемых последовательностей символов пользователь может использовать аппарат программируемых ключей. Драйвер клавиатуры позволяет запрограммировать 10 ключей с номерами 1—10. Для этого служит

команда EMT 12 с соответствующими параметрами. Выдача текста ключей осуществляется с помощью цифровых клавиш по нижнему регистру.

Драйвер клавиатуры позволяет приостановить работу процессора путем ввода служебного кода символа @ (СУ/@). При повторном вводе этого или любого другого символа процессор продолжит свою работу.

**Команды драйвера клавиатуры. Инициализация драйвера клавиатуры.** По команде EMT 4 выполняется установка векторов прерываний клавиатуры, в регистре состояния сбрасывается бит разрешения прерываний от клавиатуры, устанавливаются режимы передачи кодов по запросам рабочей программы и передачи кода $12_8$ (перевод строки) при нажатии клавиши ВВОД. Содержимое регистра R0 не сохраняется.

**Чтение кода с клавиатуры.** Команда EMT 6 считывает код из регистра данных клавиатуры и записывает его в младший байт регистра R0, после чего управление возвращается вызвавшей программе.

**Чтение строки с клавиатуры.**

Команда: EMT 10.

Входные параметры: R1 — адрес ОЗУ для записи строки; R2 — код символа-ограничителя строки (старший байт) и длина строки (младший байт) в байтах.

По этой команде происходит запись в ОЗУ принимаемой с клавиатуры строки по адресу, предварительно заданному в регистре R0. Ввод строки завершается при выполнении одного из двух условий: либо встретился код символа-ограничителя строки, либо принято и записано указанное число символов. Если завершение ввода строки произошло по символу-ограничителю, то этот символ записывается в ОЗУ в конец строки. В процессе ввода для исправления допущенных ошибок можно пользоваться редактирующей клавишей <-|-, которая обеспечивает удаление последнего введенного символа. Если значение младшего байта регистра R2 равно нулю, то длина строки считается равной $200000_8$ байтов.

После завершения ввода строки в R1 содержится адрес байта, расположенного за последним введенным, а в R2 — разность между исходным значением и длиной введенной строки.

**Установка ключей клавиатуры.**

Команда: EMT 13.

Входные параметры: R0 — номер программируемого ключа (1—10), R1 — адрес текста ключа.

Эта команда выполняет программирование ключа с номером, указанным в регистре R0. Каждая цифровая клавиша задает ключ с соответствующим номером (от К1 до К10). При нажатии на одну из этих

клавиш на нижнем регистре (при одновременно нажатой клавише HP) драйвер клавиатуры выдает текст ключа. Адрес области ОЗУ, в которой записан текст ключа, задается в регистре R1, причем первый байт области должен содержать длину текста ключа. Если ключ не был запрограммирован, то реакция на нажатие этой клавиши на нижнем регистре отсутствует. Для отмены ключа необходимо задать в R1 нулевое значение адреса ключа. Содержимое регистра R0 при выполнении команды не сохраняется.

## 6.4. УСТРОЙСТВО ОТОБРАЖЕНИЯ ИНФОРМАЦИИ

**Характеристика устройств отображения информации.** *Устройство отображения информации* — периферийное устройство микроЭВМ, содержащее экран, средства записи, хранения и представления данных в удобной для восприятия форме и обеспечивающее оперативную информационную визуальную связь пользователя с микроЭВМ.

В современных персональных ЭВМ в качестве экранных устройств отображения информации используются либо бытовые телевизионные приемники, либо специальные мониторы на базе электронно-лучевых трубок (ЭЛТ).

Устройство отображения информации, графическое оперативное запоминающее устройство (ГОЗУ) или растровая память (для хранения закодированного изображения в форме матрицы значений интенсивностей свечения точек экрана) и интерфейсный блок, именуемый видеоконтроллером или видеопроцессором (выполняющий функцию регенерации ГОЗУ и формирование графического изображения на экране электронно-лучевой трубки в соответствии с содержимым ГОЗУ) — составляют основу графического дисплея.

Все графические дисплеи (ГД) по способу формирования изображения делятся на векторные и растровые. В векторных дисплеях графическая информация формируется из отдельных линейных отрезков (векторов) посредством адресации тех участков рабочего поля экрана, которые соответствуют отображаемым элементам, или, другими словами, в векторных дисплеях графическое изображение создается параметрическим методом, при котором траектория «изображающей» точки, проходящая по экрану, воспроизводит контур графического изображения. В ра-

стровых дисплеях информация представляется изменением интенсивности светового потока при последовательной адресации всех элементов рабочего поля экрана или заранее определенной его части.

По принципу поддержания изображения на экране, дисплеи делятся на два вида: с регенерацией изображения и с его запоминанием. В дисплее с регенерацией изображения применяются ЭЛТ с малым временем послесвечения люминофора, а в дисплее с запоминанием — запоминающая ЭЛТ (ЗЭЛТ) с большим временем послесвечения, что обеспечивает хорошее качество представления изображения на экране без его регенерации.

По способу обработки прикладных программ ГД подразделяются на «интеллектуальные» и «неинтеллектуальные». Интеллектуальный дисплей с расширенным составом устройств ввода и редактирования, развитыми вычислительными возможностями на базе встроенной микроЭВМ и программным обеспечением, включая прикладное, иногда называют графической рабочей станцией, а неинтеллектуальный дисплей — графическим терминалом.

По способу подключения к ЭВМ ГД подразделяются на локальные и удаленные, а по возможности цветового кодирования — на одноцветные (монохроматические) и многоцветные.

Графические дисплеи прошли длительный путь развития. Первые ГД, появившиеся в начале 60-х гг., были векторными, с регенерацией изображения. Они отличались высококачественным изображением и возможностью вывода на экран динамически изменяющихся картин. В настоящее время векторные ГД обеспечивают вывод изображений, содержащих до нескольких десятков тысяч векторов, однако высокая стоимость, большие габаритные размеры и потребляемая мощность этих устройств существенно ограничивают объем их выпуска.

С конца 60-х до конца 70-х гг. широкое распространение получили векторные дисплеи с запоминанием изображения, обладающие относительно невысокой стоимостью по сравнению с ГД с регенерацией изображения, высокой разрешающей способностью, отсутствием мелькания и большой информационной емкостью. Вместе с тем, применение ЗЭЛТ выявило и ряд их недостатков: невозможность избирательного стирания отдельных фрагментов изображения, невысокую яркость и небольшой срок службы. В настоящее время продолжается

совершенствование ГД на ЗЭЛТ: предусматривается использование микропроцессоров для выполнения в автономном режиме функций редактирования, введение одновременно с режимом запоминания режима регенерации части изображения, применение двухцветных ЭЛТ и т. п.

С конца 70-х гг. в связи со снижением стоимости полупроводниковой памяти, появлением мощных 16- и 32- разрядных микропроцессоров, цветных ЭЛТ с высокой разрешающей способностью и специальных БИС для управления ЭЛТ большое развитие получили растровые дисплеи. Они не имеют недостатков, свойственных ГД с запоминанием изображения, и обеспечивают дополнительные функциональные возможности, в том числе получение большого числа цветов, отображение (помимо элементов линейной и штриховой графики) всевозможных трехмерных объектов и полутоновых изображений и т. п.

К числу основных параметров ГД относятся следующие.

1. *Размер рабочего поля экрана*.

2. *Число адресуемых точек экрана*. Параметр характеризует плотность размещения элементов отображения и возможность воспроизведения непрерывных линий, поверхностей, мелких деталей изображения. Он измеряется максимальным числом позиций по каждой из осей координат рабочего поля экрана, в которые может быть направлен электронный луч, оставляющий на экране следы в виде небольших «точечных» пятен — пикселей. Соседние пятна могут перекрываться, если шаг между адресуемыми точками меньше размера пятна.

3. *Разрешающая способность*. Она характеризует свойство дисплеев раздельно представлять на экране мелкие детали изображения. Эта величина измеряется размером пикселя или максимальным числом визуально различимых пикселей, отображаемых по каждой из осей координат.

4. *Быстродействие.* Для дисплеев с регенерацией изображения выражается количеством векторов или символов, выводимых за время одного кадра без мелькания, а для растровых дисплеев — значением скорости или временем обновления изображения.

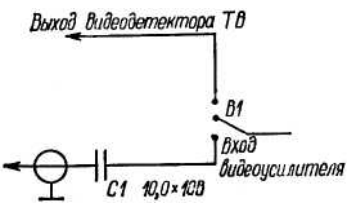


**Рис. 6.9. Схема организации видео- входа**

5. *Число одновременно отображаемых цветов* для многоцветных дисплеев или число

градаций яркости (полутонов) для монохроматических дисплеев.

6. *Качество изображения.* Оно характеризуется группой параметров, учитывающих величину нестыковки графических элементов, их нестабильность во времени, значение геометрических и нелинейных искажений изображения.

7. *Надежность.*

8. *Массогабаритные и энергетические показатели.*

9. *Тип и характеристика интерфейса* и т. п.

**Подключение устройства отображения информации к БК.** В БК в качестве устройства отображения информации можно использовать бытовой телевизор — монохромный или цветной, или специальный телевизионный монитор (отличающийся от обычного телевизора лишь отсутствием цепей приема и настройки).

Черно-белые телевизоры, имеющие вход видеосигнала (вход «ВИДЕО» или «ВМ» на задней панели телевизора), подключаются к БК просто с помощью коаксиального кабеля. При этом необходимо помнить, что БК не имеет на выходе управляющего напряжения + 12 В, которое нужно подавать на контакты 1 и 3 телевизионного разъема таким образом, чтобы плюс внешнего источника питания подключался к контакту 1.

У большинства отечественных моделей телевизоров такой вход отсутствует. Однако их несложно доработать с целью организации соответствующего входа для подачи сигнала непосредственно на выход видеодетектора [1]. Доработка сводится к установке дополнительного разъема и переключателя *П1* типа ПД1 (рис. 6.9). При разомкнутом переключателе схема не влияет на работу телевизора, а при его замыкании выходной сигнал видеодетектора шунтируется за счет подключения низкоомного резистора *R1* и телевизор работает как монитор БК.

Работа с цветными телевизорами требует, кроме входа видеосигнала, также организации входа для непосредственного управления яркостью *R-, G-, B-лучей* кинескопа. Серийные цветные телевизоры входов *R, G, B* не имеют и требуют более сложной доработки. Наиболее просто это может быть осуществлено в тех бытовых телевизорах, которые имеют отдельные видеоусилители по каждому из первичных цветов (красный, зеленый, синий).

Такому требованию удовлетворяют все унифицированные телевизоры, выпускаемые отечественной промышленностью. Отметим, что доработка ламповых телевизоров ранних моделей, имеющих один широкополосный видеоусилитель, вследствие

большой трудоемкости, на наш взгляд, нецелесообразна.

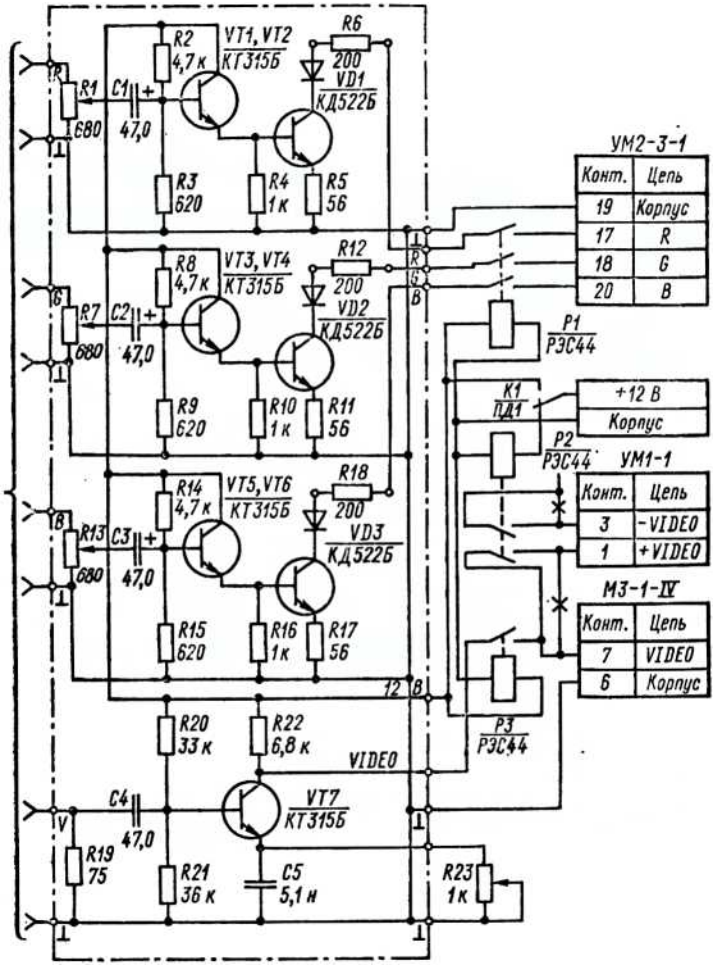



**Рис. 6.10. Принципиальная схема согласующей платы**

Ниже описывается вариант доработки телевизоров ЦПИЦТ-32-10 («Юность Ц-404», «Шилялис Ц-410»), 2 УСЦТ-51-3

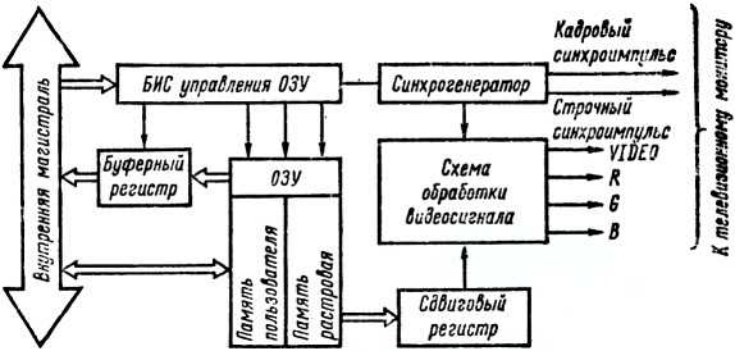



**Рис. 6.11. Структурная схема формирования выходных телевизионных сигналов**

(«Горизонт Ц-355», «Рекорд ВЦ-381») и УМИМЦТ-61-С-2 («Рубин Ц-201, 202, 208») с размерами экрана 32, 51, 61 см по диагонали. Телевизоры с меньшим размером экрана, как показал опыт эксплуатации, не позволяют получать удовлетворительное качество изображения из-за низкой разрешающей способности кинескопов. Для организации входов R, G, B в телевизорах указанных типов необходимо установить плату согласующих усилителей и дополнительный разъем. Принципиальная схема согласующей платы приведена на рис. 6.10. На ней расположены три усилителя каналов R, G, B и усилитель сигнала синхронизации. Переменные резисторы R1, R7, R13 служат для регулировки размаха видеосигнала по каждому цвету, а R23 предназначен для установки амплитуды сигнала синхронизации. При доработке телевизоров УМИМЦТ-61-С-2 емкость *С4* должна быть заменена на диод (типа Д220). Для телевизоров ЦПИЦТ-32 резисторы R6, R12, R18 должны быть соединены перемычками. Номера контактов на разъемах рис. 6.10 соответствуют обозначениям на принципиальных схемах вышеуказанных телевизоров.

Возможны два варианта доработки телевизора. В первом из них, превращающем его только в RGB-монитор, согласующая плата монтируется на кронштейне селектора каналов, магнитофонный выход телевизора используется как дополнительный разъем для организации R-, G-, B-входов и сигнала «видео», а модули видеоусилителей и радиоканала телевизора могут быть отключены. Во втором варианте, расширяющем функции телевизора, кроме согласующей платы и дополнительного разъема должны быть установлены

дополнительные реле для переключения режимов работы телевизора (положение этих реле на схеме при работе телевизора в режиме монитора показано символами ×). Их рекомендуется устанавливать вблизи коммутируемых цепей, в разрыв проводников, причем переключатель для отключения звука телевизора используется в качестве переключателя режимов работы телевизор—мониторы (К1).

**Видеоконтроллер БК.** Он обеспечивает сопряжение микроЭВМ и устройства отображения информации и предназначен для вывода на черно-белый или цветной бытовой телевизор алфавитно-цифровой и графической информации. Видеоконтроллер выполняет регенерацию ОЗУ и отображение содержимого растровой памяти на экран черно-белого или цветного телевизора. Он вырабатывает пять выходных сигналов — R, G, B, SYNC, VIDEO, причем первые четыре предназначены для подключения цветного монитора, тогда как сигнал VIDEO представляет собой полный телевизионный сигнал, подаваемый на вход видеосигнала черно-белого телевизора. На рис. 6.11 приведены основные компоненты видеоконтроллера: БИС (К1801 ВП1-037) управления памятью; буферный регистр данных для временного хранения информации, считанной из ОЗУ с последующей выдачей на внутреннюю магистраль; сдвиговый регистр данных для преобразования параллельного кода, снятого с выходов ОЗУ, в последовательный код, сдвигаемый тактовой частотой 6,0 МГц, и передачи его на вход схемы формирования видеосигнала; схема формирования видео- и синхросигнала. Видеоконтроллер формирует изображение в трех режимах: 1) монохромном с высоким разрешением 256х512 точек и двумя градациями яркости; 2) четырехцветном с разрешением 256х256 точек (возможность окрашивания любого пикселя в один из четырех цветов — красный, зеленый, синий, черный); 3) монохромном низкого разрешения 256х256 точек с четырьмя градациями яркости. Объем растровой памяти, необходимой для работы в каждом из этих режимов, составляет 16 Кбайт.

В режиме расширенной памяти, при котором объем ОЗУ пользователя расширяется с 16 до 28 Кбайт, видеоконтроллер может работать с растровой памятью объемом 4 Кбайт, а размер рабочего поля экрана составляет 64х512 точек для работы в режиме 1 и 64х256 точек для работы в режимах 2, 3.

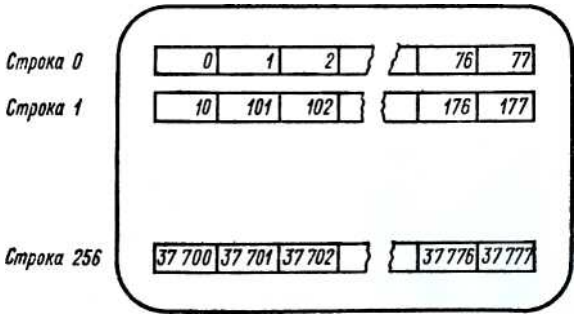



**Рис. 6.12. Расположение байтов растровой памяти на экране**

Процесс отображения растровой памяти состоит в непрерывном циклическом считывании 16-разрядных слов ОЗУ и преобразовании данных в последовательность битов; «ЛОГИЧЕСКАЯ1», записанная в ГОЗУ, вызывает засветку соответствующего пикселя на экране, или, другими словами, одной точке прямоугольного растра изображения соответствует один бит ГОЗУ. Схема обработки видеосигнала из указанной последовательности битов вырабатывает сигналы управления яркостью и интенсивностью цветов, которые после смешения с синхросигналом поступают на выход видеоконтроллера. Таким образом, циклическая регенерация изображения создает эффект статической картины на экране.

В режиме 2 (3) цвет (яркость) каждой точки кодируется соответствующими двумя последовательными битами считываемого слова, при этом 16-разрядное слово, считываемое за один цикл обращения к растровой памяти, отображается горизонтальной строчкой из восьми расположенных рядом пикселей (точек растра). В режиме 1 старший и младший байты отображаются последовательно и каждая точка имеет вдвое меньший размер по горизонтали. На рис. 6.12 приведено расположение байтов растровой памяти на экране устройства отображения. В монохромном режиме 1 первый байт, имеющий нулевой относительный адрес, располагается в верхнем левом углу экрана, следующий байт (адрес 100) — на строку ниже. В левом нижнем углу отображается байт с адресом 37700, следующие 256 байтов с адресами 1 ... 37701 располагаются во втором столбце (размером 8x256 точек) и т. д. Последние 256 байтов, имеющие относительные адреса 77 ... 37777, соответствуют крайнему

правому столбцу. Всего в режиме 1 отображаются 64 столбца по 8 точек в каждом. В режиме 2 столбцов в два раза меньше, так как каждая точка кодируется соответствующими парами последовательно расположенных битов определенных ячеек растровой памяти (например, левый верхний пиксель изображения кодируется младшими битами — нулевым и первым — байта с адресом 0).

Видеоконтроллер непрерывно отображает всю область растровой памяти, располагая ее с верхнего левого угла экрана, причем выбор адреса байта, с которого начинается отображение, определяется значением регистра смещения (см. п. 6.2). Аппаратно реализованный сдвиг растровой памяти в видеоконтроллере (с использованием регистра смещения) позволяет значительно ускорить обработку изображения.

Видеоконтроллер не имеет специальных средств для отображения символьной информации. Текст на экране устройства отображения формируется с помощью процессора. По коду (КОИ-8) символа процессор извлекает из знакогенератора графическое представление этого символа и записывает его в соответствующее место растровой памяти. Знакогенератор, с которым работает процессор, располагается в системном ПЗУ микроЭВМ.

Максимальное число строк (символов в строке), которое можно разместить на экране устройства отображения БК, определяется размером прямоугольного точечного растра (матрицы символа), отведенного под отображение одного символа. В БК для формирования символов используется матрица символа форматом 10x8 точек. Поэтому на экране можно отобразить до 25 строчек по 64 и 32 символа в строке в зависимости от режима формирования изображения (64 для режима 1, 32 для режимов 2, 3); из них 24 — информационные строки, предназначенные для ввода—вывода пользовательских программ и (или) данных, а одна — самая верхняя (нулевая) — служебная и предназначена для вывода служебной информации пользователя, а также для индикации режимов формирования информации на экране устройства отображения.

По горизонтали и по вертикали матрицы символов расположены вплотную (рис. 6.13). В каждой матрице начальные строка и столбец, а также две нижних строки и два крайних правых столбца точек не участвуют в отображении символа, благодаря чему на экране образуются промежутки как между символами

одной строки, так и между строками. Таким образом, в качестве базовой матрицы для основного набора символов в БК используется матрица форматом 7x5 точек. Исключение составляют некоторые символы строчных букв, выходящие за пределы базовой матрицы, а также часть графических символов.

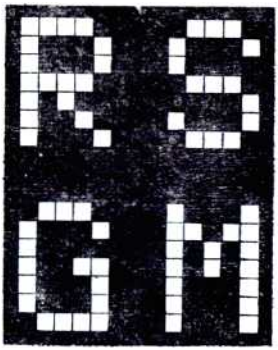


**Рис. 6.13. Изображение символов R, S, G, M на экране ЭЛТ**

**Команды драйвера устройства отображения.** Формирование графической и алфавитно-цифровой информации в трех основных режимах работы видеоконтроллера осуществляется программой — драйвером устройства отображения с помощью девяти команд.

**Инициализация драйвера.**

Команда: EMT 14.

Данная команда не имеет входных параметров и обеспечивает инициализацию драйвера устройства отображения, сброс его ячеек в исходное состояние, установку вектора прерывания, очистку экрана. Команда не сохраняет содержимое системных регистров R0—R4 и не устанавливает стек в исходное состояние.

**Передача кодов драйверу.**

Команда: EMT 16.

Команда обеспечивает передачу кода символа драйверу устройства отображения для его обработки в соответствии с назначением. Входной параметр команды — регистр R0, в младшем байте которого должен находиться код символа.

**Формирование строки символов.**

Команда: EMT 20.

Входные параметры: регистры R1 и R2.

Команда осуществляет передачу последовательности кодов символов, находящихся в области пользовательского ОЗУ, драйверу устройства отображения. Адрес расположения строки символов в области ОЗУ — это содержимое регистра R1.

Команда прекращает свою работу при выполнении одного из двух ограничивающих условий: 1) либо драйверу передано указанное в младшем байте регистра R2 число символов строки; 2) либо при передаче встретился код символа-ограничителя, помещенный в старший байт регистра R2. Отметим, что если содержимое младшего байта регистра R2 равно нулю, то длина строки принимается равной $200000_8$.

После завершения передачи кодов в регистре R1 хранится адрес байта, следующий за последним переданным байтом, а в регистре R2 —

разница между заданным и переданным числом символов строки.

**Формирование символа в служебной строке.**

Команда: EMT 22.

Входные параметры: регистры R0 и R1.

Команда осуществляет запись символа, код которого предварительно помещается в регистр R0, в указанную в регистре R1 позицию служебной строки.

Формирование символа производится в соответствии с режимами, установленными на момент задания команды.

**Установка координат курсора.**

Команда: EMT 24.

Входные параметры: регистры R1 и R2.

Команда производит установку курсора в задаваемую в регистрах R1 и R2 позицию на экране устройства отображения. При этом содержимое регистра R1 указывает положение курсора по оси X, а регистра R2 — по оси Y.

**Определение координат курсора.**

Команда: EMT 26.

Выходные данные: регистры R1 и R2.

Команда определяет положение курсора на экране, осуществляя съем и запись в регистры R1 и R2 его координат по осям X и Y соответственно.

**Формирование точки.**

Команда: EMT 30.

Входные параметры: регистры R0, R1, R2.

Команда в зависимости от значения регистра R0 формирует (содержимое R0 равно 1) или стирает (содержимое R0 равно 0) на экране устройства отображения одну растровую точку (пиксель), координаты которой по осям X и Y задаются в регистрах R1 и R2 соответственно.

**Формирование вектора.**

Команда: EMT 32.

Входные параметры: регистры R0, R1, R2.

Команда в зависимости от значения регистра R0 формирует (содержимое R0 равно 1) или стирает (содержимое R0 равно 0) на экране устройства отображения вектор, координаты конца которого указаны в регистрах R1 (координата X) и R2 (координата Y), а координаты начальной точки этого вектора определяются либо координатами последней сформированной или стертой (с помощью команды EMT 30) точки, либо координатами конца предыдущего построенного или устраненного вектора.

В случае, если координаты конечной точки задаваемого вектора превышают допустимые размеры рабочего поля экрана по осям координат, то драйвер запоминает эти координаты, но формирует только ту часть вектора, которая помещается в информационном поле экрана.

**Чтение слова состояния устройства отображения.**

Команда: EMT 34.

Выходные данные: регистр R0.

Команда производит чтение слова состояния устройства отображения информации, отражающее текущее состояние устройства.

Под состоянием устройства отображения понимается совокупность режимов, в которых оно находится. При этом единица свидетельствует о включенном состоянии данного режима, нуль — о выключенном.

Формат слова состояния приведен в табл. 6.6.

Таблица 6.6

**Формат слова состояния**

| Номер разряда | Соответствующий режим |
|---|---|
| 0 | «32 символа в строке» |
| 1 | Инверсия фона |
| 2 | Расширенная память |
| 3 | Русский алфавит |
| 4 | Подчеркивание символа |
| 5 | Инверсия символа |
| 6 | Индикация СУ |
| 7 | Блокировка редактирования |
| 8 | Режим «ГРАФ» |
| 9 | Запись в режиме «ГРАФ» |
| 10 | Стирание в режиме «ГРАФ» |
| 11 | «32 символа в служебной строке» |
| 12 | Подчеркивание символа в служебной строке |
| 13 | Инверсия символа в служебной строке |
| 14 | Гашение курсора |
| 15 | Не используется |

## 6.5. НАКОПИТЕЛЬ НА МАГНИТНОЙ ЛЕНТЕ

Долговременное хранение пользовательских программ и (или) данных возможно на самых различных периферийных устройствах. Наиболее распространены устройства, в которых для записи и хранения информации применяют магнитные носители (магнитные ленты и магнитные диски различных конструкций). Обычно эти устройства из-за своей сложности имеют высокую стоимость, поэтому с появлением дешевых бытовых персональных компьютеров для долговременного хранения информации в основном используются кассетные магнитные ленты.

В БК в качестве накопителя на магнитной ленте используется бытовой кассетный магнитофон, а в качестве физической среды для хранения информации — кассеты типа МК60, МК90 или любые другие.

**Контроллер накопителя на магнитной ленте.** Подключение кассетного магнитофона к БК и управление его работой при записи и чтении информации осуществляется с помощью программно-аппаратного контроллера накопителя на магнитной

ленте.

Контроллер разработан с учетом требований устойчивой и надежной работы канала чтения — записи информации на магнитную ленту, а также возможности использования в качестве накопителей самых дешевых магнитофонов-диктофонов (типа «Электроника-302), имеющих только разъем с микрофонным входом. На этом же разъеме у магнитофона имеется вход для управления протяжкой магнитной ленты путем размыкания цепи питания двигателя с помощью переключателя, смонтированного на микрофоне.

Аппаратная часть контроллера предназначена для преобразования уровней входных и выходных сигналов и включает буферный усилитель с фильтрами верхних и нижних частот (для записи информации), компаратор, преобразующий входной сигнал в цифровую форму (для чтения информации) и реле включения двигателя магнитофона.

Программная часть контроллера — это драйвер, который формирует сигналы для записи на магнитофон, обрабатывает цифровые сигналы после компаратора, выдает команды для управления двигателем магнитофона.

Обмен информацией с магнитофоном при ее чтении или записи осуществляется со скоростью 1200 бит/с.

**Формат записи на магнитную ленту.** Для записи информации на магнитную ленту используется разновидность метода широтно-импульсной модуляции с побитовой синхронизацией. Плотность записи информации на магнитной ленте при скорости обмена 1200 бит/с составляет 25 бит/мм, а объем информации, записанной на одной кассете типа МК60, может достигать 0,5 Мбайт.

Информация (файл) на магнитной ленте имеет следующую структуру: в начале файла помещается настроечная последовательность импульсов с длительностью, равной длительности «ЛОГИЧЕСКОЙ 1», заканчивающаяся маркером, далее размещается оглавление файла (20 байтов), за которым следует информационная часть файла, а в конце файла помещаются два байта контрольной суммы.

Формат оглавления файла имеет вид: первые 2 байта — адрес, начиная с которого информационная часть файла будет записана в ОЗУ; следующие 2 байта — длина информационной части файла в байтах и последние 16 байтов—имя файла. За каждым битом информации на магнитной ленте следует синхроимпульс.

Настроечная последовательность предназначена для стабилизации переходных процессов, возникающих в начале записи и используется при

чтении файла для автоматической настройки на частоту записи, с которой был записан данный файл. Таким образом, драйвер магнитофона позволяет читать файлы, записанные на магнитную ленту на других ЭВМ с различной скоростью.

**Команды драйвера магнитофона.** Драйвер магнитофона позволяет проводить запись файла (программы или данных) на магнитную ленту из указанной области оперативной памяти, чтение файла с магнитной ленты в заданную область ОЗУ и фиктивное чтение, при котором чтение файла не проводится, а происходит лишь поиск конца указанного файла. При записи и чтении файла двигатель магнитофона автоматически

Таблица 6.7

**Формат блока параметров**

| Номер разряда | Соответствующий режим |
|---|---|
| 0—1 | «Команда», «Ответ» |
| 2—3 | Адрес файла для записи или чтения |
| 4—5 | Длина файла для записи |
| 6—21 | Имя файла для записи или чтения |
| 22—23 | Адрес текущего файла |
| 24—25 | Длина » » |
| 26—41 | Имя » » |

запускается перед началом записи или чтения и останавливается по завершении операции. Помимо перечисленных операций драйвер позволяет осуществлять запуск двигателя и его останов по соответствующей команде.

Обращение к драйверу магнитофона производится по команде EMT 36 с набором параметров, размещенных в блоке параметров.

Команда: EMT 36.

Входные параметры: регистр R1.

Блок параметров формат которого приведен в табл. 6.7, занимает 21 слово и может быть размещен в произвольном месте ОЗУ.

В младший байт нулевого слова заносится однобитная команда, которая должна выполняться, а в старшем байте этого слова формируется результат выполнения команды. Назначение разрядов нулевого слова приведено в таблице 6.8.

Для выполнения требуемой операции управления работой магнитофона необходимо предварительно занести в блок параметров информацию, адрес блока параметров поместить в регистр R1, после чего задать команду EMT 36. Результат выполнения операции запишется в байт «Ответ» нулевого слова блока параметров. Для останова или пуска

двигателя магнитофона необходимо записать в байт «команда» нулевого слова нуль или единицу соответственно. Содержимое остальных полей блока параметров не влияет на выполнение этих операций. При записи файла на магнитную ленту необходимо учитывать, что запись всегда производится с того места, которое к моменту записи оказалось под головкой магнитофона. Поэтому для записи файла необходимо вначале перемотать магнитную ленту к тому месту, с которого будет располагаться файл. Для записи необходимо занести в байт «Команда» константу 2, в байты 2—3 — адрес файла в ОЗУ, в байты 4—5 — длину файла в байтах, в байты 6—21 —имя записываемого файла. Для чтение файла с магнитной ленты необходимо вначале перемотать ленту к месту предполагаемого расположения файла и занести в байт «Команда» константу 3, в байты 2—3 — адрес области ОЗУ, в которую прочитанный файл должен быть записан, в байты 6—25 — имя читаемого файла.

Таблица 6.8

**Назначение разрядов нулевого слова**

| Номер разряда | Назначение |
|---|---|
| 0 | Останов двигателя |
| 1 | Запуск двигателя |
| 2 | Запись файла |
| 3 | Чтение файла |
| 4 | Фиктивное чтение файла |
| 5—7 | Не используются |
| 8 | Команда завершена без ошибки |
| 9 | Имя текущего файла не совпадает с заданным (при чтении) |
| 10 | Ошибка контрольной суммы |
| 11 | Останов по вмешательству оператора |
| 12—15 | Не используются |

Выполнение операции чтения начинается с поиска файла на магнитной ленте по заданному имени. Для этого производится чтение оглавления текущего файла и сравнение его имени с заданным. Если имя текущего файла не совпадает с заданным, то оно помещается в байты 26—41 блока параметров, формируется соответствующий ответ (в байт «Ответ» записывается единица) и управление возвращается вызвавшей программе. Для продолжения поиска нужного файла необходимо вновь передать управление драйверу магнитофона без изменения блока параметров.

При совпадении имен производится чтение файла и запись его в указанную область ОЗУ. Если в байтах 2—3 был задан нулевой

адрес, то запись файла производится в ОЗУ по адресу, прочитанному в оглавлении файла.

После завершения чтения драйвер сравнивает подсчитанную контрольную сумму с контрольной суммой, записанной в конце файла на магнитной ленте. При их несовпадении в байт «Ответ» заносится признак ошибки и управление возвращается вызвавшей программе.

Операция фиктивного чтения выполняется так же как и операция чтения, за исключением того, что не происходит записи читаемого файла в ОЗУ и не производится подсчет контрольной суммы. Операцию целесообразно применять для просмотра содержимого магнитной ленты и поиска конца заданного файла перед записью нового. Операции записи, чтения или фиктивного чтения можно прервать путем нажатия клавиши СТОП на клавиатуре БК. При этом произойдет останов двигателя магнитофона, в байт «Ответ» поместится константа 4 и управление будет возвращено вызвавшей программе.

## 6.6. ПОСЛЕДОВАТЕЛЬНЫЙ КАНАЛ

Массовое применение средств вычислительной техники невозможно без создания программно-аппаратных средств для объединения различных внешних устройств (устройств хранения и отображения информации, измерительных приборов, устройств связи с объектом, аппаратуры передачи данных и т. п.) и непосредственно ЭВМ в единую систему. Эта задача возлагается на унифицированные системы сопряжения — интерфейсы.

В зависимости от вида соединения компонентов системы (магистрального, радиального, цепочечного, смешанного), способа передачи информации (параллельного, последовательного, параллельно-последовательного), принципа обмена информацией (асинхронного, синхронного) и режима передачи информации (двусторонняя одновременная передача, двусторонняя поочередная передача, односторонняя передача) интерфейсы подразделяются на классы.

В компьютере «Электроника БК-0010Ш» предусмотрен интерфейс ИРПС (для радиального подключения внешних устройств с последовательной передачей информации), обеспечивающий единые способы обмена информацией для различных устройств ввода—вывода (старт—стопных, с буфером или без него) при работе с микроЭВМ.

Интерфейс ИРПС позволяет осуществлять асинхронную передачу постоянным током (токовая петля) по четырехпроводной дуплексной линии связи, два провода которой предназначены для приема и передачи информации, а два других — для приема и передачи сигналов готовности.

Управление обменом в БК производится драйвером последовательного (телеграфного) канала, который обеспечивает передачу информации на линию в соответствии со стандартным протоколом ИРПС и заданной скоростью 50 ... 9600 бит/с, а также прием и дешифрацию информации, поступающей с линии.

Драйвер последовательного канала имеет команды, позволяющие организовать обмен массивами между БК и внешним устройством. При этом передача и прием осуществляются побайтно.

Необходимо отметить, что в любой момент времени обмен по последовательному каналу может быть прерван с помощью клавиши БК СТОП, нажатие которой вызовет немаскируемое прерывание по вектору 4.

**Команды драйвера последовательного канала. Инициализация драйвера последовательного канала.**

Команда: EMT 40.

Входные параметры: регистр R0.

Команда позволяет установить требуемую скорость обмена информацией по линии. При этом выбор скорости осуществляется по номеру, предварительно записанному в регистр R0. Набор возможных скоростей обмена и соответствующие им номера приведены ниже:

| Номер скорости | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | $10_8$ |
|---|---|---|---|---|---|---|---|---|---|
| Скорость обмена, бит/с..... | 9600 | 4800 | 2400 | 1200 | 600 | 300 | 150 | 75 | 50 |

Отметим, что при включении питания автоматически устанавливается максимальная скорость обмена — 9600 бит/с.

**Передача байта на линию.**

Команда: EMT 42.

Входные параметры: регистр R0.

Команда обеспечивает передачу кода, помещенного в младший байт регистра R0 на линию, с установленной в драйвере последовательного канала скоростью.

Передача байта начинается с младшего бита, которому предшествует стартовый бит. Перед передачей байта проверяется готовность приемной стороны к его приему. При отсутствии сигнала готовности драйвер

переходит в цикл его опроса с линии, из которого выходит либо при появлении сигнала готовности, либо при нажатии клавиши СТОП. На время передачи байта прерывания от внешних устройств маскируются.

**Прием байта с линии.**

Команда: EMT 44.

Выходные данные: регистр R0.

Команда обеспечивает прием байта с линии и запись его в младший байт регистра R0. При этом скорость передачи на линии должна совпадать с той, которая установлена в драйвере последовательного канала.

По команде EMT 44 драйвер выдает на линию сигнал готовности к приему байта и переходит в режим ожидания стартового бита. После приема байта сигнал готовности сбрасывается.

Во время ожидания байта с линии прерывание от внешних устройств разрешено; при поступлении стартового сигнала с линии прерывания от внешних устройств маскируются.

**Передача массива по линии.**

Команда: EMT 46.

Входные параметры: регистры R1, R2.

Эта команда обеспечивает передачу массива байтов, расположенного в ОЗУ по адресу, указанному в регистре R1, на линию. Длина передаваемого массива указывается в регистре R2.

После завершения передачи содержимое R2 равно нулю, а в R1 хранится адрес байта, находящегося за последним переданным байтом.

**Прием массива с линии.**

Команда: EMT 50.

Входные данные: регистры R1, R2,

Эта команда обеспечивает прием массива байтов с линии и запись его в ОЗУ, начиная с адреса, предварительно указанного в регистре R1.

После завершения приема в регистре R2 хранится длина принятого массива в байтах, а в R1 — адрес байта, находящегося за последним принятым байтом.

## 6.7. ПОРТ ВВОДА-ВЫВОДА

Программируемый порт ввода—вывода предназначен для подключения периферийных устройств пользователя, работой которых можно управлять с помощью микроЭВМ.

Порт имеет два 16-разрядных регистра, входной и выходной, через которые можно передавать и читать управляющие сигналы на контактах внешнего разъема. Оба регистра имеют на магистрали один и тот же адрес — 177714, поэтому отсутствует возможность прочитать содержимое выходного регистра. В связи с этим в системной области ПЗУ микроЭВМ зарезервирована ячейка с

адресом 256, в которую рекомендуется заносить информацию, записываемую в выходной регистр при работе с портом. Таким образом, ячейка 256 будет являться копией выходного регистра порта.

Работа с портом ввода—вывода описана в п. 3.5 (функция FP), а организация регистра ввода—вывода порта дана в п. 6.2.

## 6.8. СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

В состав системного программного обеспечения БК входят: монитор и драйверы клавиатуры, устройства отображения, магнитофона и последовательного (телеграфного) канала.

Системное программное обеспечение реализовано «в кремнии», т. е. записано в БИС системного ПЗУ К1801РЕ1, занимающем адресное пространство 100000—177777. Применение в микроЭВМ «кремниевого» программного обеспечения позволяет достичь надежности, характерной для аппаратных схем, обеспечить простоту обслуживания и постоянство интерфейса между пользователем и микроЭВМ, и в то же время сохранить гибкость, присущую программным средствам.

Монитор предназначен для инициализации микроЭВМ при включении питания (либо при перезапуске процессора с помощью

Таблица 6.9

**Адреса резервных входов команды EMT**

| Команда | Адрес на передачу управления | Команда | Адрес на передачу управления | Команда | Адрес на передачу управления | Команда | Адрес на передачу управления |
|---|---|---|---|---|---|---|---|
| EMT 52 | 160000 | EMT 62 | 160020 | EMT 72 | 160040 | EMT 102 | 160060 |
| EMT 54 | 160004 | EMT 64 | 160024 | EMT 74 | 160044 | EMT 104 | 160064 |
| EMT 56 | 160010 | EMT 66 | 160030 | EMT 76 | 160050 | EMT 106 | 160070 |
| EMT 60 | 160014 | EMT 70 | 160034 | EMT 100 | 160054 | EMT 110 | 160074 |

тумблера) и запуска рабочей программы, размещенной в сменном ПЗУ с начальным адресом $120000_8$ (там располагается интерпретатор ФОКАЛа в БК-0010 или БЕЙСИКа в БК-0010-01). Кроме того, монитор предоставляет пользователю возможность

загрузки требуемой программы с магнитной ленты или по телеграфной линии.

Кроме перечисленных программ системное ПЗУ включает в себя область связи, содержащую адреса входов в драйверы внешних устройств, а также адреса переходов на резервное сменное (третье) ПЗУ и диспетчер команд EMT.

Диспетчер команды EMT обеспечивает обработку команды EMT и передачу управления на драйверы по требуемому входу, в зависимости от аргумента команды. В диспетчере команды EMT предусмотрены 16 резервных входов, по адресам которых в четвертом сменном ПЗУ могут быть записаны требуемые пользователю программы (табл. 6.9).

**Адреса векторов прерывания.** Обработка прерываний в микроЭВМ производится по векторам, размещенным в так называемой системной области ОЗУ с адресами 0—364. Адреса векторов прерываний приведены в табл. 6.10.

**Монитор.** Он предназначен для инициализации системы при включении питания микроЭВМ и запуска системной программы, расположенной в системном ПЗУ. Кроме этого, монитор располагает некоторыми диалоговыми средствами, представляющими пользователю возможность загрузить необходимую программу с магнитной ленты или с телеграфной линии и запустить ее.

При инициализации системы в указатель стека заносится адрес $1000_8$, устанавливаются вектора прерываний по команде EMT, рабочие ячейки управляющих программ и скорость обмена по телеграфной линии (9600 бит/с), очищаются экран и пользовательский порт (адрес $177714_8$), устанавливается в исходное состояние регистр управления системными внешними устройствами, через который подключены магнитофон и телеграфная линия.

После инициализации системы управление передается рабочей программе, которая должна быть расположена во втором сменном ПЗУ, по адресу 120000. Такой программой может быть любая установленная пользователем программа. Конкретным примером является интерпретатор языка ФОКАЛ. Передача управления этой программе осуществляется с помощью команды JSR PC, @#120000, в результате выполнения которой адрес возврата запоминается в стеке. Таким образом сохраняется возможность вернуться в монитор, который в этом случае переходит в диалоговой режим. Если ПЗУ с рабочей программой отсутствует,

то при выполнении команды передачи управления возникнет прерывание по зависанию в системной магистрали и управление будет передано монитору, который перейдет в режим диалога с пользователем. Признаком входа в диалоговый режим монитора служит появление знака вопроса на экране. Для перехода в монитор из ФОКАЛа пользователь должен ввести оператор
P [ASS] M [ONITOR].

Таблица 6.10

**Таблица векторов прерывания**

| Адрес вектора прерывания | Источник прерывания |
|---|---|
| 4 | Зависание при передаче данных по системной магистрали или при нажатии клавиши СТОП |
| 10 | Резервная машинная команда |
| 14 | Прерывание по Т-разряду слова состояния процессора |
| 20 | Прерывание по команде IOT |
| 24 | Авария сетевого питания |
| 30 | Прерывание по команде EMT |
| 34 | Прерывание по команде TRAP |
| 60 | Прерывание от клавиатуры |
| 100 | Сигнал 1RQ2 (прерывание от устройства пользователя, подключенного к порту ввода—вывода) |
| 264 | Прерывание от клавиатуры (код нижнего регистра) |
| 360 * | То же от приемника последовательного канала |
| 364 * | То же от передатчика последовательного канала |

*Справедливо только для БК-0010Ш.

В диалоговом режиме монитор выполняет ряд команд, которые дают возможность загрузить программу или данные в заданную область ОЗУ с магнитной ленты или с телеграфной линии и запустить программу с заданного адреса. Монитор допускает использование команд как в полном, так и в сокращенном формате. Далее по тексту сокращенный формат команд указывается, как обычно, с использованием квадратных скобок, а подсказки, которые выводит на экран монитор, подчеркнуты.

**Загрузка с магнитной ленты.**

M [АГ] <адрес загрузки> «ВВОД»

ИМЯ? <имя файла> «ВВОД»

Адрес загрузки задается в виде восьмеричного числа и определяет адрес ОЗУ, куда необходимо считать файл. Если адрес загрузки не указан или задан равным нулю, то загрузка файла производится по адресу, содержащемуся в оглавлении

загружаемого файла; если допущена ошибка при наборе адреса, то необходимо ввести несколько нулей или пробел, после чего ввести адрес заново.

В ответ на появление подсказки «ИМЯ»? необходимо ввести имя считываемого файла. Имя должно содержать не более 16 символов. Если имя не указано, то производится загрузка файла с именем, содержащим 16 пробелов; если при вводе имени допущена ошибка, то ее можно исправить с помощью клавиши забоя <-/-, при нажатии которой стирается последний набранный символ. При поиске заданного файла на экране будут распечатаны имена файлов, просмотренных во время поиска. Если обнаружен требуемый файл, то имя его не распечатывается, а файл читается в заданную область ОЗУ. После окончания чтения на экране появится знак вопроса. Если файл был считан с ошибкой, то на экране выдается сообщение «ошибка».

После загрузки файла в ячейке 264 содержится адрес ОЗУ, начиная с которого загружен файл, а в ячейке 266 — длина файла в байтах.

**Загрузка с телеграфной линии.**

Л [ИНИЯ] <адрес загрузки> «ВВОД»

Адрес загрузки задается в виде восьмеричного числа и определяет адрес ОЗУ, куда должен быть загружен файл с телеграфной линии. Если адрес не задан или равен нулю, то загрузка производится по адресу, указанному в оглавлении файла. Исправление ошибок при вводе адреса осуществляется таким же образом, как и в предыдущей команде.

Перед загрузкой файла монитор производит инициализацию обмена. Этот процесс осуществляется путем передачи произвольного байта по линии в микроЭВМ, содержащую загружаемый файл и приема этого же байта с линии. После этого монитор переходит в режим приема оглавления файла, а затем и самого файла. Оглавление файла должно включать 4 байта, причем первые два должны содержать адрес загрузки файла, а следующие два — длину файла в байтах, не включая оглавление. Адрес загрузки помещается в ячейку 264, а длина файла — в ячейку 266. После загрузки файла на экране появляется знак вопроса. Загрузка осуществляется со скоростью 9600 бит/с.

**Запуск программы.**

С [ТАРТ] <стартовый адрес> «ВВОД»

По этой команде производится запуск программы с указанного адреса. Управление программе передается с помощью команды

JSR PC, <АДРЕС>, поэтому из вызванной программы можно вернуть управление монитору без перезапуска системы. Если адрес запуска не указан, то происходит запуск по адресу, содержащемуся в ячейке 264.

**Передача управления на пользовательское (третье) ПЗУ.**

П [УСК] «ВВОД»

По этой команде управление передается на ПЗУ, расположенное по адресу 140000.

**Запуск тестов.** Если в составе микроЭВМ есть ПЗУ с тестами, расположенное по адресу 160000, то с помощью команды Т [ЕСТ] «ВВОД» можно передать управление на тестовые программы; адрес запуска тестов — 160100.

**Перезапуск интерпретатора ФОКАЛа.** Для того, чтобы осуществить перезапуск рабочей программы, не выключая питания микроЭВМ, необходимо ввести команду, первым символом которой должна быть одна из букв латинского алфавита от *A* до *K*, например, F [OCAL] «ВВОД». По этой команде произойдет перезапуск интерпретатора и управление будет передано на ПЗУ по адресу 120000.

Монитор, кроме того, представляет средства проверки аппаратуры БК — директивы отладки, и позволяет программировать в машинных кодах. Список машинных команд БК-0010 приведен в приложении 4.

## 6.9. БЫТОВОЙ КОМПЬЮТЕР НА ПРОИЗВОДСТВЕ, В ШКОЛЕ И ДОМА

**Прикладное программное обеспечение.** Область применения персональных компьютеров определяется как их архитектурой, аппаратными возможностями, так и наличием развитого программного обеспечения и инструментальных средств его разработки, обеспечивающих развитый человеко-машинный интерфейс, простое и наглядное управление персональным компьютером.

Когда был налажен выпуск персональных компьютеров «Электроника БК-0010», их программное обеспечение ограничивалось лишь языком высокого уровня ФОКАЛ, интерпретатор которого размещался в одной микросхеме съемного ПЗУ, и несколькими игровыми, обучающими и научными программами, вполне умещающимися на одной кассете.

За прошедшие годы для БК были разработаны разнообразные

программные средства (более 500 программ). Среди них игровые, учебные, демонстрационные, деловые программы, обучающие и информационно-справочные системы, программы для научных и инженерных расчетов, систем с числовым программным управлением, САПР, программы для управления бытовой техникой, испытательным контрольно-измерительным оборудованием, технологическими объектами и т. д. [6, 7, 8, 14, 18, 20, 26].

Приведем примеры наиболее распространенных программных средств для БК. К ним относятся:

1) семейство программ ФОКОД и подсистема интерфейса внешних функций (ИВФ), устраняющих один из основных недостатков реализации ФОКАЛа на БК (медленное исполнение программ, позволяющих вызывать из ФОКАЛа подпрограммы, написанные в машинных кодах, и, тем самым, сочетать простоту программирования на ФОКАЛе с высокой скоростью, характерной для использования машинного языка);

2) многочисленные инструментальные средства программирования пользовательских задач, включающие микроотладчик и его усовершенствованные модификации для ускорения этапа отладки программ в машинных кодах; дисассемблеры, обеспечивающие возможность преобразования и визуализации на экране телевизора машинных команд в их символьном представлении на языке Ассемблера; текстовые и графические экранные редакторы; библиотеку стандартных программ для научных расчетов, пакет программ обработки литерных величин; трансляторы для широко распространенных языков программирования — Ассемблера, Лого, Форта, БЕЙСИКа и т. п.;

3) компилятор БЕЙСИКа с элементами интерпретации (аналог версии BASIC-MSX), учитывающий графические и звуковые возможности БК, размещенный в трех микросхемах съемного ПЗУ БК-0010-01;

4) специализированный Т-язык, ориентированный на педагогов и позволяющий вследствие таких своих полезных свойств как текстовая ориентация языка, сменных алфавитов, разнообразие форм представления информации (речь, музыка, графика), ориентация на учебные операции, простота подготовки и использования программ и т. п., значительно сократить затраты на разработку программированных игр, демонстрационных программ, тренажеров, программированных имитаторов, задачников, уроков

и т. д.;

5) пакет демонстрационных, обучающих и игровых программ «Рига» на основе Т-языка;

6) система машинной геометрии и графики АЛГРАФ на базе специализированного языка описания геометрической информации АЛГРАФ-Р, обеспечивающая построение различных автоматизированных систем проектирования, технологической подготовки производства, обучения;

7) кросс-средства для программирования пользовательских задач БК на языке Паскаль в операционной среде RT-11 на ДВК-2М, позволяющие в некоторых случаях компенсировать ограниченное быстродействие интерпретатора ФОКАЛа, компилятора БЕЙСИКа и расширить сервисные возможности пользователей; для физической реализации переноса программ разработан программноаппаратный интерфейс связи ДВК-2М с бытовым кассетным магнитофоном;

8) многочисленные системы вычислительного и невычислительного назначения для решения конкретных задач в различных областях народного хозяйства (системы сбора данных о распределении энергии луча в электронно-лучевой установке и управления процессом высокотемпературной пайки крупногабаритных изделий в индукционных печах, программно-аппаратный комплекс с применением БК и языка ФОКАЛ для автоматизации процесса наплавления блочного кварцевого стекла газоплазменным методом и управления производством кварцевых блоков высокого качества с жестким допуском по наружному диаметру и т. д.);

9) программы для научных и инженерных расчетов вычисления коэффициентов интерполирующего полинома, решения системы линейных уравнений методом наименьших квадратов, интегрирования и дифференцирования аналитически и таблично заданных функций, статистических расчетов, решения системы дифференциальных уравнений, обращения матриц, графоаналитической обработки данных, специальных расчетов надежности аппаратуры, механических, радиотехнических и электронных приборов и схем, физико-химических и термодинамических констант веществ и т. д.;

10) учебные и обучающие программы по арифметике, геометрии, физике, биологии, истории, географии, иностранному и русскому языку, музыкальной грамоте, основам информатики и вычислительной техники в школе, программированию и т. д.;

11) деловые программы (словари, каталоги, записные книжки, телефонные справочники и т. п.);

12) развлекательные игровые программы.

**Комплекс учебно-вычислительной техники на базе БК (КУВТ-86).** Как уже упоминалось ранее, одна из модификаций БК — «Электроника БК-0010Ш», поставляется в составе комплекса учебно-вычислительной техники (КУВТ-86) — первой отечественной системы компьютерного обучения. Ее появление связано с введением в школьные программы с 1986 г. нового предмета (основ информатики и вычислительной техники) и остро вставшей проблемы создания компьютерных учебных классов на базе серийно выпускаемых бытовых персональных ЭВМ для проведения практических занятий с учащимися.

В настоящее время в различных городах нашей страны существует более 2,5 тыс. таких классов — однородных комплексов технических средств, организованных в локально-вычислительную сеть (ЛВС) радиального типа («звезда») и обслуживаемых единым системным программным обеспечением.

В состав КУВТ-86 входит один диалоговый вычислительный комплекс ДВК-2МШ (рабочее место преподавателя — главная ЭВМ сети) и 12 персональных компьютеров БК-0010Ш (рабочие места учеников — рабочие станции сети).

Рабочее место преподавателя включает одноплатную микроЭВМ НМС 1201.01 с контроллером дисплея и интерфейсами НГМД и печатающего устройства, алфавитно- цифровой дисплей 15ИЭ-00-93, накопители на гибких мини-дисках «Электроника НГМД-6022», матричное печатающее устройство УВВПЧ-30-004, две платы контроллеров телеграфных каналов (КТЛК), вставленные в унифицированный корпус с блоком питания.

Рабочее место ученика состоит из персонального компьютера БК-0010Ш с блоком питания, телевизора для отображения информации и блока связи ИРПС. К нему можно подключить кассетный магнитофон для записи программ на магнитную ленту.

Связь между рабочей станцией и главной ЭВМ осуществляется по последовательному каналу ИРПС. Аппаратную поддержку связи в рабочей станции обеспечивает внешний блок ИРПС, подключенный к магистральному разъему БК. Со стороны ДВК поддержку ЛВС обеспечивают две платы КТЛК, имеющие по шесть каналов последовательного ввода—вывода информации. При данной организации ЛВС КУВТ-86 все функции управления и обмена программами закреплены за главной ЭВМ; накопитель на

ГМД и печатающее устройство — общие для всех рабочих станций.

В системное ПО главной ЭВМ входят программы, обеспечивающие прием—передачу файлов, организацию связи с БК-0010, программы системы управления файлами на внешнем запоминающем устройстве в операционной среде RT-11 (ОС ДВК) и другие стандартные программные средства ОС ДВК.

Системное ПО рабочей станции записано в БИС ПЗУ, содержащих помимо интерпретатора ФОКАЛа или компилятора БЕЙСИКа программные модули поддержки обмена информацией с главной ЭВМ по последовательному каналу.

Сетевые программные средства КУВТ-86 обеспечивают: протоколы начальной загрузки и представления виртуальных устройств ввода—вывода; защиту данных и программ, принадлежащих каждой из рабочих станций друг от друга; использования ресурсов главной ЭВМ рабочими станциями; хранение данных и программ, принадлежащих любой из рабочих станций в архиве; работу с клавиатуры рабочей станции в качестве пользователя главной ЭВМ.

ГЛАВА 7

# ИСПОЛЬЗОВАНИЕ ФОКАЛа В СИСТЕМАХ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ И АВТОМАТИЗАЦИИ ПРОИЗВОДСТВА

Ряд достоинств ФОКАЛа, которые отмечались в гл. 2 (интерактивный режим работы, широкий спектр операторов языка при малом объеме ОЗУ, занимаемом интерпретатором, наличие функций для работы с периферийными устройствами, возможность расширения пользовательской библиотеки функциями, программируемыми на языке Ассемблера и т. д.), обусловили применение языка в системах автоматизации проектирования (САПР), автоматизированных системах технологической подготовки производства (АСТПП), гибких производственных системах (ГПС). В последнее время опубликован ряд работ [2, 7, 23], описывающих использование ФОКАЛа для расширения возможностей интерактивно-графической системы 15УТ-4-017 [21], не только для разработки топологии интегральных микросхем, печатных плат, но и для автоматизации конструирования деталей, технологических процессов их изготовления, подготовки управляющих программ для станков с числовым программным управлением.

## 7.1. АВТОМАТИЗИРОВАННОЕ РАБОЧЕЕ МЕСТО ПРОЕКТИРОВЩИКА

Автоматизированное рабочее место (АРМ) представляет собой комплекс программно-аппаратных средств, обеспечивающих проектировщику оперативный и легкий доступ к ЭВМ, реализацию итерационных циклов проектирования в рамках диалоговых режимов работы, обмен с ЭВМ информацией в графической форме.

Автоматизированные рабочие места предназначены для решения сравнительно сложных инженерных задач и организации эффективного общения пользователя САПР с комплексом технических средств, которые состоят из мини- и (или) микроЭВМ, графических и символьных дисплеев, кодировщиков графической информации, устройств символьного и графического документирования с соответствующим системным и прикладным программным обеспечением. Для ряда АРМ характерен

интерактивный режим работы по вводу, обработке, документированию текстовой и графической информации, что обусловливает еще одно название таких комплексов — интерактивнографические комплексы. АРМ могут быть использованы в многоуровневых иерархических САПР, в составе вычислительных сетей автоматизированного проектирования, в качестве рабочих мест центральных вычислительных комплексов, технологических комплексов для адаптации конструкторского проекта к различному технологическому оборудованию, технологическим маршрутам, а также в качестве инструментальных комплексов для разработки системного и прикладного программного обеспечения подсистем САПР.

## 7.2. ИНТЕРАКТИВНО-ГРАФИЧЕСКАЯ СИСТЕМА (ИГС) «КУЛОН»

Система 15УТ-4-017 (в дальнейшем «Кулон») представляет собой специализированный программно-аппаратный комплекс, предназначенный для решения задач автоматизации разработки изделий электронной техники и в первую очередь для проектирования интегральных микросхем, типовых элементов замены (ТЭЗ), СВЧ-транзисторов и подготовки сопроводительной конструкторской документации.

Как и в любом АРМ, технические средства ИГС «Кулон» группируются вокруг мини-ЭВМ «Электроника- 100-25» и работают под управлением операционной системы (ОС) «Кулон» (рис. 7.1).

Помимо процессора, в состав мини-ЭВМ входят оперативное запоминающее устройство емкостью до 128 К слов, устройства внешней памяти на магнитных дисках (СМ-5400), магнитных лентах (ИЗОТ 5003) и гибких магнитных дисках («Электроника ГМД-70»).

Через интерфейс «Общая шина» к мини-ЭВМ подключаются технические средства рабочих мест (рис. 7.2) — подсистемы отображения и диалога, состоящие из текстовых и графических средств ввода—вывода: кодировщика графической информации ЭМ-719, запоминающего графического дисплея 15 ИГ-160×210-001, устройства управления положением маркера

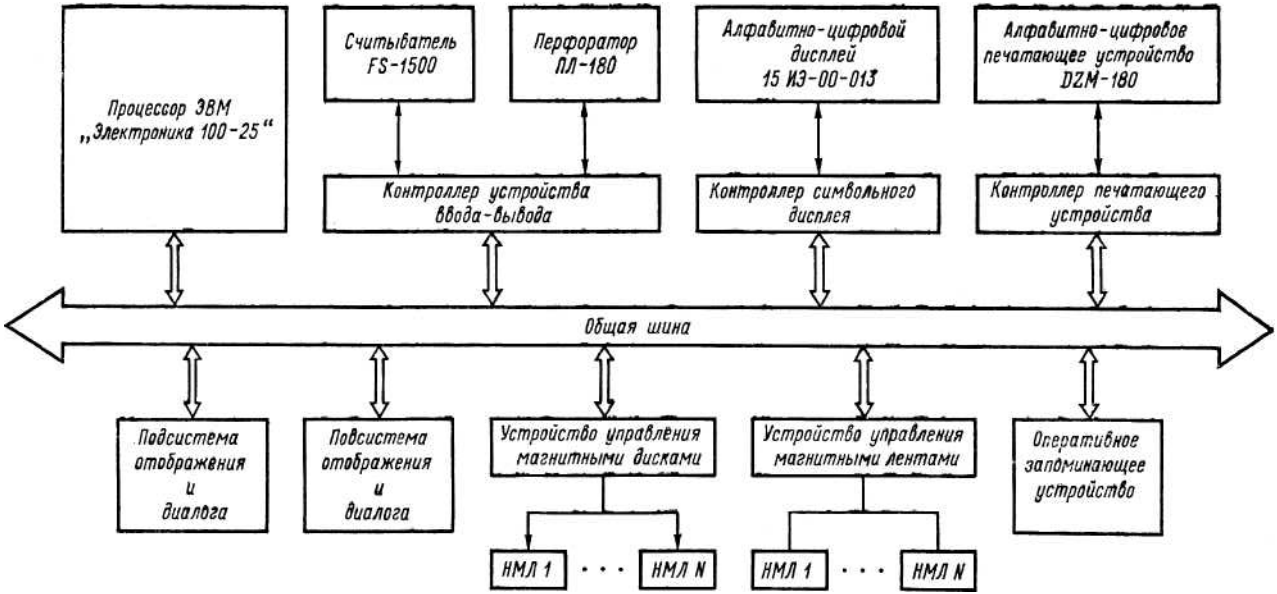



**Рис. 7.1. Технические средства ИГС «Кулон»**

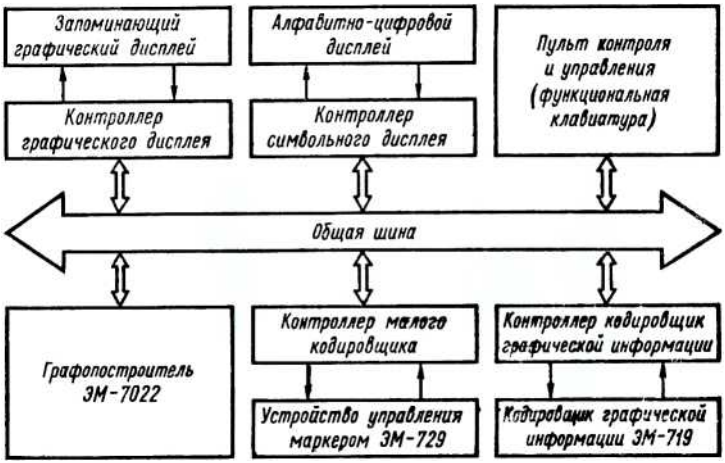



**Рис. 7.2. Технические средства рабочих мест**

ЭМ-729 или малого кодировщика, пульта контроля и управления, автоматических планшетных графопостроителей ЭМ-7022, ЭМ-732, мозаичного АЦПУ DZM-180, символьного дисплея 15 ИЭ-00-013, перфоратора ПЛ-150 и фотосчитывателя FS-1501. Общий вид ИГС «Кулон» показан на рис. 7.3.

Упрощенная структурная схема подсистемы отображения и диалога приведена на рис. 7.4. В ее состав входят интерфейсные блоки *БИ1, БИ2,* электронный блок БЭ для организации работы устройств рабочего места. Интерфейсный блок *БИ1* предназначен для обеспечения операций обмена между «Общей шиной» и подсистемой отображения и диалога. Непосредственно к нему присоединен алфавитно-цифровой дисплей. Остальная аппаратура подключена к *БИ1* через устройство параллельного обмена *БИ2.* Блок *БИ2* преобразует последовательные коды информации в параллельные и осуществляет непосредственное управление графическим дисплеем.

Запоминающий графический дисплей (ЗГД) используется в системе для оперативного отображения графической информации. Он создан на основе запоминающей электронно-лучевой трубки 31ЛН6 и имеет следующие технические характеристики: размеры рабочего поля экрана 160х210 мм, толщина линий не более 0,8 мм, скорость построения линии не менее 100 м/с, время стирания изображения 0,5 с.

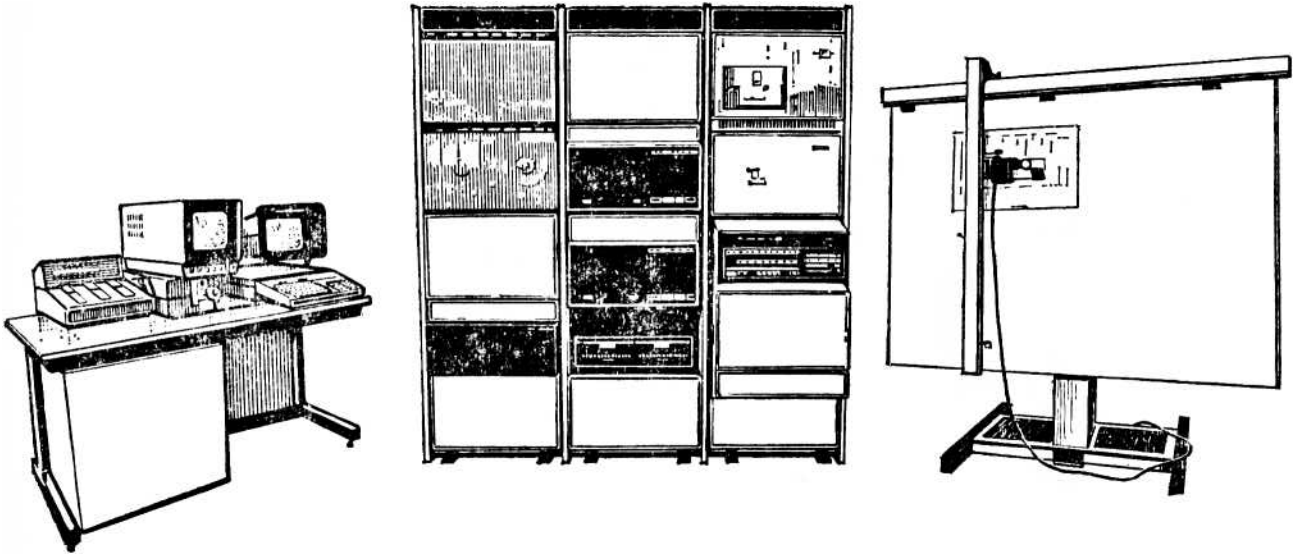


**Рис. 7.3. Общий вид ИГС «Кулон»**

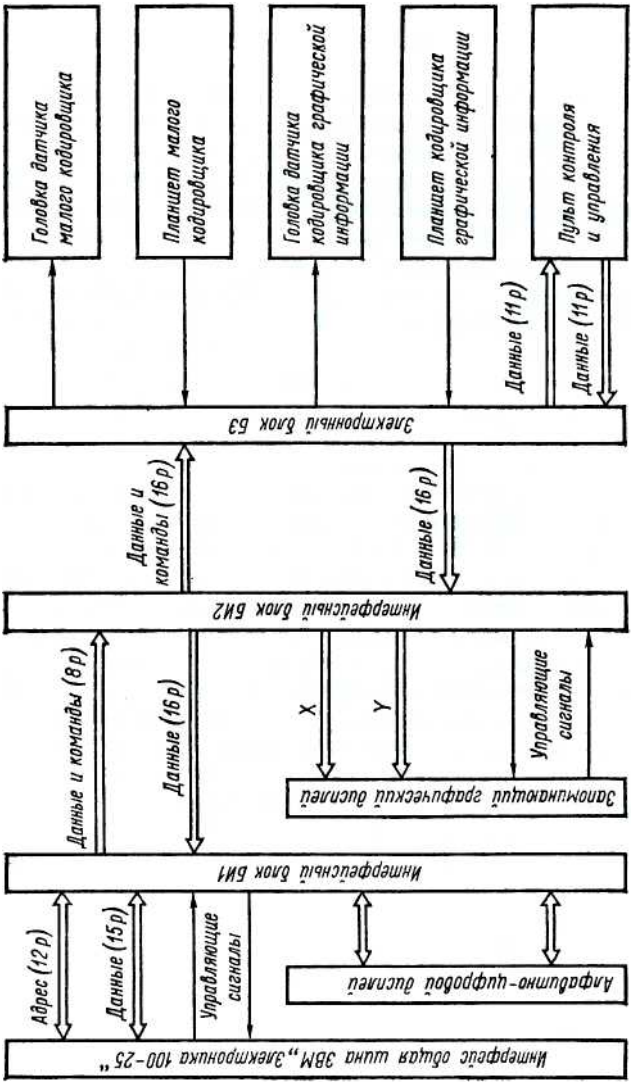



**Рис. 7.4. Упрощенная структурная схема подсистемы отображения и диалога**

Два кодировщика графической информации, входящие в состав подсистемы отображения и диалога, предназначены для полуавтоматического кодирования информации с вводом ее непосредственно в ЭВМ. Они устроены практически одинаково, отличаются друг от друга лишь функциональными возможностями и обслуживаются одной и той же электронной аппаратурой. Кодировщик ЭМ-719 имеет следующие технические данные: размеры рабочего поля планшета 1100х1500 мм, погрешность выдачи координат ± 0,125 мм. Малый кодировщик ЭМ-729 используется для управления маркером ЗГД и имеет рабочее поле 300х300 мм. Рабочее поле ЭМ-729 с помощью масштабных преобразований приводится в однозначное соответствие с рабочим полем экрана ЗГД. Перемещая считывающую головку ЭМ-729 по планшету, пользователь задает тем самым координаты маркера на экране ЗГД и визуально контролирует его положение.

Пульт контроля и управления предназначен для задания команд пользователя в режиме описания и редактирования графической информации и индикации положения считывающей головки ЭМ-719 или ЭМ-729. Его функциональная клавиатура содержит 54 клавиши и используется для задания команд общего управления устройствами подсистемы отображения и диалога, изображением на экране ЗГД, графического редактирования элементов изображения, формирования и управления библиотекой элементов топологии, описания элементов изображения при кодировании.

Графопостроитель ЭМ-7022 подключается к каналу ЭВМ через устройство управления и имеет следующие характеристики: рабочее поле 1200х1600 мм, максимальная скорость черчения 250 мм/с, разрешающая способность по координатам $X$ и $Y$ — 0,1 мм, погрешность вычерчивания ± 0,15 мм, число пишущих инструментов — 5.

Графопостроитель планшетного типа ЭМ-732 управляется от ЭВМ «Электроника-60М» через промежуточный носитель — магнитную ленту. Его основные технические данные: рабочее поле 1600х1200 мм, погрешность вычерчивания ± 0,2 мм, минимальный шаг перемещения 0,025 м, максимальная скорость движения при вычерчивании по каждой координате 800 мм/с, число пишущих инструментов — 4.

## 7.3 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ИГС «КУЛОН»

Операционная система (ОС) «Кулон» — мультипрограммная система с фиксированным числом задач (до четырех), работающая в режиме разделения времени. ОС ориентирована только на стандартную конфигурацию технических средств ИГС и не настраивается на какие-либо другие варианты аппаратуры.

ОС «Кулон» включает комплекс управляющих диспетчерских программ, драйверы внешних устройств, инструментальные средства программирования, а также прикладное программное обеспечение — программы логико-схемотехнического и конструкторско-топологического проектирования.

Комплекс управляющих диспетчерских программ состоит из двух частей — резидентной и нерезидентной, и обеспечивает работу ИГС в режиме разделения времени, обработку многоуровневых программно-аппаратных прерываний, обмен с внешними устройствами, защиту и распределение памяти, службу времени. Кроме того, программы резидентной части ОС обеспечивают ввод и интерпретацию директив пользователя, перезагрузку файлов, формирование управляющих данных для обмена с внешними устройствами, идентификацию и обработку сбоев внешних устройств, выполнение арифметических операций над числами с обычной и удвоенной точностью, операции выделения «окна» над графическими примитивами, вычисление тригонометрических функций.

Инструментальные средства программирования включают двухпроходный транслятор с машинно-ориентированного языка Ассемблера, символические редактор и отладчик для автоматизации подготовки и отладки ассемблерных задач, и расширения системного и прикладного программного обеспечения ИГС.

Пакет прикладных программ логического моделирования и схемотехнического проектирования выполняет следующие основные функции: формирование и ввод символьной информации о принципиальных электрических и логических схемах и ее контроль; автоматическую генерацию по текстовому описанию рисунков схем на запоминающем графическом дисплее; описание эскиза логических и принципиальных электрических схем с помощью кодировщика графической информации с одновременной визуализацией изображения на запоминающем

графическом дисплее; синтез рисунков схем на экране запоминающего графического дисплея с одновременным формированием текстовой информации о типах элементов в схеме и связях между ними; контроль и редактирование текстовой и графической информации о схемах и управление изображением на экране запоминающего графического дисплея; формирование библиотеки параметров логических, схемотехнических и графических моделей типовых элементов и функциональных блоков; форматные преобразования текстовой информации для программ логического моделирования и расчета электрических характеристик; логический анализ функциональных блоков в статическом и динамических режимах, анализ их электрических характеристик; обработку описаний схем и результатов их анализа, подготовку информации и конструкторское документирование текстового и графического представления схем.

Пакет прикладных программ конструкторско-топологического проектирования обеспечивает этап проектирования и изготовления фотошаблонов (ФШ) интегральных микросхем. Функциональное назначение пакета — формирование топологии заданного технологического слоя, разбитой с учетом схемотехнического и технологического базиса на элементарные фигуры, служащие исходной информацией для различных исполнительных автоматов. Другими словами, пакет предназначен для расчета и записи на магнитную ленту данных — управляющих программ, соответствующих исходному описанию геометрии ФШ, содержанию библиотек топологических фрагментов и параметрам, задаваемым пользователем в процессе управления расчетом. По данным, сформированным на этом этапе, с помощью генераторов изображения геометрия интегральной микросхемы реализуется на ФШ.

Пакет прикладных программ конструкторско-топологического проектирования отражает основные тенденции в области проектирования и изготовления ФШ больших интегральных схем (БИС): непрерывное совершенствование технологической базы САПР; повышение требований к качеству генерируемого изображения; увеличение объема проектно-конструкторских работ вследствие усложнения функций, возлагаемых на БИС, и повышения предъявляемых к ним требований.

Программные средства проектирования топологии БИС позволяют работать в двух режимах: обработки совмещенного чертежа топологической структуры БИС и конструирования

топологии БИС с помощью типовых фрагментов. При этом они обеспечивают выполнение следующих функций:

1) описание точного совмещенного топологического чертежа с помощью кодировщика графической информации с параллельной визуализацией многослойного изображения на экране запоминающего графического дисплея и синтаксическим контролем;

2) формирование и графическое редактирование элементов и типовых фрагментов топологии БИС, представленных прямоугольниками, параллельными осям координат, наклонными прямоугольниками, односвязными многоугольниками без внутренних острых углов и границей в виде отрезков прямых, составляющих с осью абсцисс угол кратный $\pi/4$, трассой (соединительной шиной заданной ширины, границу которой составляют попарно-параллельные отрезки с углами наклона относительно оси абсцисс кратными $\pi/4$), кругами, элементами библиотеки;

3) ввод алфавитно-цифровой информации;

4) формирование библиотек элементов и типовых фрагментов топологии БИС и архива на магнитном диске или ленте;

5) манипулирование типовыми фрагментами топологии БИС на экране запоминающего графического дисплея: ввод и отображение фрагментов топологии, выполнение различных последовательностей элементарных графических операций над фрагментами (сдвиг, поворот, масштабирование, зеркальное отображение, мультиплицирование, выделение окна), удаление и копирование фрагментов топологии, проведение связей;

6) проверка на конструкторско-технологические ограничения с возможностью визуализации нарушений проектных норм на запоминающем графическом дисплее и графопостроителе;

7) компиляция послойного описания топологии БИС в элементы входного набора генераторов изображений ЭМ-559, ЭМ-5009, ЭМ-5009А;

8) упорядочение полученного набора элементов с одновременной докомпиляцией отдельных элементов этого набора;

9) трансляция описания полученного набора элементов на язык генератора изображений (запись управляющей информации на магнитную ленту);

10 декомпилирование управляющей информации для генераторов изображений в форматы исходного описания с целью

последующего редактирования и вывода на запоминающий графический дисплей и графопостроитель;

11) имитация работы генераторов изображений;

12) прорисовка фрагментов топологии БИС на графопостроителе в заданном масштабе в режимах *on-line* для ЭМ-7022 и *off-line* для ЭМ-732;

1) протоколирование работы пользователя.

## 7.4 ФАЙЛОВАЯ СИСТЕМА ОС «КУЛОН»

Файл в ОС «Кулон» — именованная последовательность записей, размещаемых на внешних носителях (томах), — магнитных дисках и магнитных лентах, и рассматриваемых в процессе пересылки и обработки, как единое целое. Другими словами, ОС «Кулон» игнорирует различия между файлами, содержащимися на томах разных типов и обеспечивает единство структуры файлов при передаче между этими томами.

Файловая система ОС «Кулон» представляет набор системных программ и предназначена для работы как с устройствами, имеющими файловую структуру (накопители на магнитных дисках), так и устройствами последовательного доступа. Программы управления файловыми структурами реализуют функции динамического формирования, расширения, копирования, удаления, переименования файлов на внешних магнитных носителях.

Том с файловой структурой ОС представляет собой группу файлов, находящихся на одном из накопителей. Все файлы имеют одинаковый формат: области заголовка и данных. ОС имеет прямой доступ к любому из этих файлов, используя указатели файлов, хранящиеся в каталоге накопителя.

Каталог программных файлов накопителя имеет постоянную длину, равную двадцати трем секторам для накопителя на магнитном диске и размещается с первого сектора магнитного диска или сразу после маркера начала магнитной ленты.

Область заголовка файла содержит всю информацию, необходимую для обработки файла, в том числе дату и время организации файла. Эта область разбита на зоны.

*Зона «владельца» файла* содержит идентификационный код пользователя, сформировавшего файл. Пользователь, желающий получить доступ к файлу, должен знать идентификационный код, под которым данный файл был сформирован, и иметь разрешение

на право доступа к каталогу и самому файлу.

*Идентификационный код* пользователя состоит из двухсимвольного имени пользователя, которому предшествует признак имени — символ ¤, и определяет категорию пользователя, работающего с ОС «Кулон». Так, имя ¤!! зарезервировано ОС для работы системного программиста; под именем ¤%% администратор ОС «Кулон» формирует общую библиотеку спроектированных фрагментов топологии, доступных каждому из пользователей; пользователь-оператор регистрируется в ОС под произвольными именами, такими как ¤ AA, ¤ ID и т. д.

*Зона имени файла* содержит имя, присвоенное файлу при его создании. Имя файла представляет собой произвольный символьный идентификатор, состоящий не более чем из шести символов.

*Зона типа файла* определяет мнемонический код, идентифицирующий файл по функции. Тип файла (или расширение) — произвольный идентификатор, содержащий от одного до шести символов. Он используется как дополнительная характеристика к имени файла. Имя файла и тип всегда отделяются друг от друга пробелом.

В общем случае, как имена, так и типы файлов могут выбираться пользователями произвольно. В то же время для ряда информационных файлов в ОС «Кулон» выделены стандартные значения типов. Например, файл типа SOURCE — исходный файл описания топологии БИС в сжатой геометрической форме, соответствующий языку описания топологии (последовательность изменений типов файлов приведена на рис. 7.5); файл типа OBJ — результат компиляции исходного SOURCE-файла в произвольные прямоугольники разбиения (покрытия) топологии БИС; файл типа PAT, который образуется из OBJ-файла, — это файл, содержащий данные для управления генератором изображения на его входном языке; файл типа DECOMP — результат обратного преобразования

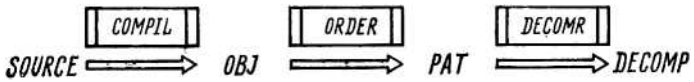



**Рис. 7.5. Последовательность изменения значений типов файлов**

(декомпиляции) PAT-файла в формат SOURCE-файла, содержащего прямоугольники покрытия топологии БИС, и т. д.

В процессе обработки файлов стандартные значения типов

файлов могут автоматически изменяться, при этом пользователь должен учитывать, что каждая конкретная программа, предназначенная для работы с файлами, оперирует только с файлами определенного типа.

*Зона указателей данных* описывает физическое расположение файла на накопителе (число секторов, занимаемое файлом и номер сектора, начиная с которого располагается данный файл). При обращении к файлу в ОС «Кулон» используется стандартная командная строка следующего формата:

$ВЫХ_1$, $ВЫХ_2$, ..., $ВЫХ_N$ ␣ $ВХ_1$, $ВХ_2$, ..., $ВХ_N$,

где вых — спецификация выходного файла; вх — спецификация входного файла; ␣ — знак пробела.

Точки означают, что число спецификаций может быть любым. В случае, если спецификация выходного файла не требуется, то она опускается.

Каждая спецификация файла имеет следующий формат:

влад ␣ устр ␣ имя ␣ тип,

где влад — идентификационный код пользователя, сформировавшего файл; устр — имя устройства, содержащего том, на котором находится файл; имя — имя файла, т. е. алфавитно-цифровая строка от одного до шести символов; тип — мнемокод, дополнительно идентифицирующий файл.

Имя устройства состоит из двух символов и номера устройства, которым предшествует признак имени — символ ¤. Когда имя не включает номер устройства, ОС «Кулон» по умолчанию подразумевает устройство с номером нуль. Приведем примеры спецификаций файла:

```
¤ID ¤MT1 TEST PAT001
¤AA ¤DK1 PRIM SOURCE
¤!! ¤DK1 PROGRM FOCALD
¤ID ¤DK INSTR TEXT
```

Последний пример эквивалентен записи ¤ID ␣ ¤DK0 ␣ INSTR ␣ TEXT.

## 7.5 ФОКАЛ КАК ИНСТРУМЕНТАЛЬНОЕ СРЕДСТВО ПРОГРАММИРОВАНИЯ ИГС «КУЛОН»

Ранее упоминалось, что в инструментальных средствах

программирования ИГС «Кулон» отсутствуют компиляторы и интерпретаторы с языков высокого уровня. Использование Ассемблера в качестве единственного языка для расширения системного и прикладного программного обеспечения ИГС затрудняет процесс разработки пакетов прикладных программ машинной графики, инженерных расчетов, предъявляет высокие требования к квалификации программистов и удлиняет цикл разработки программного обеспечения. Для сокращения трудоемкости разработки графических пакетов прикладных программ проблемно-ориентированных САПР и программных средств для решения инженерных задач расчетного характера в ОС «Кулон» был включен интерпретатор диалогового языка высокого уровня ФОКАЛ, дополненный средствами обработки графической информации, работы с дисками, с виртуальными массивами и другими сервисными средствами [7, 23].

ФОКАЛ ОС «Кулон» практически совпадает с языком перфоленточной диалоговой системы программирования ДС СМ [9]. Ниже рассматриваются только его отличия.

**Элементы данных.** В дополнение к обычным переменным языка ДС СМ в ФОКАЛе ОС «Кулон» введены табличные и системные переменные, и виртуальные массивы (не следует путать их с переменными, хранящимися в таблице переменных ФОКАЛа).

*Табличные переменные*. Переменные, первый символ имени которых — любая из букв латинского алфавита от A до Z; второй — произвольная цифра от 0 до 4, относятся к табличным. Исключение составляют переменные Z3 и Z4, их относят к обычным переменным.

Табличная переменная обладает следующими свойствами: хранится не в таблице переменных ФОКАЛа, а в специально отведенной в самом интерпретаторе таблице; занимает в памяти два слова; является глобальной переменной и служит для передачи информации между программными модулями; имеет заранее известный постоянный адрес в таблице интерпретатора.

Обращение к табличным переменным осуществляется быстрее, чем в обычном; для хранения табличной переменной требуется меньше памяти и они, в отличие от обычных переменных, при вызове новой программы пользователя сохраняют свое значение.

*Системные переменные.* Значения целого ряда специальных табличных переменных, которые называются системными, устанавливаются в процессе работы ОС «Кулон» или интерпретатора ФОКАЛа.

У системных переменных второй символ имени — это цифра 4. Системные переменные содержат информацию о текущем состоянии выполняемой пользователем программы, о текущих значениях параметров графических устройств ввода — вывода. Ниже приводится список системных переменных.

| Значение переменной | Имя переменной |
|---|---|
| Координата точки X, снятой кодировщиком графической информации ЭМ-719 или ЭМ-729 . . . | A4 |
| Координата точки Y, снятой кодировщиком графической информации ЭМ-719 или ЭМ-729 .... | B4 |
| Десятичный код клавиши функциональной клавиатуры пульта контроля и управления ..... | C4 |
| Текущий номер элемента виртуального массива, открытого по каналу *M* ..... | M4 |
| Текущий номер элемента виртуального массива, открытого по каналу *N* ..... | N4 |
| Индексы массива (при каждом обращении значение переменных увеличивается на единицу) . . . | Q4, R4 |
| Указатель вывода в файл по оператору OPERATE | S4 |
| Указатель ввода из файла по оператору OPERATE | T4 |
| Номер текущей строки выполняемой программы | X4 |
| Номер последующей строки выполняемой программы ..... | Y4 |
| Число блоков в файле, открытом оператором LIBR DIM | U4 |

**Виртуальные массивы.** В ФОКАЛе ОС «Кулон» можно использовать файлы этой системы для накопления данных в виде массивов, подобных обычным переменным с индексами. Эти массивы называются виртуальными, так как программист обращается с ними так же, как и с обычными переменными с индексами, хранящимися в оперативной памяти, а не в файлах; для краткости, файлы, содержащие значения виртуальных переменных, также называются виртуальными.

Виртуальные массивы в ФОКАЛе ОС «Кулон» описываются именем, начинающимся с буквы M или N, и указателем размерности. Массивы могут быть одномерные и двумерные.

Указатель размерности помещается в скобки и состоит из одного или двух целых положительных чисел. В указателе размерности могут использоваться числа, переменные или выражения. В случае одномерного массива указатель размерности изменяется от 0 до 16383; в случае двумерного массива первый индекс изменяется от 0 до 63, а второй — от 0 до 128. При работе с двумерными виртуальными массивами следует помнить, что

первый индекс массива — номер блока диска; второй — номер элемента в данном блоке. В каждый момент времени в оперативной памяти находится только один блок (128 элементов), поэтому время последовательного доступа к элементам, находящимся в разных блоках, значительно превосходит время, затрачиваемое на последовательное обращение к элементам одного блока.

Приведем пример работы с виртуальным массивом!

```
*1.10 L D M(5),TEST
*1.20 F J=1,600; M(J)=J^2
*1.30 L C M
```

В данном примере программа открывает виртуальный файл размером в пять блоков по каналу М и присваивает его элементам возведенные в квадрат значения индексов.

**Оператор присваивания SET.** В ФОКАЛе ОС «Кулон» в операторе присваивания можно опускать имя оператора SET.

Приведем пример оператора присваивания:

```
*PI=3.14159
*Y=Y+1
```

**Оператор стирания ERASE.** Этот оператор базового ФОКАЛа имеет в ОС «Кулон» дополнительную форму записи: ERASE P. Оператор ERASE с аргументом *P* стирает табличные и системные переменные ФОКАЛа ОС «Кулон».

**Операторы ввода—вывода.** В операторах ввода—вывода ASK и TYPE ФОКАЛа ОС «Кулон» можно использовать в качестве элемента ввода — вывода символ кавычки «”». В этом случае текст, содержащий кавычки, должен заключаться в апострофы «’».

Например, в результате выполнения директивной строки

```
*TYPE 'МИНИ-ЭВМ "ЭЛЕКТРОНИКА-79"'
```

на дисплей или устройство печати будет выведен текст: мини-ЭВМ

«Электроника-79».

В операторах TYPE и ASK в качестве аргументов можно задавать два специальных символа управления позиционированием вывода информации на экране дисплея — * и #.

Применение управляющего символа * (соответственно #), за которым следует целая переменная K (L), приведет к размещению элемента ввода — вывода в K-й строке (L-й позиции в строке) на экране символьного дисплея.

Отметим, что параметры K и L имеют следующий допустимый диапазон изменения своих значений: для K от 0 до 24, для L от 0 до 80.

Приведем пример использования символов *, #:

```
*1.10 K=10; L=5
*1.20 T *K,#L,"ОПЕРАТОР ВВОДА-ВЫВОДА"
```

При выполнении данного примера текст, заключенный в кавычки, выведется начиная с 5-й позиции 10-й строки.

**Операторы группы LIBR.** В ФОКАЛе ОС «Кулон» предусмотрена группа операторов LIBR, обеспечивающая возможность доступа к любым внешним устройствам, работающим под управлением ОС «Кулон».

Операторы группы LIBR представляют пользователю возможность хранения программ и (или) массивов данных на устройстве с файловой структурой ОС «Кулон» и присвоения этой информации собственного имени — «имени файла», которое может использоваться для обращения к хранимому материалу.

Как уже отмечалось, имя файла состоит из двух самостоятельных частей: собственно имени файла, включающего до шести алфавитно-цифровых символов, и расширения файла. Расширение файла необязательно, так как оно уточняет первую часть «имени файла», указывая на принадлежность файла к определенному типу.

Поясним основные принципы организации доступа к файлам в ФОКАЛе ОС «Кулон» и соответствующую терминологию. Доступ к файлу осуществляется через один из двух предназначенных для этого *каналов,* помеченных идентификаторами M и N. Канал, используемый в данный момент для доступа к некоторому файлу, называется *связанным,* а соответствующий файл — *открытым.* В

любой момент времени канал может быть связан только с одним файлом. Канал, не связанный ни с каким файлом, называется *свободным*. Файл, не связанный ни с каким каналом, называется *закрытым*.

*Общий формат операторов групп* LIBR. Все операторы имеют одинаковый синтаксис и в каждом из них используется от двух до пяти полей:

```
<оператор работы с файлом>::= L[IBR] <ОПР>
    [<идентификатор канала>[<<размер файла>>],]
      [<спецификация файла>].
```

где ОПР — подоператор, используемый для указания типа оператора группы LIBR (подоператор можно сократить до одного символа; за ним должен следовать хотя бы один пробел).

Возможны следующие подоператоры: S [AVE] —для записи пользовательской информации (программы, данных) в файл ОС «Кулон»; O [LD] — для загрузки (записи) программы в память ИГС из указанного файла на внешнем устройстве; R [EPLACE] — для повторной записи программы в файл, ранее созданный оператором LIBR; D [IM] — для открытия файла по одному из указанных каналов; C [LOSE] — для закрытия файла, освобождения связанного с ним канала и сохранения файла на внешнем устройстве.

К остальным полям оператора LIBR относятся: <*идентификатор канала*> — идентификатор канала, требуемый для некоторых операторов группы LIBR;

<*размер файла*> — параметр, используемый для указания размера создаваемого файла (в *блоках*); представляет собой число, помещаемое между открывающей и закрывающей круглыми скобками;

<*спецификация файла*> — рассмотренный ранее формат файла ОС «Кулон».

Необходимо отметить, что в спецификации файла отдельные поля могут быть опущены. В этом случае опущенным параметрам присваивают значения по умолчанию. Например, если в спецификации файла опущены поля «имя пользователя» и «имя устройства», то по умолчанию ФОКАЛ ОС «Кулон» используют в качестве имени пользователя и устройства те имена, под которыми пользователь зарегистрировался в ОС. Идентификаторы FOKAL и FOKALD — типы файлов по умолчанию для файлов, содержащих

программы на ФОКАЛе и массивы данных для них, соответственно.

*Оператор* LIBR SAVE. Для сохранения программы на ФОКАЛе в файле ОС «Кулон» с целью ее дальнейшего использования применяется оператор LIBR SAVE, имеющий следующий синтаксис:

```
<оператор сохранения>::= L[IBR] S[AVE]
      <спецификация файла>.
```

Выполнение этого оператора заключается в создании файла в соответствии с заданной спецификацией, выполнении оператора WRITE ALL для записи программы во вновь созданный файл, и закрытии файла.

Например, оператором *LS PROG1, находящаяся в памяти программа будет сохранена в файле под именем PROG1 FOKAL в области данного пользователя.

В случае, если спецификация файла содержит имя уже существующего файла, то будет выдано сообщение об ошибке с кодом 44.

Сохранять программы можно только в той области, в которой зарегистрирован пользователь, в противном случае будет выдано сообщение об ошибке с кодом 59.

*Оператор* LIBR OLD. Программа, которая ранее была сохранена, может быть загружена в оперативную память из файла и запущена оператором LIBR OLD, имеющим следующий синтаксис:

```
<оператор загрузки>::= L[IBR] O[LD]
     <спецификация файла>.
```

Оператор вводит программу из указанного файла в оперативную память, стирая предварительно таблицу переменных и пользовательский буфер, и запускает ее, как это произошло бы при использовании оператора GOTO базового ФОКАЛа. Например, оператор

```
*L O ¤ID ¤DK1 CHAIN
```

загрузит программу из файла CHAIN FOKAL, находящегося на внешнем устройстве DK1 в области пользователя ID, в оперативную память и запустит ее.

*Оператор* LIBR REPLACE аналогичен оператору LIBR SAVE и

используется для повторной записи программы на ФОКАЛе в файл, предварительно созданный оператором LIBR SAVE. Он имеет следующий синтаксис:

```
<оператор повторного сохранения>::= L[IBR]
      R[EPLACE].
```

Оператор предоставляет пользователю возможность записывать программу из оперативной памяти в тот же файл, из которого она была загружена (оператором L O) и удобен при отладке программы.

*Оператор* LIBR DIM предназначен для подготовки файлов, находящихся на устройствах с файловой структурой ОС «Кулон», к использованию в программах на ФОКАЛе.

Оператор имеет следующий синтаксис:

```
<оператор открытия файла>::= L[IBR] D[IM]
      <идентификатор канала>[<<размер файла>>],
      <спецификация файла>.
```

Он проверяет, существует ли файл с указанной спецификацией; если да, то он открывает по указанному каналу виртуальный файл (массив) и содержащиеся в файле данные будут использоваться во всех последующих операциях при обращении к заданному каналу; в противном случае, оператор L D создает временный файл заданного размера. Если в операторе задана не полная спецификация файла и тип файла не указан, то по умолчанию в качестве типа файла используется идентификатор FOKALD.

Например, программа

```
*1.10 L D M(6),FILE
*1.20 F J=0,99; M(J)=M(J)+10
*1.30 L C M
```

проверит существует ли файл FILE FOKALD в области данного пользователя; если такого файла нет, то он будет создан (размером 6 блоков; 1 блок — это 128 чисел с плавающей запятой) по каналу M и при последующей работе программа присвоит значение первым 100 элементам файла, а затем закроет канал и сохранит файл на устройстве ОС «Кулон».

*Оператор* LIBR CLOSE закрывает файл, открытый по указанному каналу, закрепляя все изменения, произведенные в нем,

и освобождает канал. Оператор имеет следующий синтаксис:

**<оператор закрытия файла>::= L[IBR] C[LOSE] <идентификатор канала>.**

Необходимо помнить, что если существовавший ранее файл был открыт, а затем модифицирован, то до тех пор, пока он не будет закрыт оператором LIBR CLOSE, нельзя быть уверенным, что все произведенные изменения в нем сохранятся по окончанию работы с ФОКАЛом.

**Работа с файлами прямого доступа.** Довольно часто пользователю приходится работать с большими блоками данных как с массивами, а имеющийся объем памяти не позволяет запомнить всю информацию сразу. Для разрешения указанного противоречия в ФОКАЛе ОС «Кулон» обеспечена возможность обмена информацией с устройствами с произвольным доступом (магнитными дисками) посредством операторов ввода — вывода TYPE, ASK, WRITE и расширены свойства оператора OPERATE. В стандартный список символических имен ввода — вывода оператора OPERATE дополнительно включены следующие обозначения устройств ввода — вывода: M — ввод о магнитного диска по каналу M; N — ввод с магнитного диска по каналу N; I — вывод на магнитный диск по каналу M; J — вывод на диск по каналу N.

Указателями и счетчиками ввода — вывода при работе с файлами служат системные переменные T4 и S4. Меняя значения этих переменных можно осуществить ввод — вывод с любого места в файле (прямой доступ к файлу). Диапазон изменения значений системных переменных определяется максимально допустимым размером (64 блока; 1 блок равен 512 байтам) файла прямого доступа.

Примеры

1. Приведенная ниже программа создаст в области данного пользователя на устройстве DK3 файл с именем MICHA FOKALD, содержащий таблицу случайных чисел:

```
*1.10 S4=0; L D M(3),¤DK3 MICHA
*1.20 O I; F K=1,10; T !; D 2
*1.30 L C M; O T
*1.40 Q
*2.10 F J=1,10; T FRAN()," "
```

```
*2.20 R
```

Отметим, что после выполнения данной программы созданный файл можно распечатать средствами ФОКАЛа или ОС «Кулон». Значение указателя вывода S4 равно нулю, поэтому запись данных в файл производится с его начала.

2. Если файл с именем INPUT DAT был создан, например, в другой программе с помощью операторов TYPE, WRITE или средств ОС «Кулон», то программа для чтения из него данных может иметь следующий вид:

```
*1.10 T4=20; L D N(3),INPUT_DAT
*1.20 C: СУММИРОВАНИЕ 50 ЗНАЧЕНИЙ И ПЕЧАТЬ
*1.30 SUM=0; O N
*1.40 F P=1,50; A X; SUM=SUM+X
*1.50 O K; L C N; T !,"СУММА РАВНА ",SUM,!; Q
```

Отметим, что значение указателя ввода T4 равно 20, поэтому программа производит суммирование 50-ти элементов виртуального массива, начиная с 20-го.

**Графические средства ФОКАЛа.** Для обеспечения взаимодействия пользователя с графическими средствами подсистемы отображения и диалога ИГС «Кулон» при обмене графической информацией, решении геометрических задач, формировании изображений, в ФОКАЛ ОС «Кулон» включены так называемые «*оверлейные сегменты*» или просто «*оверлеи*» и средства работы с ними.

Оверлей интерпретатора ФОКАЛа — это сегмент программы на языке Ассемблера, находящийся на внешнем запоминающем устройстве и вызываемый по мере необходимости в рабочую область интерпретатора, ранее загруженную другими его сегментами. Для вызова оверлеев в ФОКАЛе ОС «Кулон» предусмотрен оператор UOVER, который имеет следующий синтаксис:

```
<оператор работы с оверлеем>::= U[OVER]
      <имя оверлея> [<аргументы>].
```

где (имя оверлея) — двухсимвольный идентификатор; (аргументы) — поле параметров оверлея (для некоторых оверлеев параметры могут отсутствовать).

*Оверлей* F1 предназначен для выполнения всех стандартных функций графического редактора ОС «Кулон» (т. е. программы

DRAW), обеспечивающих описание точного совмещенного топологического чертежа с помощью большого кодировщика графической информации (КГИ); формирование и графическое редактирование элементов и типовых фрагментов с помощью малого кодировщика (МК), визуальный контроль топологии на экране графического дисплея и т. д. Формат обращения к оверлею имеет следующий вид: U F1 X, где X — код управляющей команды графического редактора.

В случае, если команда задается с функциональной клавиатуры, то X — десятичный код соответствующей клавиши; если команда подается с клавиатуры символьного дисплея, то значение X, увеличенное на 10 000 — десятичный код соответствующего символа на клавиатуре дисплея (см. табл. 7.1 и 7.2), в которых приведены коды ряда команд программы DRAW).

Например, в результате выполнения программы

```
*1.10 U F1 260
*1.20 U F1 10085
```

изображение на экране графического дисплея смещается влево на 1/4 ширины экрана (команда «влево») и размер перекрестия (маркера) увеличивается на весь экран графического дисплея (команда «U»).

Таблица 7.1

**Коды управляющих команд графического редактора на функциональной клавиатуре**

| Команда | $Код_{10}$ | Назначение |
|---|---|---|
| Сетка 1 | 1 | Активизирует перемещение маркера с шагом сетки 1 |
| Точечная сетка | 8 | Воспроизводит на экране ЗГД точечную сетку с шагом, равным шагу активной в данный момент сетки 1 или 2, или 3 |
| Панель | 516 | Включает КГИ и функциональную клавиатуру |
| Планшет | 522 | Включает МК |
| Шкала | 257 | Воспроизводит на экране ЗГД сетку с шагом 0,1 мкм |
| Регенерация | 272 | Обновляет изображение на экране ЗГД |

| | | |
|---|---|---|
| Вверх | 258 | Перемещает изображение на 1/4 высоты экрана ЗГД |
| Вниз | 288 | То же |
| Влево | 260 | Перемещает изображение на 1/4 ширины экрана ЗГД |
| Вправо | 320 | То же |
| Увеличение | 264 | Увеличивает масштаб изображения на экране ЗГД |
| Уменьшение | 384 | Уменьшает масштаб изображения на экране ЗГД |
| Центр | 513 | Центрирует фрагмент изображения на экране ЗГД |
| Центр увеличения | 514 | Центрирует фрагмент изображения с одновременным его увеличением в 15 раз |
| Прямоугольник | 1025 | Вызывает описание прямоугольника |
| Наклонный прямоугольник | 1028 | То же |
| Многоугольник | 1040 | То же многоугольника |
| Трасса | 1056 | То же трассы |
| Экспозиция | 1088 | Вызывает описания диафрагмы генератора изображения ЭМ-559 |
| Ячейка | 1026 | Вызывает и размещает библиотечный элемент |
| Итерация | 1032 | Мультиплицирует библиотечный элемент |
| Раскрыть ячейку | 1282 | Выводит полное изображение элемента библиотеки |
| Отбор | 1281 | Идентифицирует фигуры для редактирования |
| Область | 1296 | Идентифицирует фигуры, принадлежащие заданному окну |
| Сдвиг | 1284 | Изменяет положение фрагментов топологии |
| Копия | 1408 | Копирует изображение фигуры |
| Удалить | 1288 | Стирает фигуру или группу фигур |
| Деформировать | 1344 | Модифицирует геометрию фигур |
| Модифицировать | 1312 | Удаляет один или несколько последовательных отрезков многоугольника |
| Съем КГИ | 1793 | Определяет координаты точки, указанной КГИ |
| Съем МК | 1824 | Определяет координаты точки, указанной МК |

*Оверлей* F2 предназначен для опроса клавиатур функциональной и

символьного дисплея, и определения положения считывающей головки малого и большого кодировщиков графической информации. Формат обращения к оверлею следующий: U F2. При вызове оверлея F2 и задании любой из команд графического редактора, т. е. при нажатии соответствующей клавиши на функциональной клавиатуре или клавиатуре символьного дисплея,

Таблица 7.2

**Коды команд графического редактора на клавиатуре символьного дисплея**

| Команда | Код | Назначение |
|---|---|---|
| A | 10065 | Задает список активных слов |
| C | 10067 | Вызывает и размещает библиотечный элемент |
| Z | 10090 | Вызывает описание диафрагмы генератора изображений |
| F | 10070 | Закрывает текущий файл и обеспечивает выход из графического редактора |
| G | 10071 | Задает значение шага сетки |
| I | 10073 | Мультиплицирует библиотечный элемент |
| N | 10078 | Задает конструкторско-технологические ограничения |
| O | 10079 | Вводит параметры точки привязки фрагмента топологии |
| R | 10082 | Вводит параметры соответствия между сеткой чертежа и кодировщика |
| W | 10087 | Вводит размеры модуля изделия |
| T | 10084 | Вызывает описание текста |
| U | 10085 | Увеличивает размер маркера на весь экран |

происходит ее выполнение, а системной переменной ФОКАЛа ОС «Кулон» С4 присваивается значение кода этой команды, в соответствии с табл. 7.1 и 7.2; при нажатии клавиши «Съем» на курсоре малого или большого кодировщика системным переменным А4 и В4 присваивается значение координат X и Y снятой на кодировщике точки соответственно.

*Оверлей* F5 предназначен для преобразования графического файла типа SOURCE (исходного описания топологии изделия в сжатой геометрической форме) в файл типа FOKALD, обрабатываемый ФОКАЛом ОС «Кулон». Формат обращения к оверлею имеет вид: U F5 $a_1$, $a_2$, $a_3$, $a_4$, где $a_1$ — десятичный код

преобразуемых элементов топологии,

$a_1$ = { 3 — прямоугольник; 7 — трасса; 9 — наклонный прямоугольник; 11 — круг; 23 — многоугольник;

$a_2$ — номер слоя (диапазон изменения от 0 до 9); $a_3$ — число элементов указанного типа; $a_4$ — флаг обработки,

$a_4$ = { 0 — преобразуемые фигуры удаляются из SOURCE-файла; 1 — файл типа SOURCE не изменяется.

Перед обращением к оверлею SOURCE-файл должен быть открыт по каналу M, а файл с расширением FOKALD — по каналу N; системные переменные M4 и N4 — указатели текущих элементов в соответствующих файлах.

Например, в результате выполнения программы

```
*1.10 L D M(5),RAB_SOURCE
*1.20 L B N(5),TEST
*1.30 M4=0: N4=0
*1.40 U F5 3,1,2,1
*1.50 L C M; L C N
```

из файла RAB SOURCE в файл TEST FOKALD будут преобразованы два прямоугольника первого слоя, а сам файл RAB SOURCE останется без изменений.

Результат работы оверлея F5 (файл типа FOKALD) будет представлять собой виртуальный массив, элементы которого отражают структуру описываемых в ОС «Кулон» графических элементов прямоугольников (код прямоугольника 3); N (1) — число элементов массива, требуемое для записи прямоугольника (9); N (2) — номер слоя (1); N (3)—(N (10) — координаты вершин первого прямоугольника; N (11) = 3, N (12) = 9, N (13) = 1, N (14)— N (21) — координаты вершин второго прямоугольника; N (22) — признак конца файла (код $75_{10}$).

*Оверлей* WG предназначен для преобразования данных из

файла типа FOKALD в файл типа SOURCE. Формат обращения к оверлею имеет следующий вид: U WG.

Как и в случае оверлея F5, перед обращением к оверлею WG необходимо открыть файл с расширением FOKALD по каналу N, а SOURCE-файл по каналу *M*.

Например,

```
*1.10 L D N(15),PROG
*1.20 L D M(15),PROG1_SOURCE
*1.30 M4=0; N4=0
*1.40 U WG; L C N; L C M
```

В результате выполнения программы формируется файл PROG1 SOURCE.

*Оверлей* F6 предназначен для программного управления командами графического редактора и имеет следующий формат обращения: UF6 $a_1$, $a_2$, где $a_1$ — количество аргументов команды (характерных действий в диалоговом режиме работы с командой); $a_2$ — код управляющей команды графического редактора (см. табл. 7.1 и 7.2). При обращении к оверлею F6 геометрические данные из файла типа FOKALD преобразуются в форматы данных графического редактора в соответствии с заданной командой и одновременно с этим отображаются на экране графического дисплея.

Например, программа, приведенная ниже, сформирует на экране графического дисплея пять элементов топологии (многоугольник, два прямоугольника, трассу, три наклонных прямоугольника):

```
*1.10 L D N(5),PRIM; N4=0
*1.20 U F6 5,1040; C МНОГОУГОЛЬНИК
*1.30 U F6 4,1025; C ДВА ПРЯМОУГОЛЬНИКА
*1.40 U F6 4,1056; C ТРАССА
*1.50 U F6 9,1028;C 3 НАКЛОННЫХ ПРЯМОУГОЛЬНИКА
```

Координаты характерных точек геометрических фигур находятся в виртуальном файле (массиве), открытом по каналу N : N (0) = Y1, N (1) = X (1) — координаты первой точки многоугольника и т. д.; N (8) = Y5, N (9) = = X5 — координаты пятой, последней точки многоугольника; N (10), N (11), N (12), N

(13) — координаты противоположных вершин первого прямоугольника; N (14) N (15), N (16), N (17) — координаты противоположных вершин второго прямоугольника и т. д. Системная переменная N4 — указатель текущего элемента в виртуальном массиве.

*Оверлей* F7 предназначен для преобразования топологических элементов из SOURCE-файла, указанных с помощью команд графического редактора «ОТБОР» или «ОБЛАСТЬ», в файл типа FOKALD. Формат обращения к оверлею имеет следующий вид: U F7 $a_1$, $a_2$, где $a_1$ — десятичный код преобразуемых элементов топологии; $a_2$ — флаг обработки.

Перед обращением к оверлею должен быть открыт файл типа FOKALD по каналу N и выполнена одна из команд графического редактора «ОТБОР» или «ОБЛАСТЬ».

Например,

```
*1.10 T "ЗАДАЙТЕ КООРДИНАТЫ ПРОТИВОПОЛОЖНЫХ"
*1.11 T " ВЕРШИН ОБЛАСТИ"; U F1 1296
*1.20 L D N(10),REGION; N4=0
*1.30 U F7 9,0; L C N
```

В файл REGION FOKALD из области, указанной двумя точками клавишей «Съем» курсора, запишутся все наклонные прямоугольники, которые затем удалятся из текущего SOURCE-файла.

*Оверлей* K1 предназначен для ввода с заданного устройства текстовой информации и преобразования ее в соответствии с параметрами привязки и ориентации в форматы графического редактора с одновременной визуализацией на экране графического дисплея. Формат обращения к оверлею имеет следующий вид: U K1 X, где X — число точек привязок.

Текст может быть введен с клавиатуры символьного дисплея или из файла прямого доступа.

Перед обращением к оверлею K1 должен быть сформирован и открыт по каналу *N* файл типа FOKALD, содержащий требуемое число точек привязки текста и соответствующие им коды ориентации.

Например, нижеприведенная программа вводит текст с клавиатуры символьного дисплея и прорисовывает его на экране графического дисплея в соответствующих точках привязки с заданной ориентацией:

```
*1.10 L D N(10),PRIV; N4=0
*1.20 U K1 2; L C N
```

Сформированный файл PRIV FOKALD содержит следующую информацию; N (0), N (1) — координаты первой точки привязки текста; N (2) — код ориентации текста; N (3), N (4) — координаты второй точки привязки; N (5) — код ориентации текста.

В следующем примере оверлей K1 преобразует и прорисовывает текст, вводимый из файла прямого доступа TEXT FOKALD,

```
*1.10 L D N(3),TEST; L D M(3),TEXT
*1.20 E P; O I
*1.30 T 'ФОКАЛ ОС "КУЛОН"',!
*1.40 U K1 2; O TK
*1.50 L C N; L C M
```

Отметим, что текст на ФОКАЛе ОС «Кулон» записывается в файл прямого доступа оператором TYPE.

## ПРИЛОЖЕНИЯ

*ПРИЛОЖЕНИЕ 1*

# КРАТКИЙ СПРАВОЧНИК ПО ЯЗЫКУ ФОКАЛ-БК

### Формат описания операторов и функций.

Каждый оператор или функция описываются в следующем формате.

*Синтаксис*. Определяет формат, в котором могут использоваться оператор или функция. При этом используются следующие специальные символы: угловые скобки (< >) обозначают некоторое понятие языка ФОКАЛ (например, <*переменная*>); фигурные скобки ({ }) обозначают, что заключенная в них конструкция может быть повторена нуль и более раз подряд; квадратные скобки ( [ ] ) указывают необязательные элементы, которые могут опускаться при задании оператора или функции; вертикальная черта (|) разделяет альтернативные формы использования оператора; знак ( ␣ ) обозначает символ пробел.

Те синтаксические конструкции, которые не заключены в угловые скобки, должны записываться в строках ФОКАЛа точно так, как они описаны в синтаксическом определении.

Символы ::= используются для описания заключенных в угловые скобки понятий языка и читаются «это есть».

Отметим, что хотя для описания функций используется, как правило, оператор SET, функции могут использоваться в операторах TYPE, XECUTE или в арифметических выражениях.

*Описание.* Объясняет назначение оператора или функции и приводит сопутствующую информацию.

*Диагностическое сообщение.* Приводит коды ошибок и дает их краткое объяснение.

*Пример*: иллюстрирует использование оператора или функции.

*Связанные операторы или функции*: приводится список операторов или функций, имеющих отношение к описываемому оператору (функции) с точки зрения совместного использования или выполнения родственных функций, если они имеются.

**Описание операторов и функций.**

*Ввод данных* ASK

*Синтаксис.*

```
A[SK] <элемент ввода>{,<элемент ввода>},

<элемент ввода>::= <переменная> | "<текст>" |
      %[M.0N] | ¤ .
```

*Описание.* Этот оператор используется для присвоения переменным числовых значений, которые вводятся с клавиатуры. В операторе допускается произвольное сочетание *элементов ввода,* причем текст, записываемый в строке, может содержать любые символы кроме кавычек. При выполнении оператора ASK текст полностью выводится на печать, что удобно использовать для «подсказок» и примечаний. Элементы ввода !, % и ¤ используются для перехода на новую строку, изменения формата вывода чисел и распечатки таблицы переменных соответственно.

Ввод числового значения начинается после появления на экране символа «:», а заканчивается одним из допустимых ограничителей: пробелом, запятой, точкой с запятой, возвратом каретки (клавиша ВВОД) или символом @. Одним оператором ASK можно ввести значения нескольких переменных.

*Диагностические сообщения*: код ошибки 16 — во входных данных больше 23-х символов.

*Пример*:

```
*1.10 ASK "ЗАДАЙТЕ СКОРОСТЬ",V," И ВРЕМЯ",T,!
*1.20 TYPE "РАССТОЯНИЕ S=",V*T,!
```

*Комментарии* COMMENT

*Синтаксис:*

```
C[OMMENT] <строка>
```

*Описание.* Этот оператор используется для включения в текст программы комментариев, которые могут облегчить понимание программы или любой ее части, описать реализованный в программе алгоритм и т. д. Комментарии не влияют на выполнение программы и могут располагаться в любом месте. В *строку комментариев* можно включать любые символы, в том числе прописные и строчные буквы русского алфавита.

*Пример:*

```
*1.10 COMMENT ЭТА ПРОГРАММА ВЫЧИСЛЯЕТ СИНУС
*1.20 COMMENT УГЛА, ЗАДАННОГО В РАДИАНАХ, И
*1.30 COMMENT ВЫВОДИТ ЕГО НА ПЕЧАТЬ
*1.40 ASK "ЗАДАЙТЕ УГОЛ (В РАД.)",X,!
*1.50 TYPE %4,"SIN(",X,")=",%,FSIN(X),!
```

*Вызов подпрограммы* DO. *Синтаксис:*

```
D[O] [<номер группы> | <номер строки> | A[LL]]
```

*Описание.* Этот оператор используется для обращения к строке или группе строк как к подпрограмме. По завершении выполнения подпрограммы управление передается оператору, непосредственно следующему за оператором DO. При обращении к *группе строк* выполнение подпрограммы начинается со строки с наименьшим номером и продолжается либо до исчерпания строк в группе, либо до встречи с оператором RETURN; при обращении к *строке* выполняются все операторы в этой строке.

Оператор DO ALL (или просто DO) вызывает выполнение всей программы, начиная со строки с наименьшим номером (аналогично оператору GOTO). Номер группы или номер строки может быть переменной.

*Диагностические сообщения:* код ошибки 6 — несуществующий *номер группы* или *номер строки*; код ошибки 9 — переполнение стека (глубина рекурсии больше 42).

*Пример:*

```
*1.10 SET A=5; DO 2; DO2
*1.20 TYPE A,!; QUIT
*2.10 TYPE A,!
*2.20 SET A=A-1
```

*Связанные операторы:* RETURN, GOTO.

*Стирание информации* ERASE.

*Синтаксис:*

```
E[RASE] [<номер строки> | <номер группы> |
A[LL] | T[EXT]]
```

*Описание.* Этот оператор стирает в памяти компьютера либо всю таблицу переменных (ERASE), либо только текст программы (весь ERASE TEXT или указанные строки), либо целиком программу и таблицу переменных (ERASE ALL).

Перед началом работы рекомендуется выполнять оператор ERASE ALL, а перед запуском отлаженной программы — оператор ERASE (он может быть первым оператором программы).

*Диагностические сообщения:* код ошибки 1 — неправильный *номер строки*; код ошибки 5 — несуществующий *номер строки.*

*Пример:*

```
*ERASE ALL
*1.10 SET A=2
*1.20 SET C=3
*1.30 TYPE A-B
*1.40 TYPE A-C
*ERASE 1.3; COMMENT СТИРАЕТ СТРОКУ 1.30
```

*Абсолютное значение* FABS.

*Синтаксис*

```
SET <переменная>=FABS(<выражение>)
```

*Описание.* Эта функция используется для получения абсолютного значения *выражения.* Она преобразует отрицательные числа в положительные, оставляя нуль и положительные числа без изменений. Чаще всего функция FABS используется для нахождения модуля разности двух чисел, если их относительная величина неизвестна.

*Пример:*

```
*1.10 SET A=-3.7; SET B=5.1
*1.20 TYPE "АБСОЛЮТНАЯ ВЕЛИЧИНА",A
*1.25 TYPE " РАВНА ",FABS(A),!
*1.30 TYPE "АБСОЛЮТНАЯ ВЕЛИЧИНА",B
*1.35 TYPE " РАВНА ",FABS(B),!
```

*Связанные функции* FSGN.

*Арккосинус* FACOS

*Синтаксис:*

**SET <переменная>=FACOS(<выражение>)**

*Описание.* Эта функция вычисляет угол (в радианах), косинус которого задается *выражением*; для преобразования из радианов в градусы используется множитель 180/π. Значение аргумента функции должно лежать в диапазоне от —1 до +1; —1 соответствует косинусу ±180° (±π радиан), а +1 — косинусу 0 или 360° (0 или 2π радиан). Если значение аргумента по модулю больше единицы, то возникает ошибка.

*Диагностическое сообщение:* код ошибки 20 — значение аргумента по модулю больше единицы.

*Пример:*

```
*1.10 ASK X
*1.20 SET B=FACOS(X)
*1.30 TYPE "АРККОСИНУС ",X," РАВЕН ",B
*1.35 TYPE " РАДИАН",!
```

*Связанные функции:* FSIN, FCOS, FTAN, FASIN, FATAN.

*Арксинус* FASIN.

*Синтаксис:*

**SET <переменная>=FASIN(<выражение>)**

*Описание.* Эта функция вычисляет угол (в радианах), синус которого задается *выражением*; для преобразования из радианов в градусы используется множитель 180/π. Значение аргумента функции должно лежать в диапазоне от —1 до +1; —1 соответствует синусу 270° или —90° (3π/2 или —π/2 радианов), а +1 — синусу 90° (π/2 радиан). Если значение аргумента по модулю больше единицы, то возникает ошибка.

*Диагностическое сообщение:* код ошибки 20 — значение аргумента по модулю больше единицы.

*Пример:*

```
*1.10 ASK X
*1.20 SET X=FASIN(X)
*1.30 TYPE "АРКСИНУС ",X," РАВЕН "
*1.35 TYPE B*180/3.14159," ГРАДУСОВ",!
```

*Связанные функции:* FSIN, FCOS, FACOS, FATAN, FTAN.

*Арктангенс* FATAN.
*Синтаксис:*

```
SET <переменная>=FATAN(<выражение>)
```

*Описание.* Эта функция вычисляет угол (в радианах), тангенс которого задается *выражением;* для преобразования из радианов в градусы используется множитель 180/π. Аргумент функции может принимать любые значения из интервала от —$10^{19}$ до +$10^{19}$, а значения функции лежат в интервале от —π/2 до π/2, т. е. от —90 до 90°.
*Пример:*

```
*1.10 ASK X
*1.20 SET B=FATAN(X)
*1.30 TYPE "АРКТАНГЕНС ",X," РАВЕН ",B
*1.35 TYPE " РАДИАН",!
```

*Связанные функции:* FSIN, FCOS, FTAN, FACOS, FASIN.
*Ввод — вывод символов* FCHR.

*Синтаксис:*

```
SET <переменная>=FCHR(<выражение>{,<выражение>})
```

*Описание.* Эта функция используется для ввода и (или) вывода символов, соответствующих кодам КОИ-7, причем при вводе функция преобразует символы КОИ-7 в десятичное представление, а при выводе преобразует *десятичное* значение аргумента в символ КОИ-7. Если значение аргумента отрицательно, то функция будет считывать символ с клавиатуры, причем значение функции будет равно десятичному значению кода КОИ-7 этого символа; если значение аргумента неотрицательно, то *целая часть* аргумента, взятая по модулю 256, преобразуется в соответствующий код КОИ-7 и выводится на экран. Значение функции FCHR при этом равно целой части значения аргумента (таблица кодов и символов клавиатуры БК дается в приложении 2). При наличии нескольких аргументов функция FCHR последовательно выполняет операции ввода — вывода для каждого аргумента, а ее значение будет равно десятичному значению кода КОИ-7 последнего введенного или

выведенного символа.

*Пример:*

```
*1.10 COMMENT ВЫВОД ДЕСЯТИЧНЫХ ЗНАЧЕНИЙ
*1.11 COMMENT КОДОВ КЛАВИАТУРЫ
*1.20 TYPE FCHR(-1),!
*1.30 GOTO 1.2
```

*Косинус* FCOS.

*Синтаксис:*

```
SET <переменная>=FCOS(<выражение>)
```

*Описание.* Эта функция вычисляет косинус угла, заданного в радианах; для преобразования градусов в радианы используется множитель π/180. Значение функции находится в диапазоне от —1 до +1, а аргумент должен лежать в интервале от $-10^{38}$ до $+10^{38}$.

*Пример:*

```
*1.10 ASK "ВВЕДИТЕ УГОЛ В ГРАДУСАХ",A
*1.20 SET B=A*3.14159/180
*1.30 SET C=FCOS(B)
*1.40 TYPE "КОСИНУС ",A," РАВЕН ",C,!
```

*Связанные функции:* FSIN, FTAN, FACOS, FASIN, FATAN.

*Экспонента* FEXP.

*Синтаксис:*

```
SET <переменная>=FEXP(<выражение>)
```

*Описание.* Эта функция используется для вычисления значений экспоненциальной функции $\mathbf{e}^x$, где X — аргумент функции FEXP. Это обратная по отношению к FLOG функция.

Число e равно приблизительно 2,718282, поэтому запись FEXP (X) аналогична записи 2,718282 ^ *X*.

*Пример:*

```
*1.10 ASK "ВВЕДИТЕ ЧИСЛО",X
*1.20 TYPE "АНТИЛОГАРИФМ ",X," = ",FEXP(X),!
```

*Связанные функции:* FLOG.
*Целая часть числа* FITR.
*Синтаксис:*

```
SET <переменная>=FITR(<выражение>)
```

*Описание.* Эта функция принимает значение целой части своего аргумента со знаком аргумента. Функция FITR может также использоваться для получения дробной части числа. Это достигается вычитанием целой части из исходного числа.
*Пример:*

```
*1.10 ASK "ВВЕДИТЕ ЧИСЛО",A
*1.20 SET B=FITR(A); SET C=FABS(A-B)
*1.30 TYPE "ЦЕЛАЯ ЧАСТЬ ",A," РАВНА ",B,!
*1.40 TYPE "ДРОБНАЯ ЧАСТЬ ",A," РАВНА ",C,!
```

*Связанные функции:* FABS.
*Управление курсором* FK.
*Синтаксис:*

```
XECUTE FK(<X>,<Y>)
```

*Описание.* Эта функция предназначена для установки курсора на экране в точку с координатами (<X>, <Y>). Координаты X и Y могут быть арифметическими выражениями, десятичные значения которых должны удовлетворять следующим условиям: —63 ≤ X ≤ 63, —23 ≤ Y ≤ 23.

Изменение положения курсора на экране осуществляется относительно левого верхнего угла экрана, имеющего координаты (0, 0). Экран имеет 24 позиции по вертикали (положительное направление оси Y — сверху вниз) и 64 — по горизонтали (положительное направление оси X — слева направо).

Функция FK принимает десятичное значение координаты Y.
*Пример:*

```
*XECUTE FK(32,12); С КУРСОР -— В ЦЕНТР ЭКРАНА
```

*Натуральный логарифм* FLOG.
*Синтаксис:*

**SET <переменная>=FLOG(<выражение>)**

*Описание.* Эта функция используется для вычисления натурального логарифма *выражения.* Если значение выражения не положительно, то выдается сообщение об ошибке. Это обратная по отношению к FEXP функция.

*Диагностическое сообщение:* код ошибки 19 — аргумент функции меньше или равен нулю.

*Пример:*

```
*1.10 ASK "ВВЕДИТЕ ЧИСЛО",A
*1.20 TYPE "НАТУРАЛЬНЫЙ ЛОГАРИФМ ",A
*1.25 TYPE " РАВЕН ",FLOG(A),!
```

*Связанные функции:* FEXP.

*Десятичный логарифм* FLOG10.

*Синтаксис:*

**SET <переменная>=FLOG10(<выражение>)**

*Описание.* Эта функция используется для вычисления десятичного логарифма *выражения.* Если выражение принимает значение, равное или меньшее нуля, то выдается сообщение об ошибке.

Для перехода к вычислению логарифма по любому основанию используется тождество

$$\log_a x = \log_b x / \log_b a.$$

*Диагностическое сообщение:* код ошибки 19 — аргумент функции, который равен нулю или меньше его.

*Пример:*

```
*1.10 ASK "ВВЕДИТЕ ЧИСЛО",A
*1.20 SET B=FLOG10(A); SET C=B/FLOG10(2)
*1.30 TYPE "ДЕСЯТИЧНЫЙ ЛОГАРИФМ ОТ ",A
*1.35 TYPE " РАВЕН ",B,!
*1.40 TYPE "ЛОГАРИФМ ПО ОСНОВАНИЮ 2 ОТ ",A
*1.45 TYPE " РАВЕН ",C,!
```

*Связанные функции:* FLOG.

*Организация цикла* FOR.

*Синтаксис:*

**F[OR] <переменная>=<начальное значение>[[,<шаг>],<предельное значение>];{<оператор>}**

*Описание.* Этот оператор используется для организации повторяющихся (циклических) вычислений *операторов,* находящихся в одной строке с оператором FOR. Число итераций (повторений) цикла определяется значениями управляющих параметров цикла (начальным значением, шагом, предельным значением):

$$\max\left(\mathrm{E}\left[\frac{\langle\text{Предельное значение}\rangle - \langle\text{Начальное значение}\rangle + \langle\text{Шаг}\rangle}{\mathrm{abs}\,(\langle\text{Шаг}\rangle)}\right], 1\right),$$

где E — целая часть числа; abs — абсолютное значение.

Управляющие параметры могут быть арифметическими выражениями; если параметр «шаг» опущен, то по умолчанию цикл выполняется с шагом 1; если опущено *предельное значение,* то оператор FOR оказывается аналогичен оператору SET.

*Диагностическое сообщение:* код ошибки 07 — неправильный формат оператора FOR.

*Пример:*

```
*1.10 COMMENT ВЫЧИСЛЕНИЕ N-ФАКТОРИАЛ С
*1.11 COMMENT ПОМОЩЬЮ ОПЕРАТОРА ЦИКЛА
*1.20 ASK "ЗАДАЙТЕ N ( <=34 )",N
*1.30 SET NF=1
*1.40 FOR J=2,N; SET NF=NF*(N-J+2)
*1.50 TYPE %,"N-ФАКТОРИАЛ = ",NF,!
```

*Работа с портом ввода — вывода* FP. *Синтаксис:*

**SET <переменная>=FP(<код операции>,<маска>)**

*Описание.* Эта функция используется для обмена информацией между устройством пользователя, подключенным через разъем «УП» к порту, и компьютером в соответствии с указанным кодом операции: 0 — чтение регистра ввода по маске; 1 — очистка битов регистра вывода по маске; 2 — установка битов регистра вывода по маске; 3 — чтение регистра вывода по маске (*маска — восьмеричное число* в диапазоне от 0 до 177777 или переменная с десятичным

значением от —32768 до 32767). При выполнении операции чтения с регистра ввода и регистра вывода выполняется (поразрядно) операция «ЛОГИЧЕСКОЕ И» между считываемой с регистра информацией и маской.

Обнуление (установка) битов регистра вывода по маске происходит в соответствии с теми битами маски, которые установлены в единицу. Функция FP принимает десятичное значение считанной или записанной информации.

*Диагностическое сообщение* код ошибки 27 — неправильный код операции.

*Пример:*

```
*C УСТАНАВЛИВАЕТ ВСЕ БИТЫ РЕГИСТРА ВЫВОДА
*XECUTE FP(2,177777);
*TYPE FP(3,40),!; C 40(8)=32(10)
  32.0000
*
```

*Случайное число* FRAN.

*Синтаксис:*

```
SET <переменная>=FRAN([1])
```

*Описание.* Эта функция используется для генерации псевдослучайных чисел в интервале (—1, 1) с равномерным законом распределения и нулевым математическим ожиданием. Для повторения одной и той же последовательности случайных чисел необходимо выполнить функцию FRAN (1), которая устанавливает начальное состояние этой последовательности. Это связано с наличием системной переменной, которая изменяется при каждом обращении к функции FRAN вида FRAN ( ), но устанавливается в начальное состояние при обращении вида FRAN (1).

*Пример:*

```
*1.10 COMMENT ГЕНЕРАЦИЯ СЛУЧАЙНЫХ ЧИСЕЛ НА
*1.11 COMMENT ИНТЕРВАЛЕ (A,B)
*1.20 ASK "ЗАДАЙТЕ НИЖНЮЮ ГРАНИЦУ",A
*1.30 ASK "ЗАДАЙТЕ ВЕРХНЮЮ ГРАНИЦУ",B
*1.40 TYPE %,A+[FRAN()+1]*(B-A)/2,!
*1.50 GOTO 1.40
```

*Вызов пользовательской функции* FSBR.
*Синтаксис:*

```
SET <переменная>=FSBR(<номер группы>,
     <выражение>)
```

*Описание.* Эта функция используется для выполнения группы строк (или одной строки) как подпрограммы, которая представляет собой процедуру-функцию с одним формальным параметром (специальной переменной &). Первый аргумент функции указывает номер группы, реализующей алгоритм программируемой функции FSBR; второй аргумент (выражение) является фактическим параметром, значение которого присваивается специальной переменной & при обращении к функции FSBR.

Возврат из подпрограммы осуществляется либо по оператору RETURN, либо по исчерпанию строк в группе. В результате обращения к функции FSBR последнее вычисленное значение специальной переменной & становится значением функции.

*Диагностические сообщения:* код ошибки 6 — несуществующий номер строки или группы в функции FSBR; код ошибки 9 — переполнение стека (глубина рекурсии больше 20).

*Пример:*

```
*1.10 COMMENT ВЫЧИСЛЕНИЕ ФАКТОРИАЛА ЧИСЛА N
*1.20 ASK "ВВЕДИТЕ ЧИСЛО",N
*1.30 TYPE FSBR(2,N); QUIT
*2.10 IF (1-&)2.20; RETURN
*2.20 SET &=&*FSBR(2,&-1)
```

*Связанные операторы:* RETURN. *Знак числа* FSGN.
*Синтаксис:*

```
SET <переменная>=FSGN(<выражение>)
```

*Описание.* Эта функция используется для определения знака выражения: если значение выражения больше нуля, то значение функции равно единице; если значение выражения равно нулю, то значение функции тоже равно нулю; если значение выражения меньше нуля, то значение функции равно минус единице.

Эта функция применяется, главным образом, в тех

вычислениях, в которых не могут использоваться числа определенного знака.

*Пример.* В приведенном ниже примере вычисляется кубический корень из произвольного числа, при этом на этапе извлечения кубического корня (возведения в степень 1/3) знак «отделяется» от числа, а потом «добавляется» к результату:

```
*1.10 ASK "ВВЕДИТЕ ЧИСЛО",A
*1.20 SET B=FSGN(A); SET C=FABS(A)
*1.30 SET D=B*C^(1/3)
*1.40 TYPE "КУБИЧЕСКИЙ КОРЕНЬ ИЗ ",A," РАВЕН ",D
```

*Связанные функции:* FABS.

*Синус* FSIN.

*Синтаксис:*

```
SET <переменная>=FSIN(<выражение>)
```

*Описание.* Эта функция вычисляет синус угла, заданного в радианах; для преобразования из градусов в радианы используется множитель $\pi/180$.

Значение функции находится в диапазоне от —1 до +1, а значение *выражения* должно лежать в интервале $(—10^{38}, +10^{38})$.

*Пример:*

```
*1.10 ASK "ЗАДАЙТЕ УГОЛ В ГРАДУСАХ",A
*1.20 SET B=FSIN(A*3.141592/180)
*1.30 TYPE "СИНУС ",A," РАВЕН ",B
```

*Связанные функции* FCOS, FTAN, FASIN, FACOS, FATAN.

*Квадратный корень* FSQT.

*Синтаксис:*

```
SET <переменная>=FSQT(<выражение>)
```

*Описание.* Эта функция вычисляет квадратный корень из арифметического *выражения,* значение которого должно быть неотрицательным числом, в противном случае возникает ошибка.

*Диагностическое сообщение:* код ошибки 17 — корень квадратный из отрицательного числа.

*Пример:*

```
*1.10 ASK "ВВЕДИТЕ ЧИСЛО",A
*1.20 IF [FSGN(A)]1.60; SET B=FSQT(A)
*1.40 TYPE "КВАДРАТНЫЙ КОРЕНЬ ИЗ ",A
*1.45 TYPE " РАВЕН ",B,!
*1.50 QUIT
*1.60 TYPE "ОШИБКА -- ЧИСЛО < 0"
```

*Формирование точки* FT. *Синтаксис:*

```
XECUTE FT(<код операции>,<X>,<Y>)
```

*Описание.* Эта функция формирует или стирает на экране точку с координатами (<*X*>, <*Y*>) в зависимости от *кода операции:* 0 — стирание точки; 1 — формирование точки.

Координаты X и Y могут быть арифметическими выражениями, десятичные значения которых должны удовлетворять следующим условиям: $0 \le X \le 511$; $0 \le Y \le 239$, и отсчитываются относительно левого верхнего угла экрана, имеющего координаты (0, 0), причем положительное направление оси X — слева направо, а оси Y — сверху вниз.

Функция FT принимает десятичное значение координаты Y.

*Пример:*

```
*1.10 COMMENT ЗАПОЛНЕНИЕ ЭКРАНА СЛУЧАЙНЫМИ
*1.11 COMMENT ЧИСЛАМИ
*1.15 XECUTE FCHR(12); COMMENT ОЧИСТКА ЭКРАНА
*1.20 SET X=[FRAN()+1]*511
*1.30 SET Y=[FRAN()+1]*239
*1.40 XECUTE FT(1,X,Y); GOTO 1.2
```

*Связанные функции:* FV.

*Тангенс* FTAN.

*Синтаксис:*

```
SET <переменная>=FTAN(<выражение>)
```

*Описание.* Эта функция вычисляет тангенс угла, заданного в радианах; для преобразования из градусов в радианы используется множитель $\pi/180$. Значение выражения должно принадлежать интервалу $(-\pi/2, +\pi/2)$. При попытке вычислить тангенсы 90° ($\pi/2$

радиан) и 270° (—π/2 радиан), равные соответственно +∞ и —∞, произойдет потеря значимости, но сообщение об ошибке не появится.

*Пример:*

```
*1.10 ASK "ЗАДАЙТЕ УГОЛ В РАДИАНАХ",A
*1.20 SET B=FTAN(A*3.141592/180)
*1.30 TYPE "ТАНГЕНС ",A," РАВЕН ",B,!
```

*Связанные функции:* FSIN, FCOS, FASIN, FACOS, FATAN.

*Формирование вектора* FV.

*Синтаксис:*

```
XECUTE FV(<код операции>,<X>,<Y>)
```

*Описание.* В зависимости от *кода операции* эта функция формирует или стирает на экране вектор с координатами *конца вектора* (<X>, <Y>); *код операции:* 0 — стирание вектора; 1 — формирование вектора.

Координаты конца вектора X и Y могут быть арифметическими выражениями, десятичные значения которых должны удовлетворять следующим условиям: $0 \le X \le 511$; $0 \le Y \le 239$, и отсчитываются относительно левого верхнего угла экрана, имеющего координаты (0, 0), причем положительное направление оси *X* — слева направо, а оси *Y* — сверху вниз. Координаты начала вектора либо предварительно задаются с помощью функции FT, либо равны координатам конца ранее построенного вектора. Функция FV принимает десятичное значение координаты Y.

*Пример:*

```
*1.10 COMMENT ПОСТРОЕНИЕ КВАДРАТА
*1.20 XECUTE FV(0,100,100)
*1.30 XECUTE FV(1,100,200); FV(1,200,200)
*1.40 XECUTE FV(1,200,100); FV(1,100,100)
```

*Связанные функции:* FT.

*Управление общей шиной* FX.

*Синтаксис:*

```
SET <переменная>=FX(<код операции),<адрес>
      [,<выражение>]
```

*Описание.* Эта функция используется для организации работы с периферийными устройствами и обращения к ячейкам памяти компьютера.

*Код операции* определяет тип выполняемой операции:

1 — запись значения выражения по адресу общей шины (функция FX принимает десятичное значение записываемой величины); 0 — операция ЛОГИЧЕСКОЕ И» над содержимым ячейки по адресу общей шины и значением выражения (функция FX принимает десятичное значение результата операции); 1 — чтение слова по адресу общей шины (третий аргумент может быть опущен; функция FX принимает десятичное значение прочитанного слова).

*Адрес* общей шины представляет собой *восьмеричное число* или идентификатор переменной и должен быть *четным.* Он указывает адрес ячейки памяти или регистра периферийного устройства, к которому производится обращение. Значение выражения должно находиться в диапазоне от —32768 до +32767. Запись в ячейки памяти с адресами от 0 до $2000_8$ запрещена!

*Диагностические сообщения:* код ошибки 13 — нечетный адрес или попытка записи в запрещенную область; код ошибки 26 — обращение по несуществующему адресу общей шины.

*Пример.* Программа опрашивает 7-й бит (бит готовности) регистра состояния клавиатуры БК (адрес общей шины 177660) и при его наличии выводит на экран символ, код которого читается из регистра данных клавиатуры адрес общей шины 177662):

```
*1.10 IF [FX(0,177660,128)]1.10,1.20,1.10
*1.20 SET X=FX(1,177662)
*1.30 XECUTE FCHR(X); TYPE !
```

*Условная передача управления* IF.

*Синтаксис:*

```
I[F] (<выражение>)<mm.nn>[,<kk.ll>[,<ii.jj>]]
```

*Описание.* Этот оператор используется для передачи управления на строку программы в зависимости от значения арифметического выражения. Если значение выражения *меньше* нуля, то управление передается на строку с номером <*mm.nn*>; если значение выражения равно нулю, то на строку с номером <*kk.ll*>; если значение выражения больше нуля, то на строку с номером

<ii.jj>.

*Диагностическое сообщение:* код ошибки 5 — несуществующий *номер строки.*

*Пример:*

```
*1.10 ASK "ВВЕДИТЕ ДВА ЧИСЛА",A,B
*1.20 IF (A-B)1.3,1.4,1.5
*1.30 TYPE A," МЕНЬШЕ ",B,!; QUIT
*1.40 TYPE A," РАВНО ",B,!; QUIT
*1.50 TYPE A," БОЛЬШЕ ",B,!; QUIT
```

*Безусловная передача управления* GOTO.

*Синтаксис:*

```
G[OTO] [<номер строки>]
```

*Описание.* Этот оператор используется как для передачи управления на строку о указанным номером, так и для запуска программы. *Номер строки* может быть либо десятичным числом от 01.01 до 99.99 с шагом 0.01 или от 100.1 до 127.9 с шагом 0.1, или переменной. Если номер строки опущен, то управление передается на строку с наименьшим номером. Операторы, расположенные в строке за оператором GOTO, никогда не выполняются и могут рассматриваться как комментарии.

*Диагностическое сообщение:* код ошибки 5 — несуществующий *номер строки.*

*Пример:*

```
*1.10 ASK "ВВЕДИТЕ ЧИСЛО МЕНЬШЕЕ 10",A
*1.20 IF (10-A)1.4,1.4; TYPE "ВЕРНО",!
*1.25 GOTO 1.1
*1.40 TYPE "НЕВЕРНО",!; GOTO 1.1
```

*Связанные операторы:* DO.

*Работа с магнитофоном* LIBRARY.

*Синтаксис:*

```
L[IBRARY] S[AVE] <имя файла> | G[ET] <имя
      файла> | F[GET] <имя файла> |
O[UTPUT] <имя файла данных> |
```

**I[NPUT] <имя файла данных> | M[OTOR] | R[ESET]**

*Описание.* Операторы группы LIBRARY предназначены для управления обменом информацией между компьютером и магнитофоном как внешним запоминающим устройством.

Оператор LIBRARY SAVE выводит текст находящейся в памяти программы в файл с указанным именем; оператор LIBRARY GET загружает в память программу из указанного файла (если в памяти находилась программа, то она будет уничтожена); оператор LIBRARY FGET просматривает магнитную ленту и выводит на экран имена встретившихся на ней файлов; оператор LIBRARY OUTPUT выводит находящуюся в памяти компьютера таблицу переменных в указанный файл; оператор LIBRARY INPUT заносит данные в таблицу переменных, считывая их из указанного файла; оператор LIBRARY MOTOR включает, а LIBRARY RESET выключает двигатель магнитофона (эти операторы будут выполняться только на магнитофоне, который допускает дистанционное управление двигателем).

*Имя файла* (в том числе файла *данных)* может состоять из любых алфавитно-цифровых символов, число которых не должно превышать 15.

*Диагностическое сообщение:* код ошибки 21 — неверное задание имени файла; код ошибки 22 — ошибка контрольной суммы при чтении файла; код ошибки 23 — ошибка по длине файла.

*Пример:*

```
*C ВЫВОДИТ НА ЭКРАН СПИСОК ФАЙЛОВ
*XECUTE LIBRARY FGET AAA
```

*Редактирование* MODIFY.

*Синтаксис:*

**M[ODIFY] <номер строки>**

*Описание.* Этот оператор используется для редактирования (т.е. замены, вставки или удаления символов) строки программы.

Оператор MODIFY может задаваться только в директивной строке и должен быть в ней последним оператором. После ввода оператора MODIFY он распечатывает строку с указанным *номером*

*строки* (сам номер не выводится). Выведенный текст можно редактировать с помощью клавиш (→, ←, ВС и ГТ), управляющих перемещением курсора, и клавиш (|—>, <—|, <-/- и СБР|-->), обеспечивающих удаление символов, а также сдвижку и раздвижку строки. Редактирование можно завершить в любой момент независимо от положения курсора, нажав клавишу ВВОД. Операция MODIFY не стирает таблицу переменных и не может использоваться для редактирования номера строки.

*Диагностическое сообщение:* код ошибки 5 — несуществующий *номер строки.*

*Останов* QUIT.

*Синтаксис:*

**Q[UIT]**

*Описание.* Этот оператор прекращает выполнение программы и переводит интерпретатор ФОКАЛа-БК в режим задания директивных строк. Оператор QUIT всегда последний выполняемый оператор в программе. Завершение программы оператором QUIT является «хорошим тоном» с точки зрения методики программирования.

*Пример:*

```
*1.10 DO 2
*1.20 QUIT
*2.10 TYPE "ЭТА СТРОКА НАПЕЧАТАЛАСЬ БЫ "
*2.15 TYPE "ДВАЖДЫ, ЕСЛИ БЫ НЕ БЫЛО СТРОКИ",!
*2.20 WRITE 1.2
```

*Связанные операторы:* GOTO.

*Выход из подпрограммы* RETURN.

*Синтаксис:*

**R[ETURN]**

*Описание.* Этот оператор используется для выхода из подпрограммы, т. е. для возврата к оператору следующему за оператором DO или за оператором, содержащим обращение к функции FSBR. Операторы, следующие за оператором RETURN в той же строке, никогда не выполняются, поэтому за оператором

RETURN целесообразно помещать только комментарии.

*Пример:*

```
*1.10 COMMENT ПРОГРАММА, РЕАЛИЗУЮЩАЯ ФУНКЦИЮ
*1.11 COMMENT FSGN
*1.20 ASK "ЗАДАЙТЕ ЧИСЛО",A; DO 2
*1.30 TYPE %2; QUIT
*2.10 IF (A)2.2,2.3; SET X=1; RETURN
*2.20 SET X=-1; RETURN
*2.30 SET X=0; С ЗДЕСЬ RETURN МОЖНО БЫЛО НЕ
*2.31 С УКАЗЫВАТЬ
```

*Связанные операторы и функции:* DO, FSBR.

*Присваивание* SET.

*Синтаксис:*

```
S[ET] <переменная>=<выражение>
```

*Описание.* Этот оператор присваивает значение арифметического выражения *переменной.*

*Диагностическое сообщение,* код ошибки 7 — неправильный формат оператора SET.

*Пример:*

```
*1.10 ASK "ЗАДАЙТЕ РАДИУС ОКРУЖНОСТИ",R
*1.20 SET PI=3.141592; SET L=2*PI*R
*1.30 TYPE "ДЛИНА ОКРУЖНОСТИ РАВНА ",L,!
```

*Вывод* TYPE.

*Синтаксис:*

```
T[YPE] <элемент вывода>{,<элемент вывода>}

<элемент вывода>:== <переменная>|<выражение>|
      "<текст>" | ! | ¤ | %[M.0N]
```

*Описание.* Этот оператор предназначен для вывода чисел, значений переменных и арифметических выражений, строк текста и таблицы переменных (¤). В операторе TYPE допускается произвольное сочетание перечисленных элементов вывода и управляющих символов (знака % для задания формата вывода

чисел и знака ! для организации строк вывода). Оператор TYPE ¤ — последний выполняемый оператор в строке.

*Пример:*

```
*1.10 TYPE "1 ДЮЙМ = ",%5.02,2.54," СМ",!
*1.20 TYPE "1 М = ",%3,100,” СМ",!
```

*Вывод текста программы* WRITE.

*Синтаксис:*

```
W[RITE] [<номер группы> | <номер строки> |
     A[LL]]
```

*Описание.* Этот оператор предназначен для вывода строки, группы строк или текста *всей* (ALL) программы на экран. Оператор WRITE, как и оператор WRITE ALL, выводит текст всей программы, задается, как правило, в директивной строке и используется для просмотра текста всей программы или ее части.

*Пример:*

```
*WRITE 5; С ВЫВОДИТ ПЯТУЮ ГРУППУ СТРОК
```

*Выполнение* XECUTE.

*Синтаксис:*

```
X[ECUTE] <выражение>
```

*Описание.* Этот оператор вычисляет значение арифметического *выражения* или, как частный случай выражения, значение стандартной функции. Как правило, оператор XECUTE используется в тех случаях, когда пользователя интересует не значение, которое принимает функция, а выполняемая ею операция ввода — вывода или управления.

*Пример:*

```
*1.10 COMMENT ВЫВОД СИМВОЛОВ А И В
*1.20 XECUTE FCHR(65,66)
```

ПРИЛОЖЕНИЕ 2

Таблица 1

**Коды ошибок и диагностические сообщения ФОКАЛа-БК**

| Код ошибки | Текстовое сообщение |
|---|---|
| 00 | Готовность к работе |
| 01 | Неправильный номер строки |
| 02 | Неправильное имя функции или переменной |
| 03 | Непарные скобки |
| 04 | Неправильная команда |
| 05 | Несуществующий номер строки |
| 06 | Несуществующий номер группы или номер строки в операторе DO |
| 07 | Неправильный формат SET или FOR |
| 08 | Двойной или отсутствующий оператор в выражении |
| 09 | Переполнение стека |
| 10 | Переполнение памяти текстом программы |
| 11 | Нет места для переменных |
| 12 | Порядок больше 38 |
| 13 | Запрещенный адрес шины функции FX |
| 14 | Попытка деления на нуль |
| 15 | Попытка возведения в отрицательную или слишком большую степень |
| 16 | Слишком много символов во входных данных |
| 17 | Корень квадратный из отрицательного числа |
| 18 | Переполнение входного буфера |
| 19 | Логарифм нуля или отрицательного числа |
| 20 | В функциях FASIN или FACOS аргумент по модулю больше единицы |
| 21 | Ошибка в имени файла |
| 22 | Ошибка контрольной суммы |
| 23 | Ошибка по длине файла |
| 24 | Останов операций магнитофона по прерыванию опера тора |
| 25 | Останов по клавише СТОП |
| 26 | Несуществующее устройство |
| 27 | Неправильный код операции в функции FP |

ПРИЛОЖЕНИЕ 3
Таблица 2

**Коды и символы клавиатуры БК-0010**

| Код | | Маркировка на клавиатуре БК-0010 | Назначение клавиш (или десятичного кода) БК-0010 |
|---|---|---|---|
| десятичный | восьмеричный | | |
| | | Управляющие символы и символы редактирования; адрес вектора прерывания 60 | |
| 8 | 10 | ← | Перевод курсора на одну позицию влево |
| 10 | 12 | ВВОД | Ввод строки |
| 12 | 14 | СБР | Очистка экрана |
| 13 | 15 | УСТ.ТАБ | ФОКАЛ не поддерживает; в драйвере — установка позиций табуляции * |
| 14 | 16 | РУС | Переключение на регистр РУС * |
| 15 | 17 | ЛАТ | Переключение на регистр ЛАТ * |
| 16 | 20 | СБР.ТАБ | ФОКАЛ не поддерживает; в драйвере — сброс табулируемой позиции * |
| 18 | 22 | ↖ | Исходная установка курсора |
| 19 | 23 | ВС | Перевод курсора в начало текущей строки |
| 20 | 24 | ГТ | Перевод курсора на восемь позиций вправо |
| 21 | 25 | Клавиши нет | Код 21 используется в функции FCHR для перевода курсора в начало следующей строки |
| 22 | 26 | ⇤ | Сдвижка в строке |
| 23 | 27 | ⇥ | Раздвижка строки |
| 24 | 30 | ⇤+ | Удаление последнего символа в строке |
| 25 | 31 | → | Перемещение курсора на одну позицию по направлению стрелки 2* |
| 26 | 32 | ↑ | |
| 27 | 33 | ↓ | |
| 28 | 34 | ↖ | |
| 29 | 35 | ↗ | |
| 30 | 36 | ↘ | |
| 31 | 37 | ↙ | |

| Код | | Маркировка на клавиатуре БК-0010 | Назначение клавиш (или десятичного кода) БК-0010 |
|---|---|---|---|
| десятичный | восьмеричный | | |
| | | Символы; адрес вектора прерывания 60 | |
| 32 | 40 | | Пробел |
| 33 | 41 | ! (ПР) | Восклицательный знак |
| 34 | 42 | » (ПР) | Кавычки |
| 35 | 43 | # (ПР) | Номер |
| 36 | 44 | ¤ (ПР) | Знак денежной единицы |
| 37 | 45 | % (ПР) | Процент |
| 38 | 46 | & (ПР) | Амперсанд |
| 39 | 47 | ' (ПР) | Апостроф |
| 40 | 50 | ( (ПР) | Левая круглая скобка |
| 41 | 51 | ) (ПР) | Правая круглая скобка |
| 42 | 52 | * (ПР) | Звездочка |
| 43 | 53 | + (ПР) | Плюс |
| 44 | 54 | , | Запятая |
| 45 | 55 | — | Минус |
| 46 | 56 | . | Точка |
| 47 | 57 | / | Дробная черта |
| 48 | 60 | 0 | Цифры; адрес вектора прерывания 60 |
| 49 | 61 | 1 | |
| 50 | 62 | 2 | |
| 51 | 63 | 3 | |
| 52 | 64 | 4 | |
| 53 | 65 | 5 | |
| 54 | 66 | 6 | |
| 55 | 67 | 7 | |
| 56 | 70 | 8 | |
| 57 | 71 | 9 | |
| | | Символы; адрес вектора прерывания 60 | |
| 58 | 72 | : | Двоеточие |
| 59 | 73 | ; | Точка с запятой |
| 60 | 74 | < (ПР) | Меньше |
| 61 | 75 | = (ПР) | Равно |
| 62 | 76 | > (ПР) | Больше |
| 63 | 77 | ? (ПР) | Вопросительный знак |
| | | Прописные буквы латинского алфавита; адрес вектора прерывания 60 (ЛАТ, ЗАГЛ) | |
| 64 | 100 | @ | Коммерческое «а» |
| 65 | 101 | *A* | Буквы |
| 66 | 102 | *B* | |
| 67 | 103 | *C* | |

| Код | | Маркировка на клавиатуре БК-0010 | Назначение клавиш (или десятичного кода) БК-0010 |
|---|---|---|---|
| десятичный | восьмеричный | | |
| 68 | 104 | *D* | |
| 69 | 105 | *E* | |
| 70 | 106 | *F* | |
| 71 | 107 | *G* | |
| 72 | 110 | *H* | |
| 73 | 111 | *I* | |
| 74 | 112 | *J* | |
| 75 | 113 | *K* | |
| 76 | 114 | *L* | |
| 77 | 115 | *M* | |
| 78 | 116 | *N* | |
| 79 | 117 | *O* | |
| 80 | 120 | *P* | |
| 81 | 121 | *O* | |
| 82 | 122 | *R* | |
| 83 | 123 | *S* | |
| 84 | 124 | *T* | |
| 85 | 125 | *U* | |
| 86 | 126 | *V* | |
| 87 | 127 | *W* | |
| 88 | 130 | *X* | |
| 89 | 131 | *Y* | |
| 90 | 132 | *Z* | |
| Символы; адрес вектора прерывания 60 | | | |
| 91 | 133 | [ | Левая квадратная скобка |
| 92 | 134 | / | Обратная дробная черта |
| 93 | 135 | ] | Правая квадратная скобка |
| 94 | 136 | ^ | Возведение в степень |
| 95 | 137 | _ | Подчеркивание |
| 96 | 140 | @ (ЛАТ, СТР) | Слабое ударение (') |
| Строчные буквы латинского алфавита; адрес вектора прерывания 60 (ЛАТ, СТР) | | | |
| 97 | 141 | *a* | |
| 98 | 142 | *b* | |
| 99 | 143 | *c* | |
| 100 | 144 | *d* | |
| 101 | 145 | *e* | |
| 102 | 146 | *f* | |
| 103 | 147 | *a* | |
| 104 | 150 | *h* | |
| 105 | 151 | *i* | |
| 106 | 152 | *i* | |

| Код | | Маркировка на клавиатуре БК-0010 | Назначение клавиш (или десятичного кода) БК-0010 |
|---|---|---|---|
| десятичный | восьмеричный | | |
| 107 | 153 | *k* | |
| 108 | 154 | *l* | |
| 109 | 155 | *m* | |
| 110 | 156 | *n* | |
| 111 | 157 | *o* | |
| 112 | 160 | *p* | |
| 113 | 161 | *q* | |
| 114 | 162 | *r* | |
| 115 | 163 | *s* | |
| 116 | 164 | *t* | |
| 117 | 165 | *u* | |
| 118 | 166 | *v* | |
| 119 | 167 | *w* | |
| 120 | 170 | *x* | |
| 121 | 171 | *v* | |
| 122 | 172 | *z* | |
| Символы; адрес вектора прерывания 60 (ЛАТ, СТР) | | | |
| 123 | 173 | { | Левая фигурная скобка |
| 124 | 174 | \| | Вертикальная черта |
| 125 | 175 | } | Правая фигурная скобка |
| 126 | 176 | ~ | Надчеркивание |
| 127 | 177 | ЗБ | Забой |
| Управляющие символы; адрес вектора прерывания 274 | | | |
| 129 | 201 | ПОВТ | Многократная выдача ранее введенного символа * [4]* |
| 130 | 202 | ИНД СУ | Режим индикации управляющих символов [3]* |
| 132 | 204 | БЛОК | Блокировка редактирующих функций [3]* |
| 137 | 211 | ТАБ | ФОКАЛ не поддерживает; в драйвере — перевод курсора к следующей табулируемой позиции * |
| 140 | 214 | РП | ФОКАЛ не поддерживает; в драйвере — режим «расширенная память» [3]* |
| 144 | 220 | ШАГ | Управляющий код |
| 145 | 221 | ! (НР, ПР) | Управление яркостью; включение первой градации яркости (красный) |
| 146 | 222 | « (НР, ПР) | Включение второй градации яркости (зеленый) |

| Код | | Маркировка па клавиатуре БК-0010 | Назначение клавиш (или десятичного кода) БК-0010 |
|---|---|---|---|
| десятичный | восьмеричный | | |
| Управляющие символы; адрес вектора прерывания 274 | | | |
| 147 | 223 | # (НР, ПР) | Включение третьей градации яркости (синий) |
| 148 | 224 | ¤ (НР, ПР) | Включение четвертой градации яркости (черный) |
| 149 | 225 | ГРАФ | ФОКАЛ не поддерживает; в драйвере — перевод в графический режим |
| 150 | 226 | ЗАП | ФОКАЛ не поддерживает; в драйвере — включение режима записи в графическом режиме |
| 151 | 227 | СТИР | ФОКАЛ не поддерживает; в драйвере — включение режима стирания в графическом режиме |
| 152 | 230 | РЕД (НР) | Включение режима редактирования строки |
| 153 | 231 | СБР\|—> | Стирание правой от курсора части строки |
| 154 | 232 | КУРСОР (НР) | Переключение индикации курсора |
| 155 | 233 | 32/64 (НР) | Установка числа символов в строке |
| 156 | 234 | ИНВ.С (НР) | Установка режима негативной индикации символов |
| 157 | 235 | ИНВ.Э. (НР) | Установка режима негативной индикации экрана |
| 158 | 236 | УСТ.ИНД. (НР) | Установка режимов формирования индикаторов в служебной строке |
| 159 | 237 | ПОДЧ (НР) | Включение режима подчеркивания символов |
| Символы табличной графики и дополнительные символы; адрес вектора прерывания 274 (НР) | | | |
| 160 | 240 |  | Графические знаки |
| 161 | 241 |  | |
| 162 | 242 |  | |
| 163 | 243 |  | |
| 164 | 244 |  | |

| Код | | Маркировка на клавиатуре БК-0010 | Назначение клавиш (или десятичного кода) БК-0010 |
|---|---|---|---|
| десятичный | восьмеричный | | |
| 165 | 245 |  | |
| 166 | 246 |  | |
| 167 | 247 |  | |
| 168 | 250 |  | |
| 169 | 251 |  | |
| 170 | 252 |  | |
| 171 | 253 |  | |
| 172 | 254 |  | |
| 173 | 255 |  | |
| 174 | 256 |  | |
| 175 | 257 |  | |
| 176 | 260 |  | |
| 177 | 261 |  | |
| 178 | 262 |  | |
| 179 | 263 |  | |

| Код | | Маркировка на клавиатуре БК-0010 | Назначение клавиш (или десятичного кода) БК-0010 |
|---|---|---|---|
| десятичный | восьмеричный | | |
| 180 | 264 |  | |
| 181 | 265 |  | |
| 182 | 266 |  | |
| 183 | 267 |  | |
| 184 | 270 |  | |
| 185 | 271 |  | |
| 186 | 272 |  | |
| 187 | 273 |  | |
| 188 | 274 |  | |
| 189 | 275 |  | |
| 190 | 276 |  | |
| 191 | 277 |  | |

| Код | | Маркировка на клавиатуре БК-0010 | Назначение клавиш (или десятичного кода) БК-0010 |
|---|---|---|---|
| десятичный | восьмеричный | | |
| | | Строчные символы русского алфавита; адрес вектора прерывания 60 (РУС, СТР) | |
| 192 | 300 | ю | Буквы |
| 193 | 301 | а | |
| 194 | 302 | б | |
| 195 | 303 | ц | |
| 196 | 304 | д | |
| 197 | 305 | е | |
| 198 | 306 | ф | |
| 199 | 307 | г | |
| 200 | 310 | х | |
| 201 | 311 | и | |
| 202 | 312 | й | |
| 203 | 313 | к | |
| 204 | 314 | л | |
| 205 | 315 | м | |
| 206 | 316 | н | |
| 207 | 317 | о | |
| 208 | 320 | п | |
| 209 | 321 | я | |
| 210 | 322 | р | |
| 211 | 323 | с | |
| 212 | 324 | т | |
| 213 | 325 | у | |
| 214 | 326 | ж | |
| 215 | 327 | в | |
| 216 | 330 | ь | |
| 217 | 331 | ы | |
| 218 | 332 | з | |
| 219 | 333 | ш | |
| 220 | 334 | э | |

| Код | | Маркировка на клавиатуре БК-0010 | Назначение клавиш (или десятичного кода) БК-0010 |
|---|---|---|---|
| десятичный | восьмеричный | | |
| 221 | 335 | щ | |
| 222 | 336 | ч | |
| 223 | 337 | ЗБ (ъ) | |
| | | Прописные символы русского алфавита; адрес вектора прерывания 60 (РУС, ЗАГЛ) | |
| 224 | 340 | Ю | Буквы |
| 225 | 341 | А | |
| 226 | 342 | Б | |
| 227 | 343 | Ц | |
| 228 | 344 | Д | |
| 229 | 345 | Е | |
| 230 | 346 | Ф | |
| 231 | 347 | Г | |
| 232 | 350 | Х | |
| 233 | 351 | И | |
| 234 | 352 | Й | |
| 235 | 353 | К | |
| 236 | 354 | Л | |
| 237 | 355 | М | |
| 238 | 356 | Н | |
| 239 | 357 | О | |
| 240 | 360 | П | |
| 241 | 361 | Я | |
| 242 | 362 | Р | |
| 243 | 363 | С | |
| 244 | 364 | Т | |
| 245 | 365 | У | |
| 246 | 366 | Ж | |
| 247 | 367 | В | |
| 248 | 370 | Ь | |
| 249 | 371 | Ы | |

| Код | | Маркировка на клавиатуре БК-0010 | Назначение клавиш (или десятичного кода) БК-0010 |
|---|---|---|---|
| десятичный | восьмеричный | | |
| 250 | 372 | З | Буквы |
| 251 | 373 | Ш | |
| 252 | 374 | Э | |
| 253 | 375 | Щ | |
| 254 | 376 | Ч | |
| 255 | 377 | ЗБ (Ъ) | |

Примечание. В скобках указаны клавиши, которые должны быть нажаты для выработки требуемого кода.

---

* Коды используются только в драйвере клавиатуры.

[2]* Назначение, указанное в колонке 4, справедливо только в режиме ФОКАЛа «РЕД».

[3]* Коды передаются из драйвера клавиатуры в драйвер ТВ, но не передаются в программу.

[4]* Десятичный код и прорисовка символа соответствуют коду и прорисовке последнего введенного символа.

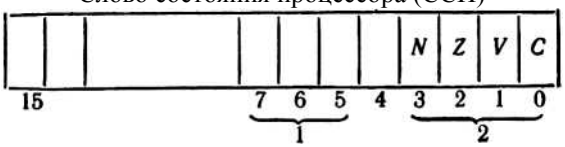
## Система машинных команд ПК БК-0010

Таблица 4

**Символы и сокращения, используемые при описании команд**

| Символ | Значение |
|---|---|
| *c* | Код условия (двоичный) |
| *c* (...) | Содержимое ... |
| *C* | Код условия (восьмеричный) или бит С в ССП |
| *d* | Приемник (двоичный) |
| *dst* | » (регистр или ячейка памяти) |
| *D* | » (восьмеричный) |
| *loc* | Ячейка |
| *n* | Число (двоичное) |
| *N* | » (восьмеричное) или бит *N* в ССП |
| СК | Программный счетчик (счетчик команд) |
| ССП | Слово состояния процессора |
| *r* | Регистр (двоичный) |
| *reg* | » |
| *R* | » (восьмеричный) |
| *s* | Источник (двоичный) |
| *src* | » (регистр, ячейка памяти или число) |
| *S* | » (восьмеричный) |
| *V* | Бит *V* в ССП |
| *x* | Пословное смещение (двоичное) |
| *X* | » » (восьмеричное) |
| *Z* | Бит Z в ССП |
| ↓ (*УС*) | Протолкнуть в аппаратный стек |
| (*УС*) ↑ | Вытолкнуть из аппаратного стека |
| ← ... | Придается значение ... |
| /\ | Логическое И |
| \/ | » ИЛИ |
| $\underline{\vee}$ | » ИСКЛЮЧИТЕЛЬНОЕ ИЛИ |
| ~ | » НЕ |

Слово состояния процессора (ССП)




1 — приоритет ЦП; 2 — коды условий

*ADC* ПРИБАВИТЬ ПЕРЕНОС К ПРИЕМНИКУ 0055*DD*

Двоичный код: 0 000 101 101 *ddd ddd*
Операция: *dst* ← с (*dst*) + *c (C).*
Формат
ассемблера: *ADC dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — устанавливается, если *c (dst)* равно 077777 и *c (C)* равно единице, иначе сбрасывается;
*C* — устанавливается, если *c* (*dst*) равно 177777 и *c* (*C*) равно единице, иначе сбрасывается.
Описание: прибавляет к приемнику содержимое бита *C.*

*ADCB* ПРИБАВИТЬ ПЕРЕНОС К ПРИЕМНИКУ—БАЙТУ 1055*DD*

Двоичный код: 1 000 101 101 *ddd ddd*
Операция: *dst* ← с *(dst) + c (C)*
Формат
ассемблера: *ADCB dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — устанавливается, если *c (dst)* равно 177 и *c (C)* равно единице, иначе сбрасывается;
*C* — устанавливается, если *c (dst)* равно 377 и *c (C)* равно единице, иначе сбрасывается.
Описание: прибавляет к приемнику (байту) содержимое бита *C.*

*ADD* ПРИБАВИТЬ ИСТОЧНИК К ПРИЕМНИКУ 06*SSDD*

Двоичный код: 0 110 *sss sss ddd ddd*
Операция: *dst ← c (src) + c (dst)*
Формат

ассемблера: *ADD src, dst*

Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — устанавливается, если происходит переполнение (оба операнда имели один и тот же знак, а результат — противоположный знак), иначе сбрасывается; *C* — устанавливается, если произошел перенос из бита 15, иначе сбрасывается.

Описание: прибавляет содержимое источника к содержимому приемника и помещает результат в приемник; первоначальное содержимое приемника теряется; содержимое источника остается без изменений.

*ASL* АРИФМЕТИЧЕСКИ СДВИНУТЬ ПРИЕМНИК ВЛЕВО 0063*DD*

Двоичный код: 0 000 110 011 *ddd ddd*

Операция: *dst* ← *c(dst)* сдвинутые на один бит влево.

Формат ассемблера: *ASL dst*

Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — загружается результатом операции ИСКЛЮЧАЮЩЕЕ ИЛИ над *N* и *C* (определенными *после* сдвига);
*C* — загружается значение бита, сдвинутого из самого старшего бита приемника.

Описание: каждый бит слова приемника сдвигается на одну позицию влево; самый старший бит (бит 15) сдвигается в бит *C* в ССП; самому младшему биту (биту 0) присваивается значение нуль.
Эта инструкция выполняет умножение на два приемника с учетом знака. Если после сдвига устанавливается бит *V*, то в приемнике произошло изменение знака.

*ASLB* АРИФМЕТИЧЕСКИ СДВИНУТЬ БАЙТ—ПРИЕМНИК ВЛЕВО 1063*DD*

Двоичный код: 1 000 110 011 *ddd ddd*
Операция: *dst* ← *c (dst)* сдвинутое на один бит влево.
Формат
ассемблера: *ASLB dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — загружается результатом операции ИСКЛЮЧАЮЩЕЕ ИЛИ над *N* и *C* (определенными *после* сдвига);
*C* — загружается значением бита, сдвинутого из самого старшего бита приемника.
Описание: каждый бит байта-приемника сдвигается на одну позицию влево; самый старший бит (бит 7 или 15) сдвигается в бит *C* в ССП; освободившемуся самому младшему биту (биту 0 или 8) присваивается значение 0.

*ASR* АРИФМЕТИЧЕСКИ СДВИНУТЬ ПРИЕМНИК ВПРАВО
0062*DD*

Двоичный код: 0 000 110 010 *ddd ddd*
Операция: *dst* ← *c (dst),* сдвинутое на один бит вправо.
Формат
ассемблера: *ASR dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — загружается результатом операции ИСКЛЮЧАЮЩЕЕ ИЛИ над *N* и *C* (определенными *после* сдвига);
*C* — загружается значением бита, сдвинутого из самого младшего бита приемника.
Описание: каждый бит приемника сдвигается на одну позицию вправо; самый старший бит (бит 15) воспроизводится; самый младший бит (бит 0) сдвигается в бит *C* в ССП. Эта инструкция выполняет деление на два приемника с учетом знака.

*ASRB* АРИФМЕТИЧЕСКИ СДВИНУТЬ БАЙТ—ПРИЕМНИК ВПРАВО 1062*DD*

Двоичный код: 1 000 110 010 *ddd ddd*
Операция: *dst* ← *c (dst),* сдвинутые на один бит вправо.
Формат
ассемблера: *ASRB dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — загружается результатом операции ИСКЛЮЧАЮЩЕЕ ИЛИ над *N* и *C* (определенными *после* сдвига);
*C* — загружается значением бита, сдвинутого из самого младшего бита приемника.
Описание: каждый бит байта-приемника сдвигается на одну позицию вправо; самый старший бит (бит 7 или 15) воспроизводится; самый младший бит (бит 0 или 8) сдвигается в бит *C* в ССП.

*BCC* ВЕТВЛЕНИЕ, ЕСЛИ ПЕРЕНОС СБРОШЕН 1030*XXX*

Двоичный код: 1 000 011 0*xx xxx xxx*
Операция: СК ← *c* (СК) + 2·(смещение), если *C* = 0
Формат
ассемблера: *BCC loc*
Коды условий: не меняются.
Описание: младший байт инструкции - это пословное смещение со знаком от текущего СК к целевому адресу *loc*;
если *C* = 0, то удвоенное смещение прибавляется к *c* (СК); если *C* = 1, то инструкция игнорируется.

*BCC* ВЕТВЛЕНИЕ, ЕСЛИ ПЕРЕНОС УСТАНОВЛЕН 1034*XXX*

Двоичный код: 1 000 011 1*xx xxx xxx*
Операция: СК ← *c* (СК) + 2·(смещение), если *C* = 1
Формат
ассемблера: *BCS loc*
Коды условий: не изменяются.
Описание: младший байт инструкции — это *пословное*

*смещение*

со знаком от текущего СК к целевому адресу *loc*; если *C* = 1, то удвоенное смещение прибавляется к *c* (СК); если *C* = 0, то инструкция игнорируется.

*BEQ* ВЕТВЛЕНИЕ, ЕСЛИ РАВНО НУЛЮ 0014*XXX*

Двоичный код: 0 000 001 1*xx xxx xxx*
Операция: СК ← *c* (СК) + 2·(смещение), если *Z*= 1.
Формат
ассемблера: *BEQ loc*
Коды условий: не изменяются.
Описание: младший байт инструкции — это *пословное смещение* со знаком от текущего СК к целевому адресу *loc*; если Z = 1, то удвоенное смещение прибавляется к *c* (СК); если *Z* = 0, то инструкция игнорируется.

*BGE* ВЕТВЛЕНИЕ, ЕСЛИ БОЛЬШЕ ЧЕМ ИЛИ РАВНО НУЛЮ 0020*XXX*

Двоичный код: 0 000 010 0*xx xxx xxx*
Операция: СК ← *c* (СК) + 2·(смещение), если $N \veebar V = 0$.
Формат
ассемблера: *BGE loc*
Коды условий: не изменяются.
Описание: младший байт инструкции — это пословное смещение со знаком от текущего СК к целевому адресу *loc*; если $N \veebar V = 0$, то удвоенное смещение прибавляется к *c* (СК); если $N \veebar V = 1$, то инструкция игнорируется (таким образом, эта инструкция, когда следует за сложением двух положительных чисел, вызывает ветвление, даже если результат сложения отрицательный).

*BGT* ВЕТВЛЕНИЕ, ЕСЛИ БОЛЬШЕ НУЛЯ 0030*XXX*

Двоичный код: 0 000 011 0*xx xxx xxx*
Операция: СК ← *c* (СК) + 2·(смещение), если $Z \vee (N \veebar V) = 0$.
Формат *BGT loc*

ассемблера:
Коды условий: не изменяются.
Описание: младший байт инструкции — это *пословное смещение* со знаком от текущего СК до целевого адреса *loc*; если $Z \vee (N \veebar V) = 0$, то удвоенное смещение прибавляется к *с* (СК); если $Z \vee (N \veebar V) = 1$, то инструкция игнорируется (эта инструкция ведет себя аналогично инструкции *BGE,* за исключением того, что она не вызывает ветвления при нулевом результате).

*BHI* ВЕТВЛЕНИЕ, ЕСЛИ ВЫШЕ 1010*XXX*

Двоичный код: 1 000 001 0*xx xxx xxx*
Операция: СК ← *с* (СК) + 2·(смещение), если $C \vee Z = 0$.
Формат
ассемблера: *BHI loc*
Коды условий: не изменяются.
Описание: младший байт инструкции — это пословное смещение со знаком от текущего СК к целевому адресу *loc*; если $C \vee Z = 0$, то удвоенное смещение прибавляется к *с* (СК); если $C \vee Z = 1$, то инструкция игнорируется (таким образом, инструкция *BHI,* следующая за инструкцией *CMP,* вызовет ветвление, если первое сравниваемое число имеет большее значение *без учета знака,* чем второе сравниваемое число).

*BHIS* ВЕТВЛЕНИЕ, ЕСЛИ ВЫШЕ ИЛИ ТО ЖЕ САМОЕ 1030*XXX*

Двоичный код: 1 000 011 0*xx xxx xxx*
Операция: СК ← *с* (*СК*) + 2·(смещение), если $C = 0$.
Формат
ассемблера: *BH1S loc*
Коды условий: не изменяются.
Описание: идентично инструкции *BCC*; включена в набор инструкций для удобства программиста (если инструкция *BHIS* следует за инструкцией *CMP,* то имеет место ветвление, когда первое сравниваемое число имеет значение без учета знака большее или

равное значению второго сравниваемого числа.)

*BIC* СБРОСИТЬ БИТЫ ПРИЕМНИКА ПО ИСТОЧНИКУ 04*SSDD*

Двоичный код: 0 100 *sss sss ddd ddd*
Операция: *dst* ← [~*c* (*src*)] /\ *c* (*dst*)
Формат ассемблера: *BIC src, dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — сбрасывается;
*C* — не изменяется.
Описание: в приемнике сбрасывается каждый бит, соответствующий единичному биту источника; содержимое источника не изменяется.

*BICB* СБРОСИТЬ БИТЫ БАЙТА-ПРИЕМНИКА ПО ИСТОЧНИКУ 14*SSDD*

Двоичный код: 1 100 *sss sss ddd ddd*
Операция: *dst* ← [~*c* (*src*)] /\ *c* (*dst*)
Формат ассемблера: *BICB src, dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — сбрасывается;
*C* — не изменяется.
Описание: в байте-приемнике сбрасывается каждый бит, соответствующий единичному биту в источнике; содержимое байта-источника не изменяется.

*BIS* УСТАНОВИТЬ БИТЫ ПРИЕМНИКА ПО ИСТОЧНИКУ 05*SSDD*

Двоичный код: 0 101 *sss sss ddd ddd*

Операция: *dst* ← *c* (*src*) ∨ *c* (*dst*)
Формат
ассемблера: *BIS src, dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — сбрасывается;
*C* — не изменяется.
Описание: содержимое приемника заменяется результатом операции ИЛИ над содержимым источника и первоначальным содержимым приемника; таким образом, в приемнике устанавливается каждый бит, соответствующий единичному биту в источнике; все остальные биты приемника не изменяются; содержимое источника не изменяется.

*BISB* УСТАНОВИТЬ БИТЫ БАЙТА-ПРИЕМНИКА ПО ИСТОЧНИКУ 15*SSDD*

Двоичный код: 1 101 *sss sss ddd ddd*
Операция: *dst* ← *c* (*src*) ∨ *c* (*dst*)
Формат
ассемблера: *BISB src, dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — сбрасывается;
*C* — не изменяется.
Описание: содержимое байта-приемника заменяется результатом операции ИЛИ над содержимым источника и первоначальным содержимым приемника; таким образом, в байте-приемнике устанавливается каждый бит, соответствующий единичному биту в источнике; все остальные биты приемника не изменяются, содержимое байта-источника не изменяется.

*BIT* ПРОВЕРИТЬ БИТЫ ИСТОЧНИКА И ПРИЕМНИКА 03*SSDD*

Двоичный код: 0 011 *sss sss ddd ddd*
Операция: *c* (*src*) /\ *c* (*dst*)
Формат
ассемблера: *BIT src, dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — сбрасывается;
*C* — не изменяется.
Описание: выполняет логическую операцию И над содержимым слов источника и приемника и соответственно устанавливает коды условий *N* и *Z*; ни содержимое источника, ни содержимое приемника не изменяются.

*BITB* ПРОВЕРИТЬ БИТЫ БАЙТОВ ИСТОЧНИКА И ПРИЕМНИКА 13*SSDD*

Двоичный код: 1 011 *sss sss ddd ddd*
Операция: *c* (*src*) /\ *c* (*dst*)
Формат
ассемблера: *BITB src, dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — сбрасывается;
*C* — не изменяется.
Описание: выполняет логическую операцию И над содержимым байтов источника и приемника и соответственно устанавливает коды условий *N* и *Z;* ни содержимое источника, ни содержимое приемника не изменяются.

*BLE* ВЕТВЛЕНИЕ, ЕСЛИ МЕНЬШЕ ЧЕМ ИЛИ РАВНО НУЛЮ 0034*XXX*

Двоичный код: 0 000 011 1*xx xxx xxx*
Операция: СК ← *c* (СК) + 2·(смещение), если $Z \vee (N \veebar V) = 1$.
Формат

ассемблера: *BLE loc*
Коды условий: не изменяются.
Описание: младший байт инструкции — это *пословное смещение* со знаком от текущего СК к целевому адресу *loc*; если $Z \vee (N \veebar V) = 1$, то удвоенное смещение прибавляется к *с* (СК); если $Z \vee (N \veebar V) = 0$, то инструкция игнорируется (поведение этой инструкции близко инструкции *BLT,* за исключением того, что *BLE* вызывает ветвление при нулевом результате).

*BLO* ВЕТВЛЕНИЕ, ЕСЛИ НИЖЕ 1034*XXX*

Двоичный код: 1 000 011 1*xx xxx xxx*
Операция: СК ← *с* (СК) + 2·(смещение), если $C = 1$.
Формат
ассемблера: *BLO loc*
Коды условий: не изменяются.
Описание: идентична инструкции *BCS*; включена в набор инструкций для удобства программиста; когда инструкция *BLO* следует за инструкцией *CMP,* ветвление происходит, если первое сравниваемое число имеет значение *(без учета знака)* меньшее, чем значение второго сравниваемого числа.

*BLOS* ВЕТВЛЕНИЕ, ЕСЛИ НИЖЕ ИЛИ ТО ЖЕ САМОЕ 1014*XXX*

Двоичный код: 1 000 001 1*xx xxx xxx*
Операция: СК ← *с* (СК) + 2·(смещение), если $C \vee Z = 1$.
Формат
ассемблера: *BLOS loc*
Коды условий: не изменяются.
Описание: младший байт инструкции — это *пословное смещение* со знаком от текущего СК к целевому адресу *loc*; если $C \vee Z = 1$, то удвоенное смещение прибавляется к с (СК); если $C \vee Z = 0$, то инструкция игнорируется; когда инструкция *BLOS* следует за инструкцией *CMP,* то ветвление будет иметь место, если первое сравниваемое число

имеет значение *(без учета знака)* меньшее чем или равное значению второго сравниваемого числа.

*BLT* ВЕТВЛЕНИЕ, ЕСЛИ МЕНЬШЕ ЧЕМ НОЛЬ 0024*XXX*

Двоичный код: 0 000 010 1*xx xxx xxx*
Операция: СК ← (СК) + 2·(смещение), если $N \veebar V = 1$.
Формат
ассемблера: *BLT loc*
Коды условий: не изменяются.
Описание: младший байт инструкции — это *пословное смещение* со знаком от текущего СК к целевому адресу *loc*; если $N \veebar V = 1$, то удвоенное смещение прибавляется к *с* (СК); если $N \veebar V = 0$, то инструкция игнорируется; когда инструкция *BLT* следует за инструкцией *CMP,* то она вызывает ветвление; если первое сравниваемое число отрицательное, а второе положительное, даже если при этом происходит переполнение; если первое сравниваемое число положительное, а второе отрицательное, то эта инструкция не вызывает ветвления.

*BMI* ВЕТВЛЕНИЕ, ЕСЛИ МИНУС 1004*XXX*

Двоичный код: 1 000 000 1xx *xxx xxx*
Операция: СК ← *с* (СК) + 2·(смещение), если $N = 1$.
Формат
ассемблера: *BMI loc*
Коды условий: не изменяются.
Описание: младший байт инструкции — это *пословное смещение* со знаком от текущего СК к целевому адресу *loc*; если $N = 1$, то удвоенное смещение прибавляется к *с* (СК); если $N = 0$, то инструкция игнорируется.

*BNE* ВЕТВЛЕНИЕ, ЕСЛИ НЕ РАВНО НУЛЮ 0010*XXX*

Двоичный код: 0 000 001 0*x xxx xxx*
Операция: СК ← *с* (СК) + 2·(смещение), если $Z = 0$.
Формат

ассемблера: *BNE loc*
Коды условий: не изменяются.
Описание: младший байт инструкции — это *пословное смещение* со знаком от текущего СК к целевому адресу *loc*, если Z = 0, то удвоенное смещение прибавляется к *с* (СК); если Z = 1, то инструкция игнорируется.

*BPL* ВЕТВЛЕНИЕ, ЕСЛИ ПЛЮС 1000*XXX*

Двоичный код: 1 000 000 0*xx xxx xxx*
Операция: СК ← *с* (СК) + 2·(смещение), если *N* = 0.
Формат
ассемблера: *BPL loc*
Коды условий: не изменяются.
Описание: младший байт инструкции — это *пословное смещение со знаком* от текущего СК к целевому адресу *loc*; если *N* = 0, то удвоенное смещение прибавляется к *с* (СК); если *N* = 1, то инструкция игнорируется.

*BPT* ПЕРЕЙТИ К ЛОВУШКЕ ТОЧКИ ПРЕРЫВАНИЯ 000003

Двоичный код: 0 000 000 000 000 011
Операция: ↓ (УС) ← *с* (ССП)
↓ (УС) ← *с* (СК)
СК ← *с* (14)
ССП ← *с* (16)
Формат ассемблера: *BPT*
Коды условий: загружаются из второго слова вектора ловушки.
Описание: выполняет переход к ловушке по фиксированным ячейкам памяти 000014—000016.

*BR* ВЕТВЛЕНИЕ (БЕЗУСЛОВНОЕ) 0004*XXX*

Двоичный код: 0 000 000 1*xx xxx xxx*
Операция: СК ← *с* (СК) + 2·(смещение).
Формат
ассемблера: *BR loc*
Коды условий: не изменяются.

Описание: младший байт инструкции — это *пословное смещение со знаком* от текущего СК к целевому адресу *loc*; удвоенное смещение прибавляется к *с* (СК) (без каких-либо условий).

*BVC* ВЕТВЛЕНИЕ, ЕСЛИ ИНДИКАТОР ПЕРЕПОЛНЕНИЯ СБРОШЕН 1020*XXX*

Двоичный код: 1 000 010 0*xx xxx xxx*
Операция: СК ← *с* (СК) + 2·(смещение), если *V* = 0.
Формат
ассемблера: *BVC loc*
Коды условий: не изменяются.
Описание: младший байт инструкции — это *пословное смещение со знаком* от текущего СК к целевому адресу *loc*; если *V* = 0, то удвоенное смещение прибавляется к *с* (СК); если *V* = 1, то инструкция игнорируется.

*BVS* ВЕТВЛЕНИЕ, ЕСЛИ ИНДИКАТОР ПЕРЕПОЛНЕНИЯ УСТАНОВЛЕН 1024*XXX*

Двоичный код: 1 000 010 1*xx xxx xxx*
Операция: СК ← *с* (СК) + 2·(смещение), если *V* = 1.
Формат
ассемблера: *BVS loc*
Коды условий: не изменяются.
Описание: младший байт инструкции — это *пословное смещение со знаком* от текущего СК к целевому адресу *loc*; если *V* = 1, то удвоенное смещение прибавляется к *с* (СК); если *V* = 0, то инструкция игнорируется.

*CCC* СБРОСИТЬ ВСЕ КОДЫ УСЛОВИЙ 000257

Двоичный код: 0 000 000 010 101 111
Операция: *C* ← 0, *V* ← 0, *Z* ← 0, *N* ← 0.
Формат
ассемблера: *CCC*
Коды условий: все сбрасываются.

Описание: сбрасывает все коды условий.
Комментарий: смотри также инструкцию *CLc.*

*CLc* СБРОСИТЬ ЗАДАННЫЙ КОД УСЛОВИЯ 00024*CC*

Двоичный код: 0 000 000 010 10*c ccc*
Операция: заданный код (коды) условий ← 0.
Формат ассемблера: *CLc*
Коды условий: заданный код (коды) сбрасывается; все остальные коды условий не изменяются.
Описание: биты 3, 2, 1 и 0 инструкции соответствуют кодам условий *N, Z, V* и *C* соответственно; если какие- либо из этих битов в инструкции установлены, то соответствующие коды условий будут сброшены.

| *Мнемоника* | *Код* | *Действие* |
|---|---|---|
| *NOP* | 000240 | Никакие коды условий не |
| *CLC* | 000241 | $C \leftarrow 0$ |
| *CLV* | 000242 | $V \leftarrow 0$ |
| Нет | 000243 | $C \leftarrow 0$, $V \leftarrow 0$ |
| *CLZ* | 000244 | $Z \leftarrow 0$ |
| Нет | 000245 | C ← 0, $Z \leftarrow 0$ |
| » | 000246 | $V \leftarrow 0$, $Z \leftarrow 0$ |
| » | 000247 | $C \leftarrow 0$, $V \leftarrow 0$, $Z \leftarrow 0$ |
| *CLN* | 000250 | $N \leftarrow 0$ |
| Нет | 000251 | $C \leftarrow 0$, $N \leftarrow 0$ |
| » | 000252 | V ← 0, $N \leftarrow 0$ |
| Нет | 000253 | $C \leftarrow 0$, $V \leftarrow 0$, $N \leftarrow 0$ |
| » | 000254 | $Z \leftarrow 0$, $N \leftarrow 0$ |
| » | 000255 | $C \leftarrow 0$, $Z \leftarrow 0$, $N \leftarrow 0$ |
| » | 000256 | $V \leftarrow 0$, $Z \leftarrow 0$, $N \leftarrow 0$ |
| *CCC* | 000257 | $C \leftarrow 0$, $V \leftarrow 0$, $Z \leftarrow 0$, $N \leftarrow 0$ |

*CLR* СБРОСИТЬ ПРИЕМНИК 0050*DD*

Двоичный код: 0 000 101 000 *ddd ddd*
Операция: *dst* ← 0
Формат ассемблера: *CLR dst*
Коды условий: *N* — сбрасывается;

*Z* — устанавливается;
*V* — сбрасывается;
*C* — сбрасывается.

Описание: слову-приемнику присваивается значение 0.

*CLRB* СБРОСИТЬ ПРИЕМНИК—БАЙТ 1050*DD*

Двоичный код: 1 000 101 000 *ddd ddd*
Операция: *dst* ← 0
Формат ассемблера: *CLRB dst*
Коды условий: *N* — сбрасывается;
*Z* — установлен;
*V* — сбрасывается;
*C* — сбрасывается.

Описание: байту-приемнику присваивается значение 0.

*CMP* СРАВНИТЬ ИСТОЧНИК С ПРИЕМНИКОМ 02*SSDD*

Двоичный код: 0 010 *sss sss ddd ddd*
Операция: *c* (*src*) + [~*c* (*dst*) + 1]
Формат ассемблера: *CMP src, dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — устанавливается, если было переполнение (операнды имели противоположные знаки, а знак результата такой же, как знак приемника), иначе сбрасывается;
*C* — устанавливается, если не было переноса из самого старшего бита результата, иначе сбрасывается.

Описание: вычитает содержимое приемника из содержимого источника и устанавливает коды условий в соответствии с результатом; ни содержимое источника, ни содержимое приемника не изменяются (обычно эта инструкция сопровождается какой-либо инструкцией условного ветвления).

*CMPB* СРАВНИТЬ ИСТОЧНИК—БАЙТ
С ПРИЕМНИКОМ—БАЙТОМ 12*SSDD*

Двоичный код: 1 010 *sss sss ddd ddd*
Операция: *c* (*src*) + [~*c* (*dst*) + 1]
Формат
ассемблера: *CMPB src, dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — устанавливается, если было переполнение (операнды имели противоположные знаки, а знак результата такой же, как знак приемника), иначе сбрасывается ;
*C* — устанавливается, если не было переноса из самого старшего бита результата, иначе сбрасывается.
Описание: вычитает содержимое байта приемника из содержимого байта источника и устанавливает коды условий в соответствии с результатом; ни содержимое байта-источника, ни содержимое байта-приемника не изменяются (обычно эта инструкция сопровождается какой-либо инструкцией условного ветвления).

*COM* ВЗЯТЬ ДОПОЛНЕНИЕ ПРИЕМНИКА 0051*DD*

Двоичный код: 0 000 101 001 *ddd ddd*
Операция: *dst* ← ~*c* (*dst*)
Формат
ассемблера: *COM dst*
Код условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — сбрасывается;
*C* — устанавливается.
Описание: приемник заменяется его дополнением до единицы (каждый нулевой бит устанавливается в единицу, а каждый единичный бит сбрасывается в нуль).

*COMB* ВЗЯТЬ ДОПОЛНЕНИЕ ПРИЕМНИКА—БАЙТА 1051*DD*

Двоичный код: 1 000 101 001 *ddd ddd*
Операция: *dst* ← ~*c* (*dst*)
Формат
ассемблера: *COMB dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — сбрасывается;
*C* — устанавливается.
Описание: байт приемника заменяется его дополнением до единицы (каждый нулевой бит устанавливается в единицу, а каждый единичный бит сбрасывается в нуль).

*DEC* УМЕНЬШИТЬ ПРИЕМНИК 0053*DD*

Двоичный код: 0 000 101 011 *ddd ddd*
Операция: *dst* ← *c* (*dst*) — 1
Формат
ассемблера: *DEC dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — устанавливается, если *c (dst)* равно 100000, иначе сбрасывается;
*C* — не изменяется.
Описание: вычитает единицу из содержимого слова приемника.

*DECB* УМЕНЬШИТЬ ПРИЕМНИК—БАЙТ 1053*DD*

Двоичный код: 1 000 101 011 *ddd ddd*
Операция: *dst* ← *c* (*dst*) — 1
Формат
ассемблера: *DECB dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
Z — устанавливается, если результат равен нулю, иначе сбрасывается;

*V* — устанавливается, если *c (dst)* равно 200, иначе сбрасывается;
*C* — не изменяется.

Описание: вычитает единицу из содержимого байта приемника.

*EMT* ПЕРЕЙТИ К ЛОВУШКЕ ЭМУЛЯТОРА 104000—104377

Двоичный код: 1 000 100 0*nn nnn nnn*
Операция: ↓ (УС) ← *c* (ССП)
↓ (УС) ← *c* (СК)
СК ← *c* (30)
ССП ← *c* (32)
Формат ассемблера: *EMT NNN*
Коды условий: загружается из второго слова вектора ловушки.
Описание: выполняет переход к ловушке через вектор в фиксированных ячейках памяти 000030—000032; младший байт инструкции декодером ЦП не используется.
Комментарий: во многих операционных системах эта инструкция используется для обращения к утилитным подпрограммам, поэтому пользователю рекомендуется использовать вместо нее инструкцию *TRAP*.

*HALT* ОСТАНОВИТЬ ПРОЦЕССОР 000000

Двоичный код: 0 000 000 000 000 000
Операция: Останов процессора
Формат ассемблера: *HALT*
Коды условий: не изменяются.
Описание: прекращает работу процессора.

*INC* УВЕЛИЧИТЬ ПРИЕМНИК 0052*DD*

Двоичный код: 0 000 101 010 *ddd ddd*
Операция: *dst* ← *c* (*dst*) + 1
Формат ассемблера: *INC dst*
Коды условий: *N* — устанавливается, если результат меньше нуля,

иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*v* — устанавливается, если *c (dst)* равно 077777, иначе сбрасывается;
*C* — не изменяется.

Описание: прибавляет единицу к содержимому слова приемника.

*INCB* УВЕЛИЧИТЬ ПРИЕМНИК—БАЙТ 1052*DD*

Двоичный код: 1 000 101 010 *ddd ddd*
Операция: *dst* ← *c* (*dst*) + 1
Формат ассемблера: *INCB dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
Z — устанавливается, если результат равен нулю, иначе сбрасывается;
*v* — устанавливается, если *c (dst)* равно 177, иначе сбрасывается;
*C* — не изменяется.

Описание: прибавляет единицу к содержимому байта приемника.

*IOT* ПЕРЕЙТИ К ЛОВУШКЕ ВВОДА—ВЫВОДА 000004

Двоичный код: 0 000 000 000 000 100
Операция: ↓ (УС) ← *c* (ССП)
↓ (УС) ← *c* (СК)
СК ← *c* (20)
ССП ← *c* (22)
Формат ассемблера: *IOT*
Коды условий: загружаются из второго слова вектора ловушки.
Описание: выполняет переход к ловушке через вектор в фикси рованных ячейках памяти 000020—000022.

*JMP* ПЕРЕЙТИ К ПРИЕМНИКУ 0001*DD*

Двоичный код: 0 000 000 001 *ddd ddd*
Операция: СК ← адрес *dst*
Формат
ассемблера: *JMP dst*
Коды условий: не изменяются.
Описание: программному счетчику СК присваивается значение адреса приемника.

*JSR* ПЕРЕЙТИ К ПОДПРОГРАММЕ 004*RDD*

Двоичный код: 0 000 100 *rrr ddd ddd*
Операция: *temp* ← адрес *dst*
↓ УС ← с (*reg*)
*reg* ← *c* (СК)
СК ← *c (temp),* где *temp* — это внутренний регистр процессора.
Формат
ассемблера: *JSR reg, dst*
Коды условий: не изменяются.
Описание: эта инструкция вычисляет адрес приемника, сохраняет заданный регистр в аппаратном стеке *(регистр для перехода к подпрограмме)* и сохраняет в этом регистре текущее значение СК (такое сохранение СК обеспечивает связку из подпрограммы *назад* в вызывающую программу); наконец, СК присваивается значение адреса приемника, что порождает переход к подпрограмме; возврат в вызывающую подпрограмму обеспечивается парной инструкцией *RTS.*

*MARK* ПОМЕТИТЬ СТЕК ДЛЯ ОЧИСТКИ 0064*NN*

Двоичный код: 0 000 110 100 *nnn nnn*
Операция: УС ← *c* (СК) + *2NN*
СК ← *c* (*R*5)
*R5* ← (УС) ↑
Формат
ассемблера: *MARK NN*
Коды условий: не изменяются.
Описание: когда аргументы передаются подпрограмме путем

помещения ихваппаратныйстек,эта инструкция может использоваться для очистки стека при возврате из подпрограммы.

*MFPS* ПРОЧИТАТЬ ССП 1067*DD*

Двоичный код: 1 000 110 111 *ddd ddd*
Операция: *dst* ← *c* (ССП)
Формат
ассемблера: *MFPS dst*
Коды условий: *N* — устанавливается, если *c* (ССП) < 0, иначе сбрасывается;
*Z* — устанавливается, если *c* (ССП) = 0, иначе сбрасывается;
*V* — обнуляется;
*C* — не изменяется.
Описание: помещает в приемник содержимое слова состояния процессора.

*MOV* ПЕРЕСЛАТЬ ИСТОЧНИК В ПРИЕМНИК 01*SSDD*

Двоичный код: 0 001 *sss sss ddd ddd*
Операция: *dst* ← *c* (*src*)
Формат
ассемблера: *MOV src, dst*
Коды условий: *N* — устанавливается, если *c (src)* < 0, иначе сбрасывается;
*Z* — устанавливается, если *c (src)* = 0, иначе сбрасывается;
*V* — сбрасывается;
*C* — не изменяется.
Описание: пересылает копию содержимого слова источника в приемник; первоначальное содержимое приемника теряется; содержимое слова источника не изменяется.

*MOVB* ПЕРЕСЛАТЬ ИСТОЧНИК—БАЙТ В ПРИЕМНИК 11*SSDD*

Двоичный код: 1 001 *sss sss ddd ddd*

Операция: *dst* ← *c (src)*
Формат
ассемблера: *MOVB src, dst*
Коды условий: *N* — устанавливается, если *c (src)* < 0, иначе сбрасывается;
*Z* — устанавливается, если *c (src)* = 0, иначе сбрасывается;
*V* — сбрасывается;
*C* — не изменяется.
Описание: пересылает копию содержимого байта источника в байт приемника; первоначальное содержимое приемника теряется; содержимое байта источника не изменяется.

Замечание. Если приемником является *регистр,* то пересылка происходит в младший байт регистра приемника. *Старший байт* регистра приемника заполняется значением самого старшего бита байта источника (что создает эффект сохранения арифметического знака содержимого байта источника).

*MTPS* ЗАНЕСТИ В ССП 1064*SS*

Двоичный код: 1 000 110 100 *sss sss*
Операция: ССП ← *c (src)*
Формат
ассемблера: *MTPS src*
Коды условий: биты 0—3 источника помещаются в соответствующие биты слова состояния процессора, формируя коды условий.
Описание: пересылает копию содержимого слова источника в слово состояния процессора.

*NEG* ИЗМЕНИТЬ ЗНАК ПРИЕМНИКА 0054*DD*

Двоичный код: 0 000 101 100 *ddd ddd*
Операция: *dst* ← *~c (dst)* + 1
Формат
ассемблера: *NEG dst*
Код условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю,

иначе сбрасывается;
*V* — устанавливается, если результат равен 100000, иначе сбрасывается;
*C* — сбрасывается, если результат равен нулю, иначе устанавливается.

Описание: слово приемника заменяется его дополнением до двух (отрицательным значением).

*NEGB* ИЗМЕНИТЬ ЗНАК ПРИЕМНИКА—БАЙТА 1054*DD*

Двоичный код: 1 000 101 100 *ddd ddd*
Операция: *dst* ← *~c (dst)* + 1
Формат ассемблера: *NEGB dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — устанавливается, если результат равен 200, иначе сбрасывается;
*C* — сбрасывается, если результат равен нулю, иначе устанавливается.

Описание: байт приемника заменяется его дополнением до двух (отрицательным значением).

*NOP* НЕТ ОПЕРАЦИИ 000240

Двоичный код: 0 000 000 010 100 000
Операция: отсутствует.
Формат ассемблера: *NOP*
Коды условий: не изменяются.
Описание: эта инструкция такая же самая, как «*не сбрасывать никаких кодов условий*» (см. *CLc*), хотя она никакого действия не производит, занимает одно слово памяти и требует полного цикла при «выполнении» (эта инструкция дается для удобства программиста; например, если разрабатываемая программа содержит «точки прерывания», то инструкции *BPT* могут временно заменяться путем перекрытия их инструкциями *NOP*).

*RESET* СБРОСИТЬ ШИНУ В ИСХОДНОЕ СОСТОЯНИЕ 000005

Двоичный код: 0 000 000 000 000 101
Операция: посылает по шине сигнал *INIT*
Формат ассемблера: *RESET*
Описание: сбрасывает все устройства на шине в состояние, которое они имеют после подачи питания.

*ROL* ПРОВЕРНУТЬ ВЛЕВО ПРИЕМНИК 0061*DD*

Двоичный код: 0 000 110 001 *ddd ddd*
Операция: *C, dst ← c (C, dst),* провернутое на один бит влево.
Формат ассемблера: *ROL dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — загружается результатом операции ИСКЛЮЧАЮЩЕЕ ИЛИ над *N* и *C* (какими они оказываются *после* поворота);
*C* — загружается значением бита, сдвинутым из самого старшего бита приемника.
Описание: бит *C* в ССП и слово приемника, рассматриваемые как 17-битовое «слово», проворачиваются влево на один бит; в освободившийся самый младший бит приемника (бит 0) загружается содержимое бита *C;* значение самого старшего бита приемника (бита 15) загружается в бит *C.*

*ROLB* ПРОВЕРНУТЬ ВЛЕВО ПРИЕМНИК—БАЙТ 1061*DD*

Двоичный код: 1 000 110 001 *ddd ddd*
Операция: *C, dst ← c (C, dst),* провернутое на один бит влево.
Формат ассемблера: *ROLB dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю,

иначе сбрасывается;
*V* — загружается результатом операции ИСКЛЮЧАЮЩЕЕ ИЛИ над *N* и *C* (какими они оказываются *после* поворота);
*C* — загружается значением бита, сдвинутым из самого старшего бита приемника.

Описание: бит *C* в ССП и байт приемника, рассматриваемые как 9-битовый «байт», проворачиваются влево на один бит; в освободившийся самый младший бит приемника (бит 0 или 8) загружается содержимое бита *C*; значение самого старшего бита приемника (бита 7 или 15) загружается в бит *C*.

*ROR* ПРОВЕРНУТЬ ВПРАВО ПРИЕМНИК 0060*DD*

Двоичный код: 0 000 110 000 *ddd ddd*
Операция: *C, dst* ← *c (C, dst),* провернутое вправо на один бит.
Формат ассемблера: *ROR dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — загружается результатом операции ИСКЛЮЧАЮЩЕЕ ИЛИ над *N* или *C* (какими они оказываются *после* поворота);
*C* — загружается значением бита, сдвинутого из самого младшего бита приемника.

Описание: бит *C* в ССП и слово приемника, рассматриваемые как 17-битовое «слово», проворачиваются вправо на один бит; в освободившийся самый старший бит (бит 15) загружается содержимое бита *C*; значение самого младшего бита приемника (бита 0) загружается в бит *C*.

*RORB* ПРОВЕРНУТЬ ВПРАВО ПРИЕМНИК—БАЙТ 1060*DD*

Двоичный код: 1 000 110 000 *ddd ddd*
Операция: *C, dst* ← *c (C, dst),* провернутое вправо на один байт.
Формат ассемблера: *RORB dst*

Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — загружается результатом операции ИСКЛЮЧАЮЩЕЕ ИЛИ над *N* и C (какими они оказываются после поворота);
*C* — загружается значением бита, сдвинутого из самого младшего бита приемника.

Описание: бит *C* в ССП и байт приемника, рассматриваемые как 9-битовый «байт», проворачиваются на один бит вправо; в освободившийся самый старший бит (бит 7 или 15) загружается содержимое бита *C*; значение самого младшего бита байта приемника (бита 0 или 8) загружается в бит *C*.

*RTI* ВЕРНУТЬСЯ ИЗ ПРЕРЫВАНИЯ 000002

Двоичный код: 0 000 000 000 000 010

Операция: СК ← (УС) ↑
ССП ← (УС) ↑

Формат ассемблера: *RTI*

Коды условий: загружаются из аппаратного стека.

Описание: содержимое СК и ССП восстанавливается из аппаратного стека (эта инструкция используется для возврата из подпрограммы обработки прерывания или обработки ловушки).

*RTS* ВЕРНУТЬСЯ ИЗ ПОДПРОГРАММЫ 00020

Двоичный код: 0 000 000 010 000 *rrr*

Операция: СК ← *c (reg)*

Формат ассемблера: *RTS reg*

Коды условий: не изменяются.

Описание: содержимое заданного регистра загружается в СК, а в регистр выталкивается верхний элемент аппаратного стека (эта инструкция используется для возврата из подпрограмм).

*RTT* ВЕРНУТЬСЯ ИЗ ЛОВУШКИ 000006

Двоичный код: 0 000 000 000 000 110
Операция: СК ← (УС) ↑
ССП ← (УС) ↑
Формат
ассемблера: *RTT*
Коды условий: загружаются из аппаратного стека.
Описание: содержимое СК и ССП восстанавливается из аппаратного стека (эта инструкция используется для возврата из подпрограмм обработки прерываний или обработки ловушек); инструкция *RTT* идентичная инструкции *RTI* за исключением поведения, зависящего от состояния бита 4 в ССП.

*SBC* ВЫЧЕСТЬ ПЕРЕНОС ИЗ ПРИЕМНИКА 0056*DD*

Двоичный код: 0 000 101 110 *ddd ddd*
Операция: *dst ← c (dst) — c (C)*
Формат
ассемблера: *SBC dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — устанавливается, если *c (dst)* = 100000, иначе сбрасывается;
*C* — сбрасывается, если *c (dst)* = 000000 и *c (C)* = 1, иначе устанавливается.
Описание: вычитает содержимое бита *C* из приемника.

*SBCB* ВЫЧЕСТЬ ПЕРЕНОС ИЗ ПРИЕМНИКА—БАЙТА 1056*DD*

Двоичный код: 1 000 101 110 *ddd ddd*
Операция: *dst ← c (dst) — c (C)*
Формат
ассемблера: *SBCB dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;

*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — устанавливается, если с *(dst)* = 200, иначе сбрасывается;
*C* — сбрасывается, если *с (dst)* = 000 и *с (C)* = 1, иначе устанавливается.

Описание: вычитает содержимое бита *C* из байта-приемника.

## *SCC* УСТАНОВИТЬ ВСЕ КОДЫ УСЛОВИЙ 000277

Двоичный код: 0 000 000 010 111 111
Операция: *C* ← 1, *V* ← 1, *Z* ← 1, *N* ← 1.
Формат ассемблера: *SCC*
Коды условий: все устанавливаются.
Описание: устанавливает все коды условий.
Комментарий: смотри также инструкцию *SEc*

## *SEc* УСТАНОВИТЬ ЗАДАННЫЙ КОД УСЛОВИЯ 00026*CC*

Двоичный код: 0 000 000 010 11*c ccc*
Операция: заданный код (коды) условия ← 1.
Формат ассемблера: *SEc*
Коды условий: заданный код (коды) условий устанавливается; все остальные коды условий не изменяются.
Описание: биты 3, 2, 1 и 0 инструкции соответствуют кодам условий *N, Z, V* и *C* соответственно; если какие-то из этих битов в инструкции установлены, то будут устанавливаться соответствующие коды условий.

| *Мнемоника* | *Код* | *Действие* |
|---|---|---|
| *NOP* | 000260 | Никакие коды условий не устанавливаются |
| *SEC* | 000261 | *C* ← 1 |
| *SEV* | 000262 | *V* ← 1 |
| Нет | 000263 | *C* ← 1, *V* ← 1 |
| *SEZ* | 000264 | *Z* ← 1 |
| Нет | 000265 | *C* ← 1, *Z* ← 1 |
| » | 000266 | *V* ← 1, *Z* ← 1 |
| » | 000267 | *C* ← 1, *V* ← 1, *Z* ← 1 |
| *SEN* | 000270 | *N* ← 1 |
| Нет | 000271 | *C* ← 1, *N* ← 1 |

| | | |
|---|---|---|
| » | 000272 | $V \leftarrow 1, N \leftarrow 1$ |
| » | 000273 | $C \leftarrow 1, V \leftarrow 1, N \leftarrow 1$ |
| » | 000274 | $Z \leftarrow 1, N \leftarrow 1$ |
| » | 000275 | $C \leftarrow 1, Z \leftarrow 1, N \leftarrow 1$ |
| » | 000276 | $V \leftarrow 1, Z \leftarrow 1, N \leftarrow 1$ |
| *SCC* | 000277 | $C \leftarrow 1, V \leftarrow 1, Z \leftarrow 1, N \leftarrow 1$ |

## *SOB* ВЫЧЕСТЬ ЕДИНИЦУ И ВЕРНУТЬСЯ НАЗАД 077*RXX*

Двоичный код: 0 111 111 *rrr xxx xxx*
Операция: *reg* ← *c (reg)* — 1
СК ← *c* (СК) — 2·(смещение), если *c (reg)* ≠ *0*
Формат
ассемблера: *SOB reg, loc*
Коды условий: не изменяются.
Описание: шесть младших битов инструкции — это *пословное смещение без знака* от текущего СК к целевому адресу *loc*; содержимое заданного регистра уменьшается на единицу; если результат ненулевой, то удвоенное смещение *вычитается* из *c* (СК); если после уменьшения содержимое регистра равно нулю, то ветвление не происходит, а извлекается следующая инструкция.
Эта инструкция используется для управления циклами, когда *reg* действует как счетчик цикла. Передача управления может происходить только в обратном направлении (к *меньшим* адресам памяти).

## *SUB* ВЫЧЕСТЬ ИСТОЧНИК ИЗ ПРИЕМНИКА 16*SSDD*

Двоичный код: 1 110 *sss sss ddd ddd*
Операция: *dst* ← *c (dst)* + [~*c* (*src*) + 1]
Формат
ассемблера: *SUB src, dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — устанавливается, если было переполнение (операнды имели противоположные знаки, а знак источника был таким же, как знак результата), иначе

сбрасывается;
*C* — сбрасывается, если был перенос из бита 15, иначе устанавливается.

Описание: вычитает содержимое источника из содержимого приемника и помещает результат в приемник; первоначальное содержимое приемника теряется; содержимое источника не изменяется.

*SWAB* ОБМЕНЯТЬ БАЙТЫ ПРИЕМНИКА 0003*DD*

Двоичный код: 0 000 000 011 *ddd ddd*

Операция: старший байт *dst,* младший байт *dst* ← младший байт *dst,* старший байт *dst*

Формат ассемблера: *SWAB dst*

Коды условий: *N* — устанавливается, если *с (dst)* < 0, иначе сбрасывается;
*Z* — устанавливается, если старший байт *dst* = 000, иначе сбрасывается;
*V* — сбрасывается;
*C* — сбрасывается.

Описание: меняется местами старший и младший байты слова приемника.

*SXT* РАСШИРИТЬ ЗНАК ПРИЕМНИКА 0067*DD*

Двоичный код: 0 000 110 111 *ddd ddd*

Операция: *dst* ← 0, если *с (N)* = 0
*dst* ← —1, если *с (N)* = 1.

Формат ассемблера: *SXT dst*

Коды условий: *N* — не изменяется;
*Z* — устанавливается, если *с (dst)* = 0, иначе сбрасывается ;
*V* — сбрасывается;
*C* — не изменяется.

Описание: заполняет слово приемника текущим содержимым бита *N.*

*TRAP* ПЕРЕЙТИ К ЛОВУШКЕ 104400—104777

Двоичный код: 1 000 100 1*nn nnn nnn*
Операция: ↓ (УС) ← *с* (ССП)
↓ (УС) ← *с* (СК)
СК ← *с* (34)
ССП ← *с* (36)
Формат
ассемблера: *TRAP NNN*
Коды условий: загружаются из второго слова вектора ловушки,
Описание: выполняет переход к ловушке через вектор в фикси рованных ячейках памяти 000034—000036; младший байт инструкции декодером ЦП не используется.

*TST* ПРОВЕРИТЬ ПРИЕМНИК 0057*DD*

Двоичный код: 0 000 101 111 *ddd ddd*
Операция: *dst* ← *с (dst)*
Формат
ассемблера: *TST dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
Z — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — сбрасывается; *C* — сбрасывается.
Описание: устанавливает коды условий *N* и *Z* в соответствии с содержимым слова приемника.

*TSTB* ПРОВЕРИТЬ ПРИЕМНИК—БАЙТ 1057*DD*

Двоичный код: 1 000 101 111 *ddd ddd*
Операция: *dst* ← *с (dst)*
Формат
ассемблера: *TSTB dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — сбрасывается;
*C* — сбрасывается.
Описание: устанавливает коды условий *N* и *Z* в соответствии

с содержимым байта приемника.

*WAIT* ОЖИДАТЬ ПРЕРЫВАНИЯ 000001

Двоичный код: 0 000 000 000 000 001
Операция: приостановить работу процессора и ожидать прерывания.
Формат ассемблера: *WAIT*
Коды условий: не изменяются.
Описание: процессор отказывается от управления шинами (т. е. прекращает извлечения инструкций) до тех пор, пока не произойдет внешнее прерывание; если запрос на прерывание будет удовлетворен (что зависит от приоритетов), то последующая инструкция *RTI* в подпрограмме обработки прерываний возобновит выполнение с инструкции, следующей за инструкцией *WAIT*.

*XOR* ВЫПОЛНИТЬ ИСКЛЮЧАЮЩЕЕ ИЛИ НАД ПРИЕМНИКОМ И РЕГИСТРОМ 074*RDD*

Двоичный код: 0 111 100 *rrr ddd ddd*
Операция: *dst* ← *c (reg)* $\underline{\vee}$ *c (dst)*
Формат ассемблера: *XOR reg, dst*
Коды условий: *N* — устанавливается, если результат меньше нуля, иначе сбрасывается;
*Z* — устанавливается, если результат равен нулю, иначе сбрасывается;
*V* — сбрасывается;
*C* — не изменяется.
Описание: выполняется операция ИСКЛЮЧАЮЩЕЕ ИЛИ над содержимым регистра и содержимым приемника, и результат помещается в слово приемника; содержимое регистра не изменяется.

Таблица 5

**Набор инструкций в числовом порядке операционного кода**

| Код | Мнемоника | Код | Мнемоника | Код | Мнемоника |
|---|---|---|---|---|---|
| 000000 | *HALT* | 0050*DD* | *CLR* | 1030*XXX* | *BCC* или *BHIS* |
| 000001 | *WAIT* | 0051*DD* | *COM* | | |
| 000002 | *RTI* | 0052*DD* | *INC* | 1034*XXX* | *BCS* или *BLO* |
| 000003 | *BPT* | 0053*DD* | *DEC* | | |
| 000004 | *IOT* | 0054*DD* | *NEG* | 104000 | |
| 000005 | *RESET* | 0055*DD* | *ADC* | до 104377 | *EMT* |
| 000006 | *RTT* | 0056*DD* | *SBC* | | |
| 0001*DD* | *JMP* | 0057*DD* | *TST* | 104400 | |
| 00020*R* | *RTS* | 0060*DD* | *ROR* | до 104477 | *TRAP* |
| 000240 | *NOP* | 0061*DD* | *ROL* | | |
| 000241 | *CLC* | 0062*DD* | *ASR* | 1050*DD* | *CLRB* |
| 000242 | *CLV* | 0063*DD* | *ASL* | 1051*DD* | *COMB* |
| 000244 | *CLZ* | 0064*DD* | *MARK* | 1052*DD* | *INCB* |
| 000250 | *CLN* | 0067*DD* | *SXT* | 1053*DD* | *DECB* |
| 000257 | *CCC* | 01*SSDD* | *MOV* | 1054*DD* | *NEGB* |
| 000261 | *SEC* | 02*SSDD* | *CMP* | 1055*DD* | *ADCB* |
| 000262 | *SEV* | 03*SSDD* | *BIT* | 1056*DD* | *SBCB* |
| 000264 | *SEZ* | 04*SSDD* | *BIC* | 1057*DD* | *TSTB* |
| 000270 | *SEN* | 05*SSDD* | *BIS* | 1060*DD* | *RORB* |
| 000277 | *SCC* | 06*SSDD* | *ADD* | 1061*DD* | *ROLB* |
| 0003*DD* | *SWAB* | 074*RSS* | *XOR* | 1062*DD* | *ASRB* |
| 0004*XXX* | *BR* | 077*RSS* | *SOB* | 1063*DD* | *ASLB* |
| 0010*XXX* | *BNE* | 1000*XXX* | *BPL* | 11*SSDD* | *MOVB* |
| 0014*XXX* | *BEQ* | 1004*XXX* | *BMI* | 12*SSDD* | *CMPB* |
| 0020*XXX* | *BGE* | 1010*XXX* | *BHI* | 13*SSDD* | *BITB* |
| 0024*XXX* | *BLT* | 1014*XXX* | *BLOS* | 14*SSDD* | *BICB* |
| 0030*XXX* | *BGT* | 1020*XXX* | *BVC* | 15*SSDD* | *BISB* |
| 0034*XXX* | *BLE* | 1024*XXX* | *BVS* | 16*SSDD* | *SUB* |
| 004*RDD* | *JSR* | | | | |

## СПИСОК ЛИТЕРАТУРЫ


1. **Ашкенази В. А.** Литерные величины на БК-0010//Информатика и образование. 1987, № 4. С. 69—70.
2. **Безродный М. С.** Классификация и характеристики дисплеев для микроЭВМ//Микропроцессорные средства и системы. 1986. № 4. С. 28—34.
3. **Бытовая** персональная микроЭВМ «Электроника БК-0010»/С. М. Косенков, А. Н. Полосин, З. А. Счепицкий и др.// Микропроцессорные средства и системы. 1985. № 1. С. 22—25.
4. **Варсанофьев Д. В., Кушниренко А. Г., Лебедев Г. В.** Е-практикум — программное обеспечение школьного курса информатики и вычислительной техники//Микропроцессорные средства и системы. 1985. № 3. С. 27—32.
5. **Дамке М.** Операционные системы микроЭВМ. М.: Финансы и статистика, 1985. 173 с.
6. **Завилов В. Н., Константинов М. Ю., Померанец М. В.** Программирование на языке Паскаль для микроЭВМ «Электроника БК-0010» //Микропроцессорные средства и системы. 1987. № 1. С. 37—39.
7. **Квят В. Е., Потанин В. Ю.** Расширение возможностей системы схемотопологического проектирования 15УТ-4-017// Электронная промышленность. 1986. Вып. 2 (150). С. 54—55.
8. **Комплекс** программ расширенной графики для интерактивной графической системы 15УТ-4-017 «Кулон»/Л. Г. Морина, Н. А. Кочетова, Н. П. Пушков и др.//Электронная техника. Сер. 1. Электроника СВЧ. 1985. Вып. 7(379). С. 68.
9. **Комплексный** класс технических средств на базе микроЭВМ «Электроника БК-0010Ш» и ДВК-2МШ/Г. И. Фролов, С. М. Косенков, В. А. Шахнов и др.//Микропроцессорные средства и системы. 1986. № 4. С. 65—66.
10. **Кузьмин Ю. Я.** Т-язык//Информатика и образование. 1987. № 2. С. 64—70.
11. **Лингер Р., Миллс Х., Уитт Б.** Теория и практика структурного программирования. М.: Мир, 1982, 406 с.
12. **Микропроцессоры.** Книга 1. Архитектура и проектирование микро- ЭВМ. Организация вычислительных процессов/Под ред. Л. Н. Преснухина. М.: Высшая школа, 1986. 495 с.
13. **Микропроцессоры.** Книга 2. Средства сопряжения. Контролирующие и информационно-управляющие системы/Под ред. Л. Н. Преснухина. М.: Высшая школа, 1986. 417 с.
14. **МикроЭВМ** «Электроника БК-0010» в системе управления производством кварцевого стекла/П. А. Обновленский, Е. А. Рудометов, А. Л. Фокин и др.// Микропроцессорные средства и системы. 1987. № 2. С. 48—50.
15. **Дшхунян Б. Л.** Однокристальные микропроцессоры комплекта БИС серии К1801//Микропроцессорные средства и системы. 1984. № 4. С. 12—18.
16. **Персональная** ЭВМ «Ириша»: периферийные устройства, источник питания/В. Ю. Романов, В. Н. Барышников, М. А. Воронов//Микропроцессорные средства и системы. 1986. № 3. С. 53—59.
17. **Писаревский А. Н., Осетинский Л. Г., Осетинский М. Г.** ФОКАЛ —

диалоговый язык для мини-ЭВМ. Л.: Машиностроение, 1985. 195 с.
18. **Программа** ввода и редактирования графической информации со встроенным интерпретатором языка ФОКАЛ/С. В. Акулов, И. В. Кузнецов, Т. В. Останенко и др.//Электронная техника. Сер. 1. Электроника СВЧ. 1986. Вып. 6 (390). С. 78.
19. **Программирование** на языке ФОКАЛ/Г. И. Фролов, Т. А. Куправа, Л. Н. Преснухин и др. М.: МИЭТ, 1984. 79 с.
20. **Рязанский М., Яцюк О.** Палитра компьютерной графики//Информатика и образование. 1987. № 1. С. 58—62.
21. **Сборник** научных программ на Фортране/Пер. с англ. Т. 1. М.: Статистика, 1974. 322 с.
22. **Структура** системы графического отображения АЛГРАФ и язык описания геометрической информации к ней/Ю. В. Котов, Л. В. Арциховская-Кузнецова, М. В. Архипкин и др.//Тр. 4-й Всесоюзной конференции по проблемам машинной графики. Серпухов, сент. 1987. С. 26.
23. **Унифицированные** интерактивные средства проектирования изделий электронной техники/Б. Л. Толстых, И. Л. Талов, В. Н. Харин и др. М.: Радио и связь, 1985. 136 с.
24. **Учебный** класс на основе диалоговых вычислительных комплексов/Л. Н. Преснухин, Г. И. Фролов, Т. А. Куправа и др.//Микропроцессорные средства и системы. 1985. № 3. С. 39— 41.
25. **Федорчук Г. В., Черненький В. М.** Информационное и прикладное программное обеспечение. М.: Высшая школа, 1986. 159 с.
26. **Фролов Г. И., Гембицкий Р. А.** Автоматизированные системы контроля объектов. Микропроцессоры. М.: Высшая школа, 1984. 84 с.
27. **Шерр С.** Электронные дисплеи. М.: Мир, 1982. 624 с.
28. **Штильман З. М., Штильман Б. М.** Метод программирования на БЕЙСИКе и ФОКАЛе на основе алгоритмического языка//Микро- процессорные средства и системы. 1986. № 3. С.

# О Г Л А В Л Е Н И Е

ПРОИЗВОДСТВЕННОЕ ИЗДАНИЕ

Осетинский Лев Георгиевич
Осетинский Михаил Георгиевич
Писаревский Александр Николаевич

**ФОКАЛ ДЛЯ МИКРО- И МИНИ-КОМПЬЮТЕРОВ**

Редактор *Н. В. Сергеева*
Переплет художника *В. И. Коломейцева*
Художественный редактор *Н. В. Зимаков*
Технические редакторы *Т. М. Жилич, П. В. Шиканова*
Корректоры *И. Г. Иванова, З. С. Романова, Н. В. Соловьева*

ИБ № 5849

Сдано в набор 02.06.88. Подписано в печать 30.09.88. М-35775.
Формат 84×108$^{1}/_{32}$. Бумага типографская № 1. Гарнитура литературная. Печать высокая. Усл. печ. л. 15,96. Усл. кр.-отт. 15,96. Уч.-изд. л. 16,8.
Тираж 100 000 экз. Заказ 510. Цена 1 р. 20 к.

Ленинградское отделение ордена Трудового Красного Знамени издательства «Машиностроение» 191065, Ленинград, ул. Дзержинского, 10

Ленинградская типография № 6 ордена Трудового Красного Знамени Ленинградского объединения «Техническая книга» им. Евгении Соколовой Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. 193144, г. Ленинград, ул. Моисеенко, 10.

1р20к.