

В. П. ДЬЯКОНОВ

Форт-системы программирования персональных ЭВМ

Справочное пособие



МОСКВА «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ

1992

ББК 22.18
Д93
УДК 519.68

Дьяконов В. П. **Форт-системы программирования персональных ЭВМ.**— М.: Наука. Гл. ред. физ.-мат. лит., 1992—352 с.— ISBN 5-02-014460-6

Детально описываются версии, оперирующие с числами с плавающей точкой. Даются основы программирования на Форте. Приводятся свыше 400 практических примеров расширения версий языка и разработки комплексов прикладных программ, охватывающих реализацию массовых математических, научно-технических, учебных расчетов и различных системных функций. Показываются возможности ПЭВМ: реализация цветной и лого-графики на Форте, создание звуковых сигналов, изменение алфавита, построение графиков функций и т. д.

Для инженеров и студентов вузов.

Табл. 17. Ил. 55. Библиогр. 40 назв.

Д $\frac{1404000000-019}{053(02)-92}$ 138-92

ISBN 5-02-014460-6

© «Наука». Физматлит, 1992

ПРЕДИСЛОВИЕ

Язык Форт (FORTH) четвертого поколения для программирования персональных ЭВМ по популярности за рубежом вышел на третье место (вслед за Бейсиком и Паскалем) [1, 8]. Он широко применяется для разработки пакетов прикладных программ, программирования микропроцессоров, микро-ЭВМ и даже бортовых вычислителей космических кораблей многоразового пользования [2, 8, 12, 17]. На базе Форты созданы микрокалькуляторы, выполняющие аналитические действия с формулами [29].

Стек и постфиксная форма записи операций, присущие Форту, сейчас известны миллионам пользователей программируемыми калькуляторами [11]. Форт — развиваемый язык, основанный на передовых концепциях структурного программирования. Новые функции и процедуры на Форте задаются в виде слов, входящих в словарь с подсловарями. Это делает Форт практически готовой информационно-поисковой системой, прекрасно приспособленной для создания банков данных. Форт позволяет пользователю разработать свою версию языка. Он может быть единственным посредником между ПЭВМ и пользователем, имея в себе все функции управления ПЭВМ, присущие развитым операционным системам. Язык Форт сочетает достоинства интерпретаторов (диалог и немедленное исполнение слов) с достоинствами компиляторов (высокая скорость счета, малые затраты памяти под реализацию программ). Он дает полный доступ ко всем узлам ПЭВМ.

За рубежом по языку Форт опубликовано много работ, включая и книги [20—23, 25—28, 30—37, 39, 40]. В СССР Форту посвящены пока лишь небольшая книга [1] и отдельные обзорные публикации [2, 4, 13, 16]. А между тем язык Форт входит в программное обеспечение всех классов отечественных ЭВМ — от ПЭВМ «Агат», «Электроника БК-0010», «Искра-226», ДВК-2М, ДВК-3 и др. до ЭВМ серий ЕС и «Эльбрус» [1, 2, 4, 13]. Он широко применяется для программирования массовых зарубежных ПЭВМ (ZX-Spectrum, Commodore, MSX, IBM PC и др.). Поэтому детальное знакомство с ним становится действительно необходимым и своевременным.

Данная книга — первое справочное руководство по основным версиям языка Форт и системам программирования ПЭВМ на его основе. Изложение материала дается в расчете на читателя, освоившего Бейсик. Книгой можно пользоваться и без этого, но надо помнить, что Форт не очень подходит для первого знакомства с языками программирования, ибо имеет слишком выраженное «свое лицо».

С учетом этого, книга начинается с описания версий Форты, имеющих операции над числами с плавающей точкой и все присущие Бейсику функции и графические возможности. Это одновременно ликвидирует пробел в мировой литературе, связанный с отсутствием описания таких версий и их реального применения в практике математических, научно-технических и инженерных расчетов. Описанные в книге расширенные версии форт-систем программирования (FSP88, FSP88M) уникальны по числу и сложности встроенных в них процедур и функций, охватывающих реализации основных численных методов.

В книге дается описание всех базовых слов и многих расширений стандартных и массовых целочисленных версий языка Форт (FORTH-79, fig-FORTH и FORTH-83). Впервые детально рассматриваются графические возможности форт-систем и возможности синтеза звуков, характерные для современных ПЭВМ. В их числе средства цветной графики (включая реализацию быстрой лого-графики и трехмерной графики), задания графических элементов пользователя (графем), смены знаков алфавита и шрифта, синтеза звуковых сигналов. Приведены примеры реализации системных программ — монитора, декомпиляторов, средств задания экранных окон и др.

Ограниченный объем книги сделал целесообразным включение в нее только тех версий языка Форт, которые по идеологии и мнемонике слов близки к стандартным (FORTH-79 и FORTH-83). В их число вошли и наиболее мощные целочисленные версии MVP-FORTH и PC/FORTH для персональных ЭВМ класса IBM PC (включая отечественные модели ЕС-1841 и ЕС-1842, «Искра 1030» и др., совместимые с IBM PC по программному обеспечению). Не удовлетворяющие этим положениям версии с оригинальной мнемоникой слов, например ДССП-80 [5, 9], не рассмотрены.

Пожелания по книге следует направлять по адресу: 117071 Москва В-71, Ленинский просп., 15, Главная редакция физико-математической литературы издательства «Наука».

ВВЕДЕНИЕ В ЯЗЫК ПРОГРАММИРОВАНИЯ ФОРТ

§ 1.1. Общая характеристика языка Форт

В конце 60-х гг. Ч. Х. Муром был разработан язык программирования Форт. Вначале он использовался для управления с помощью ЭВМ аппаратурой радиоастрономической обсерватории (г. Кит-Пик шт. Аризона, США). Поскольку версия языка Форт для применяемой тогда ЭВМ 1130 фирмы IBM допускала длину слов не более чем в пять символов, название FOURTH (четвертый) было сокращено до FORTH.

Форт — первый язык из класса *развиваемых* пользователем (адаптируемых) языков программирования. Основными объектами Форта являются *словарные статьи* с именами, или, проще, *слова*. Они выполняют роль *операторов, команд, процедур и функций*. Типовые версии языка Форт содержат до 200—300 базовых слов. Однако пользователь может вводить свои собственные слова, а также произвольно переименовывать базовые слова. Вновь введенные словарные статьи (слова) после компиляции включаются в словарный набор (словарь) Форта и могут затем использоваться наравне с базовыми словами. Таким образом, пользователь может кардинальным образом изменить структуру и назначение языка — вплоть до полной замены (многие версии Форта допускают стирание базовых слов и замену их словами пользователя).

Способность Форта к столь разной модификации породила множество версий, плохо совместимых друг с другом. Таким образом, достоинство языка грозило обернуться крупным недостатком.

Требовалась стандартизация базового набора слов.

В 1979 г. был разработан стандарт FORTH-79 [36], который лежит в основе большинства распространенных версий языка Форт. В 1983 г. стандарт был уточнен (FORTH-83 [2, 20]). В 1973 г. Ч. Х. Мур основал фирму FORTH Inc., создавшую программное обеспечение на языке Форт. Версии Форта MVP-FORTH (на базе стандарта FORTH-79) и PC/FORTH (на базе стандарта PC/FORTH) включены в программное обеспечение массовых персональных и профессиональных ЭВМ класса IBM PC. Машины этого класса, в том числе отечественные ЕС-1841, «Искра-1030» и др., широко используются в СССР.

В последнее время интерес к языку Форт резко возрос. Появились крупные ассоциации пользователей этим языком, например Eurorep

FORTH User' Group (европейская группа пользователей) и FORTH Interest Group (группа пользователей Форт). Последняя разработала одну из самых массовых конкретных реализаций языка Форт — fig-FORTH. По существу, зародилась индустрия программного обеспечения ПЭВМ на языке Форт. По нему издаются крупные периодические издания (например, журнал «Journal of FORTH application on and research»), имеется множество книг [21, 25, 27, 32—37, 39, 40] и среди них ряд крупных монографий [21, 27, 40]. Форт применяется как для создания простых игр, так и для программирования микропроцессоров, однокристалльных ЭВМ и бортовых вычислителей космических кораблей многоразового пользования по проекту «Спейс-Шаттл» [8, 12].

Помимо свойств развиваемости пользователем, Форт интересен и рядом других ценных качеств. Этот язык уникально сочетает возможности интерпретатора и компилятора. Как и на языке Бейсик, любые фрагменты программ и отдельные слова на Форте могут выполняться сразу после ввода (режим интерпретации). Тем самым создаются благоприятные условия для оперативного диалога ПЭВМ с пользователем и легкой экспериментальной отладки даже больших программ по маленьким частям.

В то же время слова на языке Форт после компиляции плотно «укладываются» в ОЗУ ПЭВМ в виде машинных кодов (внутреннее представление слов). Форт отличается весьма экономичным использованием памяти ПЭВМ. Программы на Форте занимают мало места. Откомпилированные программы выполняются всего в 1,5—3 раза медленнее, чем подобные программы, записанные в машинных кодах или на языке ассемблер. Краткости форт-программ способствует и краткая мнемоническая форма записи многих базовых слов. Например, точка . эквивалентна команде PRINT на языке Бейсик. Однако это свойство неизбежно идет в ущерб наглядности программ. Поэтому не слишком часто используемые команды в современных версиях языка Форт даются полными словами или их общепринятыми и достаточно очевидными сокращениями.

Весьма специфической особенностью Форта является использование специального *арифметического стека* и обратной бесскобочной (*постфиксной*) формы записи операций (предложенной польским математиком Лукасевичем). Это значит, что вначале задаются операнды, а затем указывается операция над ними. С позиций логики это вполне оправданно: прежде чем выполнять какую-либо операцию, надо иметь то, над чем она выполняется. Тем не менее давление привычек, в частности алгебраической формы записи выражений, привело к тому, что именно это свойство Форта отпугивает от него многих программистов — в том числе и профессиональных.

Впрочем, в последнее время подобная логика работы получила неожиданное массовое практическое подкрепление. Многие широко

распространенные модели программируемых микрокалькуляторов (в частности, все отечественные и зарубежные калькуляторы фирмы Hewlett Packard и ряда ее филиалов) используют те же средства, что и Форт, т. е. стек и обратную бесскобочную форму записи операций. Таким образом, появились миллионы пользователей, для которых действия на калькуляторах можно рассматривать как прелюдию к эффективной работе с языком Форт.

Форт или схожие с ним языки программирования реализованы на ряде отечественных ПЭВМ (например, версия Форт для ПЭВМ «Искра-226», построенная на основе FORTH-79). Идеология Форты частично положена в основу диалоговой системы структурного программирования (ДССП) [5, 9], реализованной, в частности, на диалоговых вычислительных комплексах ДВК.2М. Версии FORTH-79 и fig-FORTH включены в состав всех массовых моделей зарубежных ПЭВМ — от дешевых и простых (ZX-Spectrum) до профессиональных (IBM PC и др.). Форт применяется и для программирования ЭВМ серии ЕС [1].

Распространению языка Форт способствуют как уже отмеченные достоинства этого языка, так и органически присущий Форту и ярко выраженный стиль *структурного программирования*. Более того, он требует от пользователя гораздо меньших специальных усилий по структурированию и верификации (проверке) программ, чем другие языки программирования.

Форт считается *системным* и *инструментальным* языком. Он дает доступ к любым узлам ПЭВМ и обширные возможности по управлению периферийным оборудованием. Форт настраивает пользователя на создание собственной операционной системы, возможности и *символика* слов которой могут полностью соответствовать привычной для профессиональной сферы деятельности пользователя. Форт хорошо согласуется с архитектурой современных ЭВМ. Уже есть однокристалльные микро-ЭВМ с базовым языком программирования Форт.

Форт обладает обширными системными функциями, в частности полным доступом к памяти и контролем за ней. Он имеет развитый аппарат преобразования числовых данных, представляемых целыми числами с основанием от 2 (бинарные числа) до 36—72.

В то же время основные версии языка Форт (FORTH-79, fig-FORTH) неудобны для оперативной вычислительной работы. В частности, потому, что они не имеют прямых средств для работы с десятичными числами с плавающей точкой и многих привычных алгебраических, тригонометрических и обратных тригонометрических функций, например $\ln x$, $\exp x$, $\sin x$ и т. д. Конечно, эти функции могут включаться в Форт в виде дополнительных слов. Однако на практике (в условиях ориентации на целочисленную арифметику) это неудобно, а для массового пользователя-непрофессионала даже и неприемлемо.

В связи с этим в последнее время появились расширенные версии

языка Форт, имеющие средства для арифметических операций с десятичными числами и вычисления типовых алгебраических и тригонометрических функций. Такие версии, например FP50 (и описанная далее модифицированная версия FSP88), снимают все ограничения на применение Форты в практике массовых математических, инженерных, научных и учебных расчетов, позволяют решать на нем задачи САПР и моделирования.

Язык Форт пережил второе рождение с появлением и широким распространением персональных ЭВМ. Отчасти это обусловлено уже упомянутыми качествами Форты: малым объемом занимаемой памяти, компактностью программ и высокой скоростью их выполнения. Хотя ограничения на объем памяти ПЭВМ сейчас быстро устраняются, эти качества сохраняют свою привлекательность и полезность, ибо способствуют решению все более сложных задач более дешевыми и доступными средствами.

Версии языка Форт для современных ПЭВМ являются в значительной мере расширенными в сравнении со стандартными версиями (в части базовых слов). В них включаются слова переопределения знаков алфавита, задания графем, создания цветной графики (построения точек, линий, дуг, окружностей и т. д.) и синтеза звуковых сигналов. В связи с этим на языке Форт успешно реализуются сложные обучающие программы. Он может успешно использоваться для построения гистограмм, графиков функций и рисунков, применяемых при подготовке различных информационных материалов. На Форте легко создавать банки данных и информационно-поисковые системы.

§ 1.2. Алфавит и слова языка Форт

Алфавит языка Форт содержит прописные и строчные латинские буквы, а также ряд специальных знаков (например, ?, !, @, #, \$, *, /, +, — и т. д.)^{*)}. Каждому знаку в однозначное соответствие ставится некоторый код. За рубежом используется стандартный код ASCII. В отечественных ПЭВМ предусматривается включение в состав алфавита прописных и строчных букв русского языка. Коды КОИ-7 и КОИ-8, принятые для отечественных ЭВМ, в ПЭВМ обычно не используются, так как у них коды букв русского языка не упорядочены и не предусмотрено дополнительных символов.

Помимо указанных выше знаков ПЭВМ может содержать ряд специальных знаков, в том числе определяемых пользователем графических элементов (графем). Задание графем значительно расширяет возможности ПЭВМ. Так, в виде графем можно задавать специальные математические знаки, буквы различных алфавитов, графические объекты для обучающих игровых программ и т. д.

^{*)} Вывод всех знаков алфавита см. в § 8.1.

С учетом этого количество знаков обычно в пределе достигает $2^8=256$. Соответственно коды знаков в десятичном представлении лежат в пределах от 0 до 255. Поскольку ПЭВМ на элементарном уровне функционирования работает только с двоичными числами — *битами*, для кодирования 256 знаков нужно восемь таких чисел. Они и образуют единицу информации — *байт*.

ОЗУ ПЭВМ можно представить в виде ряда ячеек, в каждой из которых располагается 1 байт (или знак алфавита). Номер ячейки называется ее адресом (ячейку можно уподобить почтовому ящику, приписанному по адресу). Общим числом ячеек определяется объем памяти ОЗУ. Оно измеряется в килобайтах (1К байт=1024 байтов). Часть ОЗУ используется для хранения имен слов языка Форт и их начальных адресов, другая часть — для хранения программ в машинных кодах, названных этими словами. Кроме того, имеются части ОЗУ, в которых организован арифметический стек ПЭВМ (см. ниже) и хранятся значения переменных. Определенная часть ОЗУ используется под *текстовый буфер*, служащий для хранения текстовой информации, редактор Форта и другие системные части языка.

Программа на языке Форт представляет собой совокупность небольших поименованных подпрограмм, выполняющих различные операции и функции. Подпрограммы могут по имени вызывать другие подпрограммы или самих себя (рекурсивное обращение). Имена таких подпрограмм и являются словами языка Форт. Совокупность их образует *словарь языка*.

Центральным понятием в языке Форт является слово. Словом может быть любая цепочка знаков без пробелов между ними, рассматриваемая как единое целое. Словом может быть число (операнд), команда управления (директива или оператор), имя функции, процедуры или подпрограммы.

Процедуры и функции на языке Форт представляют собой поименованные последовательности слов — *словарные статьи*. Имя их после компиляции становится в свою очередь словом. Все слова объединяются в словарь. С помощью специального слова

VLIST (или WORDS)

можно вывести весь перечень слов, присущих данной версии языка Форт. В типовых версиях почти все базовые слова выполняют законченные функции (иногда могут встречаться версии, где небольшая часть слов носит служебный характер и в прикладных программах прямо не используется).

§ 1.3. Определение и переопределение слов

Чтобы ввести в состав языка Форт новое слово, его нужно *определить*. Для этого слово вводится в ПЭВМ по схеме

: Имя Листинг слова ;

Двоеточие (:) указывает на то, что задается новое слово. Вслед за ним идет имя слова, вводимое через пробел. В самом имени пробела не должно быть. Число различаемых знаков в имени зависит от реализации языка и обычно равно пяти-восьми и больше.

Пробел в Форте используется как *разделитель* слов. Поэтому набор любого слова (базового или нового) должен выполняться без пробела. Например, нельзя вводить V LIST или VLI ST вместо VLIST— такая запись будет воспринята как ввод двух отдельных слов и будет соответствующим образом интерпретироваться.

Точка с запятой (;) указывает на окончание листинга слова. Если ввести подобное предложение и завершить ввод нажатием клавиш перевода строки (BK, PC, ENTER, RETURN и т. д.), то произойдет *компиляция* слова. В ходе компиляции новое слово транслируется в машинные коды ПЭВМ и размещается в определенной области ОЗУ ПЭВМ. Некоторые реализации языка Форт, в частности FP50 и FSP88, перед началом компиляции указывают начальный адрес (т. е. номер ячейки памяти), с которого располагаются коды скомпилированного слова.

Если процесс компиляции прошел успешно, появляется знак ОК (от слова *okay* — все в порядке) и вводимое слово (с именем и листингом) включается в словарь языка Форт. В этом легко убедиться, исполнив VLIST (в выведенном листинге появится вновь введенное слово).

Пример. Для вывода на печать текста используется слово ." в начале текста и " в конце. Слова

: DEMO ." ПРИВЕТ " ; ОК

задают слово DEMO, выводящее на экран дисплея текст в виде обычного слова ПРИВЕТ. Если теперь набрать слово DEMO и нажать клавишу перевода строки, слово ПРИВЕТ будет выведено на экран дисплея:

DEMO <ENTER> ПРИВЕТ ОК

Здесь <ENTER> означает нажатие клавиши перевода строки. В дальнейшем указание на нажатие клавиши перевода строки будет опускаться, но всегда подразумеваться.

Форт позволяет *переопределять* слова, т. е. придавать им новое имя:

: Новое имя Старое имя ;

Пример. При исполнении слов

: HELLO DEMO ;

в словарь Форта будет включено новое слово HELLO, по действию эквивалентное слову DEMO. Следует помнить, что старое слово при переопределении сохраняется.

§ 1.4. Операции с редактором

В отличие от большинства языков программирования, например Бейсика, поименованные процедуры (слова) языка Форт после определения утрачивают свой мнемонический листинг в виде записи процедуры знаками языка. Хранится лишь его скомпилированная в машинные коды «копия». Это избавляет пользователя от постоянного созерцания уже отлаженных фрагментов программ и освобождает память ПЭВМ от принципиально ненужной информации.

Однако в практике программирования нередко желательно иметь листинг слов в обычной мнемонике. Например, для использования описываемых им алгоритмов при задании новых слов или редактировании листинга. В связи с этим в Форт включаются специальные редакторы, позволяющие создавать, редактировать и записывать на машинные носители листинги слов, представляемых мнемоникой языка.

Редактор языка Форт — достаточно сложная программа, нередко занимающая в ОЗУ такой же объем, как и другие части форт-системы (иногда даже больший). В связи с этим наряду с хранением форт-редактора в составе самого языка в ряде версий редактор вводится отдельно как самостоятельная программа. После работы с редактором его можно стереть, а освободившуюся от него часть ОЗУ использовать для других целей.

Ввод редактора в действие осуществляется словом EDITOR (*editor* в переводе означает редактор). В большинстве реализаций языка Форт ввод редактора резервирует в ОЗУ ряд областей по 1К байт под хранение 16 строк с 64 знаками в каждой из них ($16 \times 24 = 1024$ байт). Каждая такая область образует лист. Листы обычно нумеруются.

Слово

s LIST

где *s* — номер листинга (от слова *list* — лист), выводит на дисплей листинг листа. Если в лист ничего не заносилось, то все его строки обычно заполняются вопросительными знаками.

Слово

s CLEAR

очищает лист с номером *s*. Если теперь задать слово s LIST (с тем же номером *s*), то незаполненный лист будет иметь вид

(*s*=5)

SCR#5 (наименование листа с номером 5)

0

1

... (номера пустых строк листа)

15

В каждую строку листа после указания ее номера можно поместить либо текст комментария, либо листинг слова. Тексты комментариев обычно даются в круглых скобках, так как для операций с бесскобочной записью такие скобки не нужны.

Чтобы расположить в строке текст, перед ним ставится буква P (от слова *put* — положить). Так, к примеру, выглядит ввод листинга слова DEMO:

```
0 P (ВВОД СЛОВА DEMO)
```

```
1 P : DEMO ." ПРИВЕТ " ;
```

Для загрузки в компилятор созданного листа с листингами слов используется слово

```
s LOAD
```

(от слова *load* — загрузка). При этом заданные в листе слова компилируются и входят в словарь языка Форт. Однако листинг слова сохраняется в листе редактора.

Любое слово и все определенные после него слова *уничтожаются* с помощью слова

```
FORGET Имя
```

(*forget* в переводе означает забывать) При уничтожении слов занимаемая ими область ОЗУ высвобождается.

У некоторых реализаций редактора языка Форт слово *n* LOAD при чистом листе вызывает автоматическое включение считывания, например, с магнитофона. При этом появляется надпись READY CASSETTE (считывание с кассеты) и нужно включить магнитофон с нужной кассетой, содержащей требуемый лист. После считывания листа появляется знак ОК и входящие в лист слова компилируются.

При считывании листа с кассетного магнитофона или диска могут быть сбои и ошибки (например, лист содержит слова, отсутствующие в словаре). Такие ситуации рассматриваются как ошибки.

По завершению создания листа выход из редактора с записью текста листа на диск или кассетный магнитофон производится с помощью слова

```
FLUSH
```

При этом от пользователя запрашивается имя файла (не у всех реализаций Форты) и происходит загрузка листа *s* на носитель (диск или магнитную ленту). Разумеется, перед использованием слова FLUSH накопитель должен быть подготовлен к работе. Например, кассетный магнитофон должен быть включен в режиме записи, а в дисковый накопитель следует вставить диск.

Редактор языка Форт имеет специальные средства по редактированию текста строк и листа. Редактор инициируется (т. е. подготов-

ливается к работе) вводом слова EDITOR и отключается после ввода слова загрузки листа LOAD.

В процессе ввода листинга строк действует система редактирования строки. После ввода каждого символа курсор дисплея (мигающий знак) перемещается на очередное знакоместо и указывает, где будет вводиться новый символ. С помощью клавиш перемещения курсора его можно передвинуть в любое место строки. Набор любого знака приводит к вставлению его в текст строки с раздвижкой ее. Нажатие клавиши забоя (или стирания DELETE) уничтожает знак слева от курсора.

§ 1.5. Понятие о стеке и текстовом буфере

Основными видами обрабатываемой ПЭВМ информации являются числа, текстовые символы и графические объекты. В сущности, обработка любой информации ПЭВМ сводится к обработке чисел. Выше уже отмечалось, что любой текстовый символ представляется числовым кодом. Аналогично этому любой график состоит из точек, заданных числами — координатами (например, x и y в декартовой системе координат) и атрибутами (данными о различных цветовых эффектах). Атрибуты точек и знаков (например, цвета знака и знакоместа, в котором он находится) также задаются числами-кодами.

Специфика языка Форт заключается в том, что он содержит специальный контейнер для оперативного хранения чисел — стек. Любое слово в Форте выполняет определенную операцию, получая нужные для этого входные данные из стека и занося результат операции (если это нужно) также в стек. Таким образом, при выполнении последовательности слов стек как бы перемещается от одного из них к другому, наподобие автомобиля на заводе, перемещаемого на конвейере от одного поста сборки к другому. Такой способ передачи информации от одной процедуры к другой эффективно использует аппаратные возможности современных ПЭВМ и является наиболее быстрым.

Стек можно образно сравнить и с винтовочной обоймой (если считать числа патронами). На рис. 1.1 схематично показан ввод трех чисел в стек и вывод третьего числа из него. Как видно из рисунка, последнее введенное в стек число выводится из него первым. И напротив, первое введенное число выводится последним.

Обычно применяют упрощенную форму изображения стека (рис. 1.2). Арифметический стек Форта — это просто набор ячеек ОЗУ, в которых хранятся числа. Форт обычно оперирует с целыми 16-разрядными и 32-разрядными числами (удвоенной разрядности). В первом случае каждое число занимает две ячейки ОЗУ (2 байта), во втором — четыре. Десятичное число с плавающей запятой занимает обычно 4—6 байт.

Однако и форма рис. 1.2 неудобна при записи большого числа операций со стеком. Поэтому при описании команд Форта принято сокращенное обозначение стека в горизонтальном положении в виде трех черточек:

Числа до операции — — — Числа после операции

Например, если в стек введены три числа 1, 2 и 3, то это обозначается так:

1 2 3 — — — 1 2 3

В скобках дается комментарий.

Первая (на рис. 1.1 или 1.2 сверху) ячейка стека называется вершиной. Ввод чисел всегда производится в вершину стека, а вывод — из вершины. При вводе числа имеющиеся в стеке числа перемещаются на одну ступеньку вниз, а при выводе — вверх.

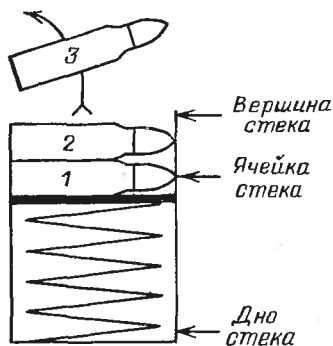


Рис. 1.1. Арифметический стек ФОРТ можно уподобить ружейной обойме

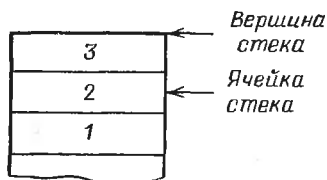


Рис. 1.2. Упрощенное изображение стека

Со стеком Форта могут выполняться самые разнообразные операции: ввод и вывод чисел, вращение (ротация) их в стеке, обмен числами между ячейками и т. д. Таким образом, аналогия между стеком и ружейной обоймой является весьма упрощенной и приемлемой лишь для описания некоторых свойств стека, например ввода и вывода чисел или простых арифметических операций.

Кроме арифметического стека в форт-системе всегда имеется второй стек — стек *возврата*. Он, в частности, следит за очередностью выполнения циклов и подпрограмм. Форт содержит ряд слов, позволяющих воздействовать и на этот стек и осуществлять обмен информацией между стеками. Таким образом, форт-система является *двух-стековой* системой, управляемой *входным потоком* информации.

Текстовая информация представляет собой последовательность кодов всех знаков текста. Например, слово

HELLO

занимает пять ячеек в ОЗУ. В эти ячейки занесены десятичные коды:

Код 72 69 76 76 79

Знак H E L L O

Строка текста в языке Форт может содержать обычно до 64 или 80 символов. Для ее оперативного хранения в ОЗУ выделяется специальная область — *текстовый буфер*. Его положение в ОЗУ характеризуется начальным адресом и длиной (в 64 или 80 байт). Имеются специальные функции для ввода текста в этот буфер, указания начального адреса, операций с текстом и преобразования текста в числовые данные. Общий объем текстовой информации ограничен лишь объемом свободной области ОЗУ, так как Форт имеет возможность вводить тексты не только в текстовый буфер, но и непосредственно в ОЗУ.

§ 1.6. Постфиксная форма выполнения операций

Как уже отмечалось, другой специфической особенностью языка Форт является выполнение операций с применением обратной скобочной записи (или иначе в *постфиксной форме*). Ее особенности отчетливо видны при исполнении арифметических команд сложения, вычитания, умножения и деления.

Рассмотрим следующее выражение:

$3-(2+5)$

Лишь многовековая привычка к такой алгебраической форме записи ведет к тому, что ее нелогичность не бросается в глаза явно. Однако стоит попытаться осмысленно прочесть эту строчку, как нелогичность операции станет очевидна. Действительно начнем: введем число 3 и умножим его на... (тут выясняется, что мы не знаем, на что умножить и какие действия будут далее проведены). Придется отложить это на потом и заняться выяснением того, что будет введено далее, и т. д. На разгадку всех этих вопросов ЭВМ затрачивает время и аппаратные ресурсы.

С позиции логики лучше заранее иметь операнды, с которыми будут производиться операции, а затем уже указать сами операции. Например, так (знак умножения обозначим как *):

$3\ 2\ 5\ +\ *$

Тут вначале вычисляется $2+5=7$, а затем результат умножается на 3.

Можно поступить и так:

2 5 + 3 *

Теперь очевидно, что нужно иметь хранилище чисел — стек. Удобно, извлекая из него числа 2 и 5, занести в стек результат 7. Затем, введя число 3, умножить его на 7 (рис. 1.3). Нетрудно заметить, что постфиксная форма избавляет нас от лишних знаков — скобок. А стек практически исключает необходимость в постоянном контроле статуса локальности и типа переменных, который, например, резко удлиняет

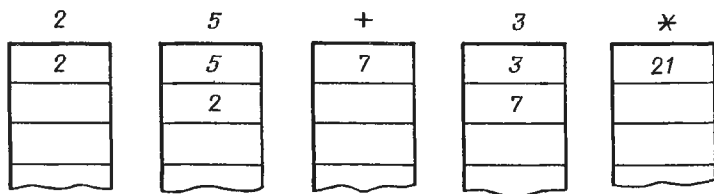


Рис. 1.3. Перемещение чисел в стеке при выполнении арифметических операций

программы на языке Паскаль или ведет к массе ошибок (из-за его отсутствия) при составлении сложных программ на языке Бейсик (впрочем, аппарат локальных переменных иногда вводится в версии языка Форт).

Эти качества, однако, достигаются дорогой ценой. Так, нужно постоянно следить за распределением чисел в стеке и уметь использовать добрый десяток весьма специфических стековых операций. Все это делает форт-программы непонятными для непосвященного пользователя, например привыкшего к «кристальной» ясности Алгола или Паскаля.

Постфиксная форма распространяется и на запись функций. Например, вместо записи ABS (-5), принятой в Бейсике, вычисление абсолютного значения числа -5 на Форте задается в виде:

-5 ABS (дает 5)

Аналогично задается выполнение графических операторов. Например,

80 50 PLOT

строит точку с координатами $x=80$ и $y=50$. На Бейсике это записывается как PLOT 80, 50. После исполнения этой команды числа 80 и 50 удаляются из стека.

Ввиду принципиальной важности знания характера оперативного перемещения чисел в стеке при подготовке программы и описании слов принято упрощенно указывать характер перемещения чисел:

Числа в стеке до операции — — — Числа в стеке после операции
Здесь знаком — — — помечено место слова, выполняющего опи-
сываемую операцию. Вершина стека находится справа от его условного
места.

Полная спецификация слова дается в виде

Имя Числа в стеке Числа в стеке (Комментарий)
до операции — — — после операции

В стеке указываются лишь те числа, которые преобразуются в ходе
операции.

Пример спецификации слова ABS:

ABS n — — — $|n|$ (Заменяет n на $|n|$ на вершине стека)

Следует помнить, что имя слова всегда мысленно помещается
на место знака — — — . Так что при $n=5$ спецификация слова, по
существу, означает следующее: —5 ABS дает 5 на вершине стека.
Знак — — — применяют для обобщенного обозначения различных
слов, имеющих самую различную длину (от одного до семи-восьми
знаков).

Специфический для Форта стиль программирования предпола-
гает, что большое число операций выполняется над операндами и
промежуточными результатами вычислений прямо в стеке. Однако
сложные вычисления трудно или почти невозможно выполнять подоб-
ным образом. Это может потребовать почти фантастических ухищре-
ний по перемещению и сохранению нужных чисел в стеке. Последнее
чем-то напоминает популярную игру «Ханойская башня»: есть три
штыря, на одном нанизан ряд дисков разного и последовательно умень-
шающегося диаметра, которые надо переместить на третий штырь и
разместить в том же порядке. При этом диски перекладываются по
одному, а второй штырь используется как промежуточный. Подоб-
ная ситуация прекрасно известна не только играющим в эту игру,
но и всем пользователям программируемыми микрокалькуляторами
со стеком.

Поэтому в Форт включен аппарат задания поименованных кон-
стант и переменных. Значения их могут браться в виде чисел с вершины
стека либо, напротив, помещаться на вершину стека. Это облегчает
сложные вычисления и делает их более наглядными.

§ 1.7. Режимы работы

ПЭВМ с загруженной форт-системой программирования может
работать в трех основных режимах:

- 1) в режиме *прямого исполнения* (IMMEDIATE);
- 2) в режиме *определения* и редактирования слов (EDITOR);
- 3) в режиме *компиляции* (COMPILE).

В режиме прямого исполнения ПЭВМ управляется *входным потоком* слов, обычно вводимым с клавишного пульта. Функции ПЭВМ при выполнении расчетных операций напоминают функции калькулятора. Примеры выполнения операций в режиме исполнения:

```
2 3 + <ENTER> 5 OK
6 2 / <ENTER> 3 OK
-2 ABS <ENTER> 2 OK
```

Для ПЭВМ признаком работы в режиме исполнения служит отсутствие во входном потоке знака : . В режиме исполнения каждое слово проверяется на соответствие его числу или слову в словаре (т. е. *интерпретируется*). Если слово — число, оно загружается в стек. Если слово включено в словарь, то запускается соответствующая ему подпрограмма в машинных кодах и выполняются все операции, присущие этому слову. Отсутствие у слова указанных признаков ведет к выдаче сообщения об ошибке.

Режим определения и редактирования слов подробно описан в § 1.4. Признаком перехода в этот режим является знак : . Данный режим имеет место до нажатия клавиши перевода строки (ПС, В/К, ENTER, RETURN и т. д.). Как отмечалось выше, определение слова заканчивается вводом знака ; . В процессе редактирования действует редактор Форт.

В режиме компиляции выполняется функция *трансляции* мнемонического листинга слова в машинные коды. После определения и редактирования слова режим компиляции вводится автоматически нажатием клавиши перевода строки. В ходе компиляции, как и в режиме исполнения, все слова, расположенные между знаками : и ; , подвергаются проверке (интерпретации). Исключением является первое слово после знака : , которое запоминается как имя (в дальнейшем мы отождествляем слово с именем, помня о предписанных ему функциях). Если все слова есть числа или элементы словаря, компиляция завершается выводом сообщения ОК. Если какое-либо слово не входит в словарь, компиляция прекращается и ПЭВМ выдает сообщение об ошибке.

Форт имеет ряд специальных слов для управления работой ПЭВМ в режиме исполнения и компиляции, в частности для приостановки ПЭВМ. В режиме исполнения имеется обширный доступ ко всем узлам ПЭВМ для их контроля; например, предусмотрен контроль за содержимым любой ячейки памяти (ПЗУ и ОЗУ).

§ 1.8. Доступ к памяти

В развитии языков программирования можно выделить два диаметрально противоположных подхода к общению пользователя с памятью (ОЗУ, ПЭВМ). В таких языках, как Алгол, Бейсик или Паскаль,

память — это «черный ящик», содержимое которого неизвестно. Оно тщательно «охраняется» от рядового пользователя. Задавая переменные, массивы или процедуры, пользователь обычно не интересуется тем, в какой области памяти они располагаются. Отдельные слова, дающие доступ к памяти, рекомендуются лишь опытным программистам, как средство расширения возможностей языка.

Казалось бы, такой подход — только достоинство. Однако это не так. По мере определения большого числа переменных, массивов и процедур пользователь начинает быстро путаться в их названиях. Еще хуже, что может потеряться оперативный контроль над памятью. В самый неожиданный момент ПЭВМ может вдруг объявить, что доступный объем памяти исчерпан. Кроме того, при решении ряда задач (управление внешними устройствами, увеличение скорости счета и т. д.) неизбежно возникает необходимость в оперативном общении с памятью и в повышении эффективности использования ресурсов ПЭВМ.

Поэтому у ряда языков (таких как ассемблер, Си или Форт) принят совсем иной подход — память открыта пользователю. Она уже не является «черным ящиком», содержимое которого загадка. Напротив, многие действия просто невозможно выполнить, не имея четкого представления о том, как они связаны с памятью ПЭВМ. В их числе задание переменных, констант, массивов, процедур и т. д. При этом на Форте легко организовать и иные способы задания данных, аналогичные используемым в других языках.

Любое слово языка Форт рассматривается в машинном представлении как ряд байтов, хранящихся в ОЗУ, начиная с некоторого *начального адреса*. Форт имеет слова, позволяющие легко найти начальный адрес любого слова (переменной, константы, функции, процедуры), а также начальный адрес основных составляющих форт-системы (вершины стека, текстового буфера и т. д.). Имеется ряд слов по просмотру содержимого памяти (как отдельных ячеек, так и их групп, занятых, например, под хранение чисел), очистке ячеек памяти, заполнению их определенными кодами, обмену содержимым между ячейками, копированию одной области памяти в другую и т. д. Простота организации памяти у ПЭВМ делает освоение этих слов достаточно простым делом.

Такой подход характерен для *профессионального программирования*. Более того, как отмечалось, Форт настраивает пользователя на создание его собственной операционной системы, на разработку своих средств языка программирования. Таким образом, для обычного пользователя освоение языка Форт есть крупный шаг к освоению профессионального подхода к программированию и существенному повышению эффективности создаваемого им прикладного программного обеспечения.

§ 1.9. Сравнение языка Форт с другими языками программирования

Современные ПЭВМ могут программироваться на всех известных языках: Алголе, Фортране, Бейсике, Лого, Паскале, Микро-прологе, Си и др. Неопытный пользователь ПЭВМ, естественно, встречает затруднения при выборе языка и может задать вполне справедливый вопрос: зачем нужен еще один «экзотический» язык, такой, как Форт?

Сравнение языков не должно быть субъективным и подчеркивать достоинства одних языков при умолчании присущих им недостатков. Главная цель такого сравнения — выявить преимущественные области применения того или иного языка. Все отмеченные выше языки достаточно универсальны и способны решать широкий круг задач.

Одно из бесспорных преимуществ языка Форт — поразительная компактность программ. Этому Форт обязан применением стека и постфиксной формы записи операций, а также особым способом организации задания в ОЗУ подпрограмм (слов) в машинных кодах.

По этому параметру Форт превосходит все известные языки программирования. Существует мнение, что из-за компактной формы адресации подпрограмм программы на Форте могут быть даже короче, чем программы на языке ассемблера [17]. Однако большинство разработчиков считает, что они все же длиннее, примерно в 1,5 раза, чем тщательно оптимизированные программы на ассемблере. Тем не менее, только очень опытный программист может написать на ассемблере программы, более короткие, чем получаемые с помощью Форта. И эта возможность до сих пор остаётся спорной.

Компактность программ способствует возможности включения в язык множества новых слов. При весьма скромном объеме ОЗУ (большинство форт-систем ориентировано на ОЗУ с емкостью до 64К байт) в форт-систему можно включить большое число самых разнообразных слов. Например, описанная в гл. 2 расширенная форт-система FSP88 имеет большее число функций, чем такие мощные языки, как Фортран и ПЛ/1. В их число входят: операции над комплексными числами; гиперболические и обратные гиперболические функции; все функции комплексного аргумента; операции над векторами и матрицами; решение систем линейных и нелинейных уравнений; вычисление производных и определенных интегралов; интерполяция; спектральный и статистический анализ; вычисление многих специальных функций и т. д. Все они реализованы не в виде множества отдельных подпрограмм, разбросанных по дискетам или магнитофонным кассетам, а в виде *единой системы* (при этом остается достаточно памяти для дальнейшего расширения системы).

Сказанное означает, что на базе форт-систем возможно создание языка гораздо более высокого уровня, чем это характерно для других

языков (и это при том, что форт-системы в минимальном варианте относятся к языкам не очень высокого уровня).

Другая уникальная возможность форт-систем — их поразительная приспособляемость ко вкусам пользователя. Минимальный вариант Форта, содержащий систему организации словаря, занимает в ОЗУ объем всего около 1К байт. Все остальное пользователь может создать сам либо воспользоваться набором базовых слов (от 100 до 300 в зависимости от реализации). Однако в отличие от всех других (из отмеченных) языков программирования Форт позволяет изменить мнемонику базовых слов, сделав ее более привычной для пользователя. Другими словами, Форт — язык, способный «сам себя создавать» (разумеется, через посредство пользователя).

Как известно, языки программирования в зависимости от характера трансляции команд в машинные коды делятся на *интерпретаторы* и *компиляторы*. Интерпретатор интерпретирует (т. е. распознает) каждое отдельное слово в сходном потоке команд и операндов (чисел) и тут же исполняет его. Это резко облегчает диалог с ПЭВМ: можно немедленно прогнать спорный фрагмент программы, проверить действие любой команды (например, вычислить $\sin x$), т. е. в конечном счете экспериментировать с ПЭВМ. Такие языки, как Бейсик, Лого и Микро-пролог, обычно являются интерпретаторами.

Однако интерпретаторы работают медленно, что схоже с переводчиком иностранных книг, пытающимся сделать перевод по смыслу отдельных слов. Замедление по сравнению со скоростью выполнения команд в машинных кодах доходит до десятков раз, а иногда и выше!

Компиляторы, напротив, разом воспринимают весь текст программы и транслируют его на машинные коды. Это напоминает работу опытного переводчика, охватывающего взглядом большой кусок текста и воспринимающего сразу смысл прочитанного. У компиляторов скорость выполнения скомпилированных программ намного больше (обычно в 10—20 раз), чем у интерпретаторов. Однако подготовка программ у них сложнее; при наличии ошибок приходится многократно компилировать программу, а этот процесс нередко занимает много времени. Зато отлаженная и скомпилированная программа компактна и выполняется быстро. К компиляторам относятся языки Алгол, Фортран, Паскаль, Си и др.

Форт своеобразен тем, что объединяет возможности интерпретаторов и компиляторов, содержит средства как для интерпретации любого отдельного слова, так и для компиляции целиком всей программы. Как и у Бейсика, у Форта есть возможность немедленного выполнения любого слова. Поэтому Форт — язык диалоговый. Правда, у Форта исполнение даже одного слова предполагает, что ранее его словарная статья была скомпилирована и в машинных кодах она хранится в ОЗУ.

Следует отметить, однако, что скорость выполнения в режиме интерпретации у Форта столь же невелика, как у других языков-интер-

претаторов. Более того, у отдельных реализаций языка Форт (например, FP50) она намного меньше, чем у Бейсика. Но зато скомпилированная программа выполняется очень быстро — всего в 1,5—3 раза медленнее, чем аналогичная программа в машинных кодах (и примерно в 10—20 раз быстрее, чем на Бейсике).

Противоречивым является вопрос о достоинствах и недостатках *мнемотики* Форта. Обычно в версиях языка Форт наиболее распространенные слова (их десятки) имеют весьма краткие мнемотические обозначения: точка, двоеточие, отдельные знаки и т. д. Это резко облегчает ввод слов. Однако из-за этого и наличия весьма специфических операций со стекком программы на языке Форт на первый взгляд больше напоминают набор китайских иероглифов, нежели осмысленный текст. Тем, кто привык программировать на Бейсике, а тем более на Алголе или Паскале, подчас трудно привыкнуть к демонической краткости форт-программ. Да и программист, освоивший Форт, чуть ли не через неделю перестает узнавать написанные им же программы.

Бесспорно, это недостаток Форта, причем серьезный. Однако следует отметить, что Форт имеет «противоядие», сглаживающее этот недостаток. Оно заключается в присущем Форту стилю структурного программирования, при котором вся программа разбивается на множество простых фрагментов, каждый из которых содержит не более десяти-тридцати слов и может отлаживаться самостоятельно. При желании не составляет особого труда осмыслить каждый такой фрагмент. В самом деле, когда мы говорим «самолет» или «автомобиль», мы сразу представляем, что за этим словом стоит, и вовсе не анализируем буквальный смысл слова, вроде «самостоятельно летающий вид транспорта». Так и в Форте — фрагмент программы, будучи когда-то детально разработанным и названным определенным именем, далее фигурирует лишь как это имя — слово. В результате пользователь, вначале окунувшийся в магию базовых слов Форта, постепенно переходит к своей привычной мнемотике и манипуляции им же созданными словами.

Недостатком (с позиции пользователя-непрофессионала) многих версий языка Форт является отсутствие ряда привычных операций, имеющих в других языках, например, отсутствие операций над числами с фиксированной и плавающей запятой, отсутствие алгебраических ($\ln x$, $\exp x$, x^y и т. д.) и тригонометрических функций, повсеместно встречающихся в расчетах. В базовых версиях языка Форт (FORTH-79, FORTH-83, fig-FORTH) перечисленных операций попросту нет.

Причина этого, однако, не в недостатках Форта, так как при необходимости все эти функции можно ввести в любую версию языка Форт. Причина в том, что чаще всего пользователь получает версию языка, где данные функции ему приходится задавать самостоятельно,

а это чрезвычайно кропотливая и ответственная работа, которая под силу лишь опытному программисту и математику.

Вот почему, решая вопрос о приобретении (или даже о знакомстве) с Форт-системой, нужно тщательно взвесить ее начальные возможности и назначение. Многие коммерческие форт-системы поставляются в расширенных версиях. Так, имеются форт-системы, ориентированные на программирование микропроцессоров и однокристальных микро-ЭВМ (форт-микроЭВМ); системы, ориентированные на деловые и экономические приложения; системы для научно-технических расчетов, разработки игр, управления внешними устройствами и т. д.

При этом нельзя забывать, что Форт — универсальный язык программирования с обширными системными возможностями, дающий полный доступ к любым узлам ПЭВМ и позволяющий детально изучить ее.

Программисты-профессионалы любят повторять известное высказывание французского короля Карла V: «Я разговариваю по-испански с богом, по-итальянски с женщинами, по-французски с мужчинами и по-немецки с моей лошадью» [14]. Кто знает, имей этот оригинал персональную ЭВМ, возможно, он говорил бы с ней на Форте.

Заканчивая сравнение Форты с другими языками, отметим, что для микро-ЭВМ с ограниченным объемом памяти Форт позволяет решать более серьезные и объемные задачи, чем другие языки программирования. Хотя ПЭВМ быстро совершенствуются и объем их памяти растет, такие свойства языка Форт, как компактность программ, высокая скорость их выполнения, диалоговый режим общения с пользователем и легкая адаптация под его вкусы, всегда будут оставаться привлекательными качествами этого языка.

Эксплуатация форт-систем имеет специфические особенности. Некоторые из них можно отнести к недостаткам: минимальный контроль ошибок, аварийных ситуаций и типов данных, ограниченный набор встроенных типов данных, большое число не вполне очевидных манипуляций со стекком. Эти недостатки — естественная плата за такие достоинства, как: простой синтаксис языка; его полная открытость для пользователя; возможность практически неограниченного (по составу слов) расширения; одновременное выполнение функций базового языка и операционной системы; безупречная структурированность; высокая эффективность (краткость и быстрота исполнения) программ; возможность перехода к символической, удобной для пользователя; возможность задания новых типов данных; удобная словарная организация команд; малые затраты памяти ОЗУ и возможность выполнения программ вне форт-системы (предусмотрена не всегда).

Есть все основания считать, что форт-системы открывают новый путь создания высокоэффективного программного продукта. Их применение не только способствует более рациональному использованию ПЭВМ, но и открывает новые сферы их применения.

ФОРТ-СИСТЕМА ПРОГРАММИРОВАНИЯ ДЛЯ НАУЧНЫХ И ИНЖЕНЕРНЫХ ПРИЛОЖЕНИЙ

§ 2.1. Общая характеристика форт-систем для научных и инженерных приложений

Для знакомства с языком Форт пользователей-непрофессионалов в области вычислительной техники (включая научных работников, инженеров и студентов) большое значение имеет базовый набор слов системы, привычность их мнемоники и соответствие возможностей системы возможностям языка Бейсик как наиболее известного и массового. Это относится к наличию в составе системы операций над числами с *плавающей точкой* (научная нотация), алгебраических и тригонометрических функций, типовых команд графики и синтеза звука.

С этих позиций особый интерес представляют версии языка Форт, имеющие такие возможности. К ним относится версия FP50 [12] (floating point FORTH), реализованная на массовых ПЭВМ ZX-Spectrum и др., построенных на 8-разрядных микропроцессорах Z80 и 8080 (отечественный аналог К580). Эта версия имеет ряд отличительных и выгодных для освоения форт-систем моментов.

1. Мнемоника подавляющего большинства слов FP50 соответствует принятой у наиболее массовой стандартной версии FORTH-79. Однако все операции относятся не только к целым числам (в этом случае они эквивалентны применяемым у FORTH-79), но и к числам с плавающей точкой. Одновременно у FP50 имеется ряд специальных целочисленных операций с повышенной скоростью исполнения (они помечены знаком %).

2. Интерпретатор Форты у FP50 построен в виде программы на языке Бейсик. Это позволяет всем, кто знаком с языком Бейсик, легко разобраться с работой интерпретатора и при необходимости существенно доработать его, не вникая в сложности создания интерпретатора в машинных кодах, доступные лишь программистам высшей квалификации.

3. В FP50 широко используются подпрограммы в машинных кодах, включенные во встроенное ПЗУ интерпретатора языка Бейсик. Поиск этих подпрограмм и их исполнение организованы, как в обычных форт-системах, и происходят значительно быстрее, чем на языке Бейсик. Вместе с тем, такой подход придает версии FP50 все

возможности базовой (для заданной ПЭВМ) версии языка Бейсик, включая операции над числами с плавающей точкой, вычисление различных математических функций, создание графики и синтез звуков. Таким образом, FP50 является многоязычной системой, использующей языки Бейсик и Форт и оперирующей с подпрограммами в машинных кодах.

FP50 обеспечивает ускорение вычислений по сравнению с Бейсиком от 2 до 32 раз (большие числа характерны для целочисленных операций). Благодаря отмеченным свойствам FP50 может быть легко усвоена самыми массовыми категориями пользователей — теми, кто освоил программируемые микрокалькуляторы или простейшие ЭВМ, программируемые на Бейсике. Подчеркивая полезность форт-системы FP50 для учебных целей, следует отметить, что по числу встроенных базовых математических функций и графических команд она заметно превосходит многие версии форт-систем (в том числе MVP-FORTH и FORTH/PC для персональных ЭВМ IBM PC).

Неудобством при работе с системой FP50 является отсутствие средств редактирования слов. Однако в систему входит отдельно загружаемый редактор ED50 (для его загрузки в FP50 используется слово e). При загрузке ED50 текстовый интерпретатор FP50 полностью уничтожается. При этом ряд его функций (модификация знаков алфавита и др.) теряется. Однако ED50 придает системе новые качества — имеется возможность декомпиляции и редактирования любого слова (кроме слов, содержащих машинные коды) в середине словаря. Если стек в системе FP50 может хранить до 300 чисел, то при переходе в редактор ED50 это число уменьшается до 150. В редакторе ED50 есть слово get для возврата в основную систему FP50. При этом необходимо перезагрузить FP50.

Существенным недостатком FP50 является отсутствие развитых средств по работе с программными файлами. В FP50 предусмотрена лишь запись всей системы с помощью кассетного магнитофона. При этом сохраняется вся система со всеми введенными в нее словами. Хотя средства FP50 на уровне слов позволяют проводить модификацию системы, это неудобно и требует от пользователя высокой квалификации и большого внимания. Загрузка из FP50 редактора ED50 и наоборот требует много времени (около 1,5 мин), а потому не может проводиться оперативно. Отметим, что при хранении FP50 на магнитном диске этот недостаток практически устраняется.

В целом система FP50 представляет собой универсальную форт-систему, ориентированную на научно-технические и инженерные приложения и предназначенную для реализации на персональных ЭВМ с небольшим объемом ОЗУ. Система FP50 занимает в ОЗУ суммарный объем около 13К байт. Ради сокращения объема ОЗУ из FP50 исключен ряд полезных системных функций, что заметно обеднило ее инструментальные возможности.

Форт-система FP50 содержит блоки data (заголовки слов и начальные адреса) и routine (скомпилированные в машинные коды словарные статьи). В них также включены программы для компиляции слов и их интерпретаций (хотя основные функции интерпретации, как отмечалось, возложены на дополнительную программу на языке Бейсик).

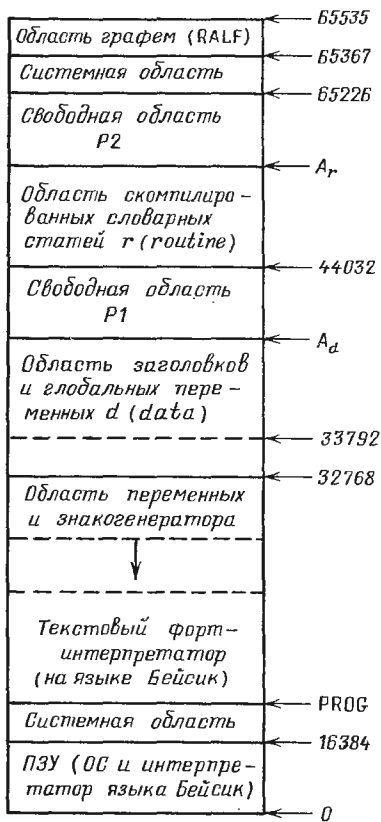


Рис. 2.1. Структура ОЗУ форт-системы FSP88

Специфика FP50 такова, что синтаксический контроль при компиляции слов происходит довольно медленно (1-2 слова в 1 с). Однако скомпилированные программы выполняются гораздо быстрее, чем на языке Бейсик. Это особенно относится к арифметическим операциям, циклам, условным выражениям и др. Наибольшей скоростью выполнения, естественно, отличаются целочисленные операции (примерно в 20 раз быстрее, чем на Бейсике). В связи с этим малая скорость синтаксического контроля слов также относится к недостаткам, характеризующим инструментальные, но не исполнительные возможности системы.

Форт-система программирования для научно-технических и учебных расчетов FSP88, разработанная автором, является существенно модернизированной версией системы FP50. Достаточно отметить, что из занимаемого в ОЗУ ПЭВМ объема памяти более 30К байт лишь около 2К байт осталось от исходной системы (это блоки data и routine, переименованные в d

и r). Многие изменения носят кардинальный характер. К ним относятся: включение в систему типовых функций операционных систем, возможность записи мнемонических листингов слов и их пакетов (16 слов), возможность переименования слов и исключения (скрытия) их имен, возможность записи и считывания скомпилированных словарных статей и соответствующих им блоков словаря. В расширения FSP88 входят удобный монитор и два декомпилятора. Один из них дает

машинные коды скомпилированной словарной статьи, а другой — ее мнемонический листинг. Имеется и ряд других системных функций. Число слов в системе увеличено втрое.

Распределение памяти в системах FP50 и FSP88 показано на рис. 2.1. Ячейки ОЗУ с адресами от 0 до 16384 занимает ПЗУ интерпретатора Бейсик. Над этой областью располагается область системных переменных бейсик-интерпретатора (частично они используются и в форт-системе). Далее идет форт-интерпретатор, выполненный в виде программы на языке Бейсик.

Область ОЗУ *d* (data в FP50) содержит оперативный знакогенератор (768 байт) и область глобальных переменных форт-системы (они обозначены буквами от A до Z, кроме I, J и K). Каждая из этих переменных занимает пять ячеек ОЗУ. С адреса 33792 начинается область заголовков слов.

В области *d* каждое имя слова занимает 6 байт и может содержать любые знаки (кроме нуля в начале и пробелов). Еще 2 байта занимает адрес компиляции, т. е. адрес начала скомпилированной словарной статьи с данным именем.

Таким образом, *N*-е по порядку число имеет абсолютный адрес имени, вычисляемый по формуле

$$A_{dN} = 33792 + (N + 1) \cdot 8.$$

Адрес компиляции A_{rN} вычисляется как

$$A_{rN} = 256 \cdot N_1 + N_2,$$

где N_1 и N_2 — значения старшего и младшего байтов.

Если имя слова содержит менее шести знаков, остальные ячейки имени автоматически заполняются пробелами (код 32). В первую ячейку очередного слова заносится код 0. Он является признаком конца области *d*.

На первый взгляд регулярная организация заголовков слов (словаря) имеет ряд недостатков. Большинство слов в системах FP50 и FSP88 имеет меньше шести знаков, так что часть области *d* пустует. С другой стороны, нельзя задавать и имена с длиной более шести знаков. Однако такая организация имеет и важные достоинства: при ней легче отыскать нужное имя (в том числе по адресу A_r), легко переименовывать и исключать имена, поиск имен идет быстро, нет необходимости в задании признаков начала и конца имен. Это повышает эффективность программ.

После компиляции каждая словарная статья занимает в области *g* определенное место с начальным адресом A_r . Статья всегда кончается кодом 201, соответствующим коду возврата (return) микропроцессора. Словарная статья является цепью машинных кодов и содержит наиболее характерные конструкции вида: обращение к другой статье (call A_r), числа-константы в скомпилированном виде, цепочки в ма-

шинных кодах, флаги (указатели переходов), цепочки ассемблированных слов и слова возврата. Адреса A_d и A_r соответствуют значениям переменных d и r текстового бейсик-интерпретатора.

Цепочки машинных кодов скомпилированных словарных статей плотно следуют друг за другом, образуя так называемый сшитый код. Первые несколько слов форт-систем FP50 и FSP88 (STKSWP, pumber, SPtoCS, 2toCS, CStoD и др.) решают такие системные функции, как ввод чисел на вершину стека, вывод их с вершины, расположение чисел в ОЗУ. Далее форт-система как бы наращивает сама себя: из этих слов образуются новые, из них — другие и т. д. В результате словарные статьи быстро преобразуются в цепочки слов вида call A_r , завершаемые словом return (код слова return — 201). Это слово и является признаком конца словарной статьи.

В режиме непосредственных вычислений на вход форт-системы подается совокупность символов — *входной поток*. Он интерпретируется *текстовым форт-интерпретатором*. Любую совокупность слитных символов интерпретатор пытается трактовать как слово словаря. Для этого имеется специальное слово wgdsc ($A_r=45548$), обеспечивающее поиск по всему словарю до тех пор, пока не встречается A_d с кодом 0 в первой ячейке. Если слово обнаружено, выполняется словарная статья с его A_r (значением переменной Z в интерпретаторе).

Если слова нет в словаре ($Z=0$), интерпретатор пытается воспринять его как число, используя для опознания обычные признаки чисел: целые числа — цепочка цифр, отрицательные — со знаком минус, дробные — с разделительной точкой и в экспоненциальной форме — при наличии знака e или E между мантисой и порядком. Если цепочка знаков — число, оно помещается на вершину арифметического стека для последующего использования.

Если слово не опознано как число, то оно сравнивается интерпретатором с символическими константами типовых управляющих структур IF, ELSE, THEN, BEGIN, WHILE, UNTIL и REPEAT (в версиях FP50 и FSP88 этих слов нет в словаре, но их можно включить в мнемонические листинги словарных статей). При успешном поиске с помощью флажков и слов перехода организуются управляющие структуры с этими словами.

Наконец, если вводимое слово не относится и к этим словам, работа системы останавливается с указанием, что слова нет. Таким образом, текстовый интерпретатор обеспечивает жесткий синтаксический контроль входного потока.

При определении новых слов интерпретатор после контроля заносит в область d коды первой после запятой цепочки знаков. Эта цепочка становится именем нового слова. Возможно обращение к этому имени в самой мнемонической словарной статье, т. е. рекурсивное обращение. После опознания последующих слов формируется последовательность машинных кодов статьи, т. е. происходит ее компиляция.

Если цепочка знаков после двоеточия опознана как ранее введенное слово, система сообщает об этом и предлагает изменить имя.

В системах FSP50 и FSP88 имеются две обширные свободные области ОЗУ. Область P1 частично используется под заголовки новых слов. Ее можно использовать для задания новых переменных, констант, массивов и т. д., ограничив число слов в системе.

Область P2 используется обычно под расширение области g. Имеется также не очень большая свободная область под областью d. Она используется под организацию ячеек стека и текущие переменные форт-интерпретатора.

В верхней части ОЗУ (рис. 2.1) расположена системная область и область графических элементов пользователя — *графем*. Дно области d содержит область *знакогенератора* (каждый из 96 знаков занимает 8 байт, так что эта область имеет $96 \cdot 8 = 768$ байт длины). Нормально в этой области хранится копия знаков из ПЗУ, но любой знак можно изменить с помощью встроенного редактора знаков.

Система FSP88 реализована на массовых 8-разрядных ПЭВМ класса ZX-Spectrum Timex и др. с микропроцессором Z80A. Однако, учитывая легкость переносимости форт-систем [9, 17], ее легко реализовать и на других ПЭВМ с микропроцессорами серии Z80A, 8080, K580 и др.

В целом система FSP88 в отличие от большинства других версий языка Форт ориентирована как на пользователей-непрофессионалов, так и на профессиональных пользователей. Она является эффективным средством для разработки и отладки пакетов прикладных программ в области математических, инженерно-технических, научных и учебных расчетов, машинной графики и др. Также может быть успешно использована и для целей обучения основам программирования на языке Форт, создания обучающих и игровых программ, САПР и др.

Система FSP88 обладает всеми функциональными возможностями языка Бейсик, сочетая их с идеологией и функциональными возможностями языка Форт, включая адаптацию к задачам пользователя, организацию и хранение словаря и естественную реализацию структурного программирования. Фактически FSP88 дает возможность пользователю создать свою версию языка программирования, наиболее близкую по мнемонике слов, их функциональному назначению и профессиональным интересам пользователя.

§ 2.2. Слова ОС и редактора

Форт обладает типовыми возможностями операционных систем и позволяет легко реализовать отсутствующие возможности их. Однако организация текстового интерпретатора на языке Бейсик в системе FSP88 делает целесообразным включение в него некоторых функций ОС.

При загрузке FSP88 появляется наименование системы и меню операций ОС:

ФОРТ-СИСТЕМА ПРОГРАММИРОВАНИЯ
FSP88 В.П.ДЬЯКОНОВ-1988
СЛОВА УПРАВЛЕНИЯ:

sp-ЗАПИСЬ ПРОГР.	vp-КОНТР. ПРОГР
jp-ВВОД ПРОГРАММ	ss-ЗАПИСЬ FSP88
k-ЛИСТИНГ СЛОВА	s-ЗАПИСЬ СЛОВА
j-ЗАГРУЗКА СЛОВА	dz-ЗАМЕНА ЗНАКА
kl-ЛИСТИНГ ЛИСТА	dn-ЗАМЕНА ИМЕНИ
cl-ОЧИСТКА ЛИСТА	pl-ВВОД В ЛИСТ
wl-ВЫВОД С ЛИСТА	nl-ОБМЕН СТРОК
sl-ЗАПИСЬ ЛИСТА	jl-ЗАГРУЗКА ЕГО
vl-ВЕРИФИК.ЛИСТА	i-ГЛАВНОЕ МЕНЮ
ul-ОБЗОР СЛОВАРЯ	f-СТИРАНИЕ СЛОВ
po/1-УПР.ПЕЧАТЬЮ	et-ВВОД ТЕКСТОВ

Рассмотрим возможности системы, реализуемые с помощью этого меню.

Входящие в него слова имеют особый статус. По существу это команды управления ПЭВМ, исполняемые только в режиме непосредственных вычислений (т. е. их нельзя включать в программы). Эти слова исполняются без указания знака ; , фиксирующего конец предложения.

Слово sp (*save program*) используется для записи программ в виде пакетов слов, начинающихся с определенного имени и до конца словаря. Перед ее вводом нужно определить слово, используя слова

' Имя DW

Слово DW (*defined words*) определяет длину записываемых на магнитный носитель областей d и г.

Слово vp (*verify program*) служит для проверки путем сравнения записанной на магнитный носитель программы с хранящейся в ОЗУ.

Слово jp (на клавише j надпись load — загрузка) обеспечивает загрузку программы. При этом создаются ранее отсутствующие части областей d и г, т. е. словарь пополняется новыми словами. Адрес загрузки должен совпадать с тем, который использовался при записи программ.

Слово ss (*save system*) обеспечивает полную запись всей системы FSP88 с измененным шрифтом и набором слов. Это слово позволяет пользователю завершить создание собственной версии FSP88 и сохранить ее на магнитном носителе.

Слово k (на клавише K надпись LIST) выводит на экран дисплея адреса A_d и A_r и листинг последнего введенного слова.

Слово s (*save*) служит для записи на магнитный носитель отдельного слова. В мнемоническом виде словарная статья слова может занимать несколько строк. После записи включается проверка (верификация) ее.

Слово j служит для считывания слова с магнитного носителя.

Слово dz — замена знака. При вводе этой команды FSP88 открывает доступ к оперативному знакогенератору и позволяет изменить

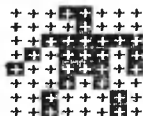
любой знак ПЭВМ. При ее задании форт-система переходит в режим редактирования знаков. Ниже показано превращение знака `m` в фигурку человечка:

РЕДАКТОР ЗНАКОВ:

```
! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @  
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _ `   
a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~ ©
```

Я В Ц Д Э Ф Г Ш И Ж Ы Л Ш Й О П К Ъ Ч Ъ Ч

НОВЫЙ ЗНАК CHR# 109



Подобным образом легко создать знаки другого языка, комплекты графем (например, для изображения электронных схем) и т. д. Все знаки хранятся в оперативном знакогенераторе (внизу области `d`) в виде машинных кодов.

Слово `dn` (*defined name*) позволяет изменить имя любого слова, наметив его словом

'Имя DW

Слово `kl` обеспечивает выдачу листингов введенных ранее слов. Слова объединяются в лист (16 слов по 64 знака в каждом, включая пробелы). Недопустимо слово `kl` до ввода слов в лист (система выходит в Бейсик).

Слово `pl` (*put list*) заносит последнее введенное слово в заданную (по запросу В СТРОКУ?) строку листа.

Слово `wl` (*word list*) инициирует компиляцию слова с указанной по запросу (СО СТРОКИ?) строки.

Слово `cl` (*clear list*) стирает лист целиком или его отдельные строки. При его исполнении выдается запрос СТРОКА?. Если ответ 0, 1, 2, ..., 15, то стирается соответствующая строка; если ответ 16, то стирается весь лист (16 строк).

Слово `pl` обменивает строки листа с номерами n_1 и n_2 (по запросу СТРОКИ?).

Слово `sl` (*save list*) — запись листа в виде двумерного массива текстовой переменной $pS(n_i, t)$, где n_i — номер строки, t — ее текст.

Слово `jl` обеспечивает загрузку всех слов в лист (но без компиляции, последняя происходит по слову `wl`).

Слово `vl` (*verify list*) обеспечивает проверку (верификации) записанного листа путем сравнения его с хранимым в ОЗУ (после `ss` лист очищается).

Слово `i` (*instruction*) выводит инструкцию — меню работы. Нажатие любой клавиши (кроме BREAK и ↓) очищает экран и возвращает систему к работе.

Слово *vl* (от *VLIST*) выводит полный перечень слов словаря (кроме входящих в текстовый интерпретатор слов *IF*, *ELSE*, *THEN*, *BEGIN*, *UNTIL*, *WHILE* и *REPEAT*).

Слово *f* (*forget*) исключает все слова после указанного по запросу (*CO СЛОВА?*). Исключаются как заголовки слов, так и их скомпилированные словарные статьи.

Слова *r0* и *r1* служат для переключения каналов вывода (*r0* — на дисплей, *r1* — на принтер).

Слово *et* (*editor text*) выводит страничный редактор текста. Он позволяет заполнять экран любым текстом, например выполнять надписи на рисунках. При вводе слова *et* в центре экрана появляется мигающий квадрат, перемещаемый в любое место экрана клавишами перемещения курсора. При нажатии клавиши *T* появляется запрос на ввод текста. Введенный и отредактированный в служебных строках текст переносится в нужное место экрана. После этого идет запрос

TEXT ВЕРЕН V/N/F?

Указание *Y* (*yes*) возобновляет перемещение курсора, *N* (*no*) стирает текст и *F* (*finish*) завершает ввод текста. Введенный ранее текст можно стереть, подведя под его первый знак курсор и повторив ввод текста.

Общение с Бейсиком. *FSP88* — многоязычная система. Она использует многие возможности Бейсика прямым обращением к подпрограммам его интерпретатора, хранящимся в ПЗУ. Однако возможно и иное использование Бейсика. Нажатие клавиши \downarrow (курсор вниз) в режиме приглашения к работе переводит *FSP88* в Бейсик. При этом можно в Бейсике выполнять все его команды, например, снимать копии изображения с экрана командой *COPY*, корректировать и выводить на печать значения системных переменных *d* (адрес A_d) и *g* (адрес A_g), просматривать записи программ (командой *LOAD "Имя"* с несуществующим именем) и т. д. На Бейсике реализованы также редакторы знаков и текста комментариев.

В Бейсик *FSP88* переходит и при возникновении ошибок. Возврат из Бейсика в *FSP88* выполняется командой *GO TO n*, где $n=0, 1, \dots, 9$. Команда *RANDOMIZE USR A_r* позволяет исполнить форт-статью с адресом A_r в Бейсике.

Внимание! Ни в коем случае не применяйте команды *RUN* и *NEW* Бейсика, так как они уничтожают системные переменные *FSP88*. Пуск интерпретатора выполняется только командой *GO TO n*, где $n=0, 1, \dots, 9$. Если команды *RUN* или *NEW* были случайно применены, необходима повторная загрузка *FSP88* с носителя.

Как отмечалось, система *FP50* обеспечивает запись новых слов только при записи всей системы. Это очень неудобно. Система *FSP88* дает более обширные возможности. Можно записывать и считывать отдельные слова в мнемоническом виде (слова *s*, *j* и *k*). При этом длина словарной статьи может достигать до пяти строк.

Пакеты слов (до 16 слов с длиной словарной статьи до 64 знаков) могут записываться (и считываться) в виде листов (слова sl, jl, kl и vl).

Особо ценной в FSP88 является возможность записи части системы с заданного адреса. При этом (слова sp, jr и vp) используются скомпилированные словарные статьи. Образно FSP88 можно представить в виде отвертки с ручкой (минимальная версия системы) со сменяемыми вставками. Таким образом, сохраняя минимальную систему, пользователь может дополнять ее библиотеками слов для реализации самых разнообразных задач.

Отметим дополнительные системные возможности FSP88. Они реализуются следующими словами:

' Имя_ _ _ A_r

оставляет адрес компиляции словарной статьи с указанным именем.

' Имя ARD_ _ _ A_d

преобразует адрес компиляции в адрес A_d имени.

' Имя1 ' Имя2 WA

дает распечатку имен с адресами A_d и A_r, начиная с имени Имя1 до имени Имя2.

Результат действия слова WA (*words adres*) представлен ниже:

Compiling * SIND * RND WA ;

SIND	33920	44,249
COS	33928	44,287
COSD	33936	44,303
TAN	33944	44,341
TAND	33952	44,357
ASN	33960	44,395
ACS	33968	44,411
ATN	33976	44,427
ASND	33984	44,443
ACSD	33992	44,481
ATND	34000	44,519
LN	34008	44,557
EXP	34016	44,573
INT	34024	44,589
SQR	34032	44,605
SGN	34040	44,621
ABS	34048	44,637

Stack 0 ---Ok

К системным возможностям FSP88 относится и ряд слов, обеспечивающих доступ к памяти и модернизацию ее. Они будут детально рассмотрены в дальнейшем.

Мнемоника слов системы FP50, как отмечалось, близка к принятой в стандарте FORTH-79. Однако в отличие от FORTH-79 действие слов относится в общем к числам с плавающей точкой. В то же

время в FР50 и FSP88 включен обширный набор специальных слов для операций только с целыми числами. Все эти слова начинаются со знака %.

К системным возможностям FSP88 относится переименование имен слов и задание новых слов, а также изменение функций текстового интерпретатора (для этого достаточно умения программировать на языке Бейсик). Переименованием слов и изменением текстового интерпретатора не следует увлекаться беспредельно. Так, переименование базовых слов затруднит понимание программ, а добавление в текстовый интерпретатор новых возможностей может привести к его чрезмерному расширению и захвату области d. Первым признаком этого является «смазывание» символов, поскольку внизу области d расположен оперативный знакогенератор. Расширение текстового интерпретатора может уменьшить и число ячеек арифметического стека.

Несмотря на указанные предупреждения, редкий пользователь, полюбивший Форт, откажется от возможности подстроить имена слов под свой вкус. Поэтому покажем более подробно, как это делается.

Допустим мы решили изменить слово SINR (синус с аргументом в радианах) на более простое слово SIN. Для этого зададим

```
SINR DW ;
```

Как отмечалось, слово DW инициирует процесс переименования. Далее задаем слово dn и в ответ на вопрос ИМЯ? введем новое имя, т. е. SIN, завершив ввод нажатием клавиши перевода строки. В результате имя SINR будет заменено на SIN (это легко проверить, исполнив слово VL ;).

Далее (начиная с гл. 3) описывается модифицированная система FSP88. В данной же главе примеры даны при использовании учебной системы, в которой сохранена мнемоника базовых слов версии FР50. В табл. 2.1 указаны переименованные слова для модифицированной системы FSP88. Переименования свелись к замене имен наиболее часто встречаемых слов для работы со стеком на более короткие. Кроме того, слова DO и LOOP, применяемые при организации циклов, заменены квадратными скобками [(DO) и] (LOOP). В круглых скобках () даются некомпиллируемые комментарии, а в фигурных { } — цепочки машинных кодов. Эти небольшие изменения позволили резко сократить длину мнемонических листингов словарных статей.

В FSP88 при задании слов мнемонический листинг словарной статьи выводится в верхнюю часть экрана с указанием адресов A_d и A_r, например:

```
Ad=35320          Ar=47886
: ТИП.С ."ФОРТ-СИСТЕМА ДЛЯ НАУЧН
О-ТЕХНИЧЕСКИХ РАСЧЕТОВ." ;
Compiling ТИП.С ;
ФОРТ-СИСТЕМА ДЛЯ НАУЧНО-ТЕХНИЧЕС
КИХ РАСЧЕТОВ.
Stack 0      ---OK
```

Переименованные слова системы FSP88

Имена слов		
FORTH-79	FSP88 (учебная версия)	FSP88M (основная версия)
NEGATE	NEGATE	NEG
—	SINR	SIN
—	COSR	COS
—	TANR	TAN
—	ASNR	ASN
—	ACSR	ACS
—	ATNR	ATN
DROP	DROP	DR
DUP	DUP	DU
?DUP	?DUP	?DU
SWAP	SWAP	SW
OVER	OVER	OV
DEPTH	DEPTH	DPT
—	%FIELD	%FLD
—	DO	
DO	% DO	%
—	LOOP	}
—	- LOOP	-}
—	+ LOOP	+}
LOOP	% LOOP	%}
+ LOOP	% + LOOP	% +}
—	LEAVE	LV
SPACE	SPACE	SP
SPACES	SPACES	SPS
VLIST	VLIST	VL
—	mc	{
—	end	}

В этом примере слово с именем ТИПС (тип системы) выводит на печать краткое назначение системы. Для вывода на печать текстов используются знаки . " и " .

Как отмечалось, форт-система FSP88 позволяет исключать имена отдельных слов из словаря (не исключая, в отличие от слова f, их скомпилированных словарных статей). Другими словами, она делает имена скрытыми. Для этого достаточно задать слово

' Имя CW ;

Соответствующее имя исчезнет из словаря, что легко проверить исполнением слова vl.

Для чего нужно скрывать слова? Отметим две необходимости в этом. Форт приучает пользователя к использованию коротких словарных статей. Так, при использовании листа редактора длина мнемонической словарной статьи не должна превышать 64 символов, включая пробелы.

Однако часто реализовать нужный алгоритм с помощью такой короткой словарной статьи попросту невозможно. Тогда приходится вводить вспомогательные слова. Однако это ведет к «тирании» слов — пользователю становится трудно ориентироваться в бесчисленном множестве основных и вспомогательных слов. Практика показывает, что это наступает, если число слов в системе достигает 500 и выше. Поэтому целесообразно вспомогательные слова после отладки вычеркивать из словаря. Вместо них можно вводить новые слова, переходя таким образом к привычной для пользователя мнемонике.

А не нарушает ли такое вычеркивание слов работу программ, использующих эти слова? Нет! Дело в том, что в скомпилированных словарных статьях идет обращение к словам не по их именам, а по адресам A_r . При вычеркивании слов соответствующие им скомпилированные словарные статьи сохраняются. Однако в непосредственном режиме вычислений, конечно, уже нельзя обращаться к скрытым словам. Впрочем, если адрес A_r такого слова известен, то

A_r EXECUT ;

позволяет исполнить и скрытое слово.

Другая необходимость в скрытии слов часто возникает в учебных программах. Учитель нередко хочет скрыть возможности учебной программы, защитив их от чрезмерно любопытных учеников. Это легко сделать, вычеркнув слово из словаря.

В FSP88 принято следующее правило: основные (разумеется, лишь по мнению автора) слова задаются именами, начинающимися с прописных букв (или цифр). Вспомогательные слова задаются строчными буквами. В расширение FSP88 входит слово CWS, скрывающее в словаре все вспомогательные слова, начиная с указанного:

' Имя CWS ;

При использовании слов CW и CWS печатается скрываемое слово и вслед за ним сообщение СКРЫТО. После частичной или полной очистки словаря необходимо вернуть указатель A_d , т. е. изменить на правильное значение системной переменной d. Для этого следует просмотреть список имен словаря (слово VL ;) и найти адрес A_d последнего слова:


' Имя ARD . ;

Значение переменной d должно на 8 превышать A_d . Для установки d выходим в Бейсик (нажав клавишу \downarrow) и задаем команду

LET $d = A_d + 8$

где A_d нужно понимать как число, индицируемое при исполнении предшествующей команды. Теперь, дав команду GO TO 0 (или 1, 2, ..., 9), вернемся в форт-систему и продолжим работу в ней.

В окончательном варианте системы FSP88 устранено постоянное сообщение о начале компиляции, имеющееся в системе FP50. Кроме того, оперативный контроль числа занятых в стеке ячеек перенесен в служебную строку и предшествует приглашению (см. пример ниже):

СТЕК 0 

Это делает использование экрана более полным.

Однако в учебных целях удобно сохранить сообщение о компиляции и выводить постоянно сообщение о числе занятых ячеек стека в рабочую зону экрана (например, с целью ее копирования командой COPY).

Весьма незначительные изменения текстового интерпретатора на языке Бейсик позволяют модифицировать вывод в нужном нам виде. В этой главе используется учебная версия вывода, а далее — окончательная. Полный листинг текстового интерпретатора FSP88 дан в приложении. При его загрузке следует задать значения A_d и A_r и командой GO TO 9900 запустить загрузку блоков d, r и RALF (графемы букв русского алфавита).

§ 2.3. Монитор и декомпиляторы системы FSP88

Система FSP88 разработана в расчете на ее полную «прозрачность» для пользователя. Она реализуется на двух уровнях — машинных кодах и мнемонических листингов словарных статей.

На уровне машинных кодов пользователь имеет возможность просмотреть коды с любого адреса с помощью слова MON (*monitor*):

A n MON ;

Здесь A — адрес ячейки ОЗУ, с которой идет просмотр, n — число ячеек.

Монитор дает распечатку содержимого ячеек ОЗУ с адреса A с шагом 5 и соответствующее число строк машинных кодов. В каждой строке даются пять кодов, а затем соответствующие им символы (если коды лежат в пределах стандарта ASCII, т. е. от 32 до 126, с расширением в область графем — коды от 127 до 164). Таким образом, монитор позволяет наглядно отображать содержимое ОЗУ системы и оценивать его характер.

Ниже представлен результат действия монитора при вводе слов 33800 100 MON ;

```
33800 110 117 109 98 101 numbe
33805 114 11 172 83 80 r SF
33810 116 111 67 83 35 toCS#
33815 172 46 32 32 32 .
33820 32 32 57 172 50 9 2
33825 116 111 67 83 32 toCS
33830 70 172 67 83 116 F Cst
```

33835	111	68	32	83	172	OD	S
33840	43	32	32	32	32	+	
33845	32	96	172	45	32	-	
33850	32	32	32	32	106		j
33855	172	42	32	32	32	*	
33860	32	32	117	172	47		u /
33865	32	32	32	32	32		
33870	127	172	94	32	32	?	^
33875	32	32	32	138	172		
33880	79	82	32	32	32		DR
33885	32	149	172	65	78		AN
33890	68	32	32	32	159		D
33895	172	115	51	32	32		ε3
33900	32	32	169	172	78		N

В данном случае отчетливо выявляется структура словаря, начиная со слова `pumber`. Хорошо видно, что 6 байт занимают заголовки слов, а 2 байта — их адреса компиляции A_r . Следует, однако, отметить, что монитор воспринимает коды независимо от их назначения. Следовательно, если в ячейках для A_r встретятся коды от 32 до 164, то будут выведены и их символические обозначения, которые могут оказаться не имеющими смысла.

Декомпиляторы FSP88 призваны дать пользователю полную информацию о скомпилированной словарной статье. *Декомпилятор*, включаемый словами

```
' Имя WC ;
```

дает полный перечень машинных кодов словарной статьи с указанным именем. Вот пример его работы:

```
' ANT WC ;
205, 168, 174, 205, 169, 173, 205, 184,
174, 205, 231, 172, 205, 169, 173, 205,
127, 172, 205, 132, 173, 205, 87, 173, 2
01,
БАЙТ = 25
```

Как видно из этого примера, коды отделяются друг от друга запятыми. Кодовый декомпилятор дает также длину (в байтах) словарной статьи. Он очень удобен для анализа словарных статей с машинными кодами.

Символьный декомпилятор обеспечивает «прозрачность» системы на мнемоническом уровне (т. е., по существу, выполняет те же функции, что и дисассемблер для программ, записанных на языке ассемблера). В одном из промежуточных вариантов системы FSP88 был встроен стандартный монитор—дисассемблер. Однако оказалось, что дисассемблирование форт-программ часто дает неверные результаты, ибо дисассемблер не учитывает специфики Форты. Поэтому он был заменен на символьный декомпилятор, учитывающий эту специфику.

Символьный декомпилятор (его проектирование будет описано ниже) решает следующие задачи.

1. Анализирует машинные коды словарной статьи, начиная с адреса A_r .

2. Если встречается код 205 команды call, то последующие два кода воспринимаются как адрес A_r^* . Он дешифруется и используется для поиска в словаре слова. Для этого просматриваются значения A_r по всему словарю. Если встречается $A_r = A_r^*$, то декомпилятор печатает соответствующее слово. Если среди всех A_r нет A_r^* , то печатается сообщение call A_r^* (A_r^* дается в виде десятичного числа). Наконец, если далее встречаются коды кроме 201 и 205, то они просто выводятся на печать как цепочка чисел.

3. Завершается декомпиляция поиском кода 201 слова return и в конце словарной статьи печатается сокращенное слово ret.

Ниже представлены результаты символьной декомпиляции слов АНТ, SIN и CStoD :

```
' ANT DC ;
DU 1+ SW NEG 1+ / SQR LN ret
' SIN DC ;
s3 239 31 56 CStoD ret
' CStoD DC ;
STKSWP call 11249 245 213 197 ST
KSWP ret
```

Слово АНТ соответствует внешнему расширению системы и состоит сплошь из обращений к ранее определенным словам. Поэтому декомпилятор дает обычный мнемонический листинг словарной статьи этого слова, полностью адекватный исходному листингу.

Слово SIN относится к базовым словам и содержит фрагмент словарной статьи в машинных кодах. В слове CStoD имеется обращение call 11249 к адресу $A_r^* = 11249$ (т. е. к ПЗУ бейсик-интерпретатора).

Таким образом, символьный декомпилятор является мощным средством для расшифровки любых слов системы. Он придает системе FSP88 уникальную «прозрачность» для пользователя. Последний может легко ознакомиться с конкретной реализацией любого слова в системе, поучиться программированию на Форте или посмеяться над неудачами разработчика (вероятно, и они не исключены).

Среди известных языков программирования, пожалуй, нет такого, который позволил бы пользователю в характерной для него мнемонике рассматривать листинги не только внешних процедур и функций, но и всех встроенных примитивов языка. Система FSP88 (как и другие версии Форта с декомпиляторами) обладает такой возможностью. Здесь надо отметить, что хранение мнемонических листингов в виде листов-экраниов, принятое в ряде версий форт-систем, требует гораздо большего объема памяти, чем декомпиляция ранее скомпилированных словарных статей. Уменьшение затрат памяти (а они крайне малы!) связаны с тем, что декомпилятор многократно использует блок d с заголовками слов для синтеза мнемонической словарной статьи.

§ 2.4. Слова для ввода-вывода

Слова ввода-вывода FSP88 по своему назначению и символике совершенно аналогичны подобным словам других версий языка Форт. Однако слова для операций с числами относятся в общем случае к числам с плавающей точкой (специальные быстрые операции относятся и к целым числам). Ниже перечислены основные слова ввода-вывода. Десятичные числа с плавающей точкой обозначены буквой *d*, целые числа — буквой *n*, любые числа — буквой *N*. Адрес указывается буквой *A*, строчный символ обозначен буквой *c* (*char*).

Слово

`. d _ _ _`

выводит число с плавающей точкой из вершины стека, число печатается на принтере или появляется на экране дисплея.

Слово

`% n _ _ _`

выводит целое число из вершины стека.

Слово

`CR _ _ _`

переводит строку при печати или выводе на терминал.

Слово

`SPACE _ _ _`

делает один пробел при печати или выводе на терминал

Слово

`SPACES n _ _ _`

делает *n* пробелов при печати или выводе на терминал.

Слово

`" _ _ _`

служит указанием о начале печати текста, указанного следом за этим словом. Содержимое текста не сравнивается со словами словаря.

Слово

`" _ _ _`

служит указанием о конце печати текста и о выполнении следующих за ним слов. Используется после слова `."`

Слово

`COUNT A _ _ _ A + 1 n`

выводит адрес *A*, увеличенный на 1, и код, находящийся по адресу *n*.

Слово

TYPE A n _ _ _

выводит на печать или на терминал n знаков, начиная со знака с адресом A .

Слово

—TRAIL A n1 _ _ _ A n2

для строки текста с начальным адресом A и длиной $n1$ знаков оставляет в стеке начальный адрес A и длину строки с удаленными из нее конечными пробелами $n2$.

Слово

KEY _ _ _ n

останавливает работу ПЭВМ до нажатия клавиши и оставляет в стеке код нажатой клавиши.

Слово

INKEY _ _ _ n

оставляет на вершине стека код клавиши, которая в последний раз использовалась со словом KEY (если KEY не использовалось, то $n=255$).

Слово

EMIT c _ _ _

выводит на печать или на терминал знак с кодом символа c .

Слова

FIELD и %FIELD

создают поле при печати соответственно в 16 и 8 отступов слева от печатаемых знаков.

Слово

EXPECT A n _ _ _

вводит n знаков в ОЗУ, начиная с адреса A .

Слово

QUERY _ _ _

вводит знаки (обычно до 80) в текстовый буфер (его адрес см. при описании слова PAD); структура которого показана на рис. 2.2.

Слово

? c _ _ _

выводит на печать или на терминал числовое значение переменной c (см. § 2.6) с плавающей запятой.

Слово

%? c _ _ _

выводит на печать или на терминал целочисленное значение переменной s .

Слово

TAB n _ _ _ _

перемещает курсор вправо на n -ю позицию, обеспечивая печать или вывод на терминал с заданной позиции.

Слово

AT $n_y n_x$ _ _ _ _

перемещает курсор на позицию, заданную строкой n_y и столбцом n_x .

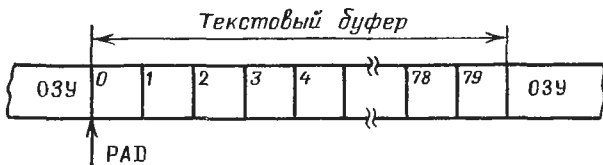


Рис. 2.2. Структура текстового буфера FSP88

Слово

WAIT _ _ _ _

приостанавливает работу ПЭВМ до нажатия клавиши Y. После этого работа возобновляется. Это слово удобно применять, если нужна выдача результатов отдельными частями.

Проиллюстрируем действие слов ввода-вывода несколькими примерами.

Программа МПРОБ

```
Ad=35328                      Ar=47936
: МПРОБ 123 . 456 4 SPACES . 789
  4 SPACES . 3 SPACES . "КОНЕЦ" ;
Compiling МПРОБ
123      456      789      КОНЕЦ
Stack 0   ---OK
```

иллюстрирует вывод трех чисел с интервалом между ними в четыре пробела, заданным словами 4 SPACES, и вывод слова КОНЕЦ с интервалом в три пробела.

Программа ТАБУЛ показывает действие одновременно слов вывода чисел с переводом строки CR и смещением позиций печати словом TAB:

```
Ad=35336                      Ar=48012
: ТАБУЛ 123 ; CR 123 3 TAB . CR
  123 6 TAB . ;
Compiling ТАБУЛ ;
123
  123
    123
Stack 0   ---OK
```

Программа ОСТАНОВ демонстрирует действие слова WAIT:

```
Ad=36424      Ar=59241
: ОСТАНОВ . "НАЖМИТЕ КЛАВИШУ Y!"
WAIT ( КОМАНДА ОСТАНОВА ) CR CR
. "ДОБРОГО ЗДОРОВЬЯ!" ;
Compiling ОСТАНО ;
НАЖМИТЕ КЛАВИШУ Y!
ДОБРОГО ЗДОРОВЬЯ!
Stack 0      ---OK
```

Слова ДОБРОГО ЗДОРОВЬЯ! выводятся на печать или на терминал после нажатия клавиши Y. Эта программа иллюстрирует также выдачу текстов-комментариев, заключенных между словами ." и ".

Диалоговая программа TEXT

```
Ad=35344      Ar=46074
: ТЕКСТ . "ВВЕДИТЕ ТЕКСТ: " QUERY
CR . "ВЫ ВВЕЛИ ТЕКСТ: " PAD 10 T
TYPE ;
Compiling ТЕКСТ ;
ВВЕДИТЕ ТЕКСТ: ABCDEFGHIJKLMNOP
ВЫ ВВЕЛИ ТЕКСТ: ABCDEFGHIJ
Stack 0      ---OK
```

демонстрирует ввод текста в текстовый буфер FSP88 и вывод части этого текста из него. При пуске программа TEXT вначале выводит запрос ВВЕДИТЕ ТЕКСТ :. Затем исполнение слова QUERY останавливает работу в ожидании ввода текста. В приведенном примере введен текст — латинские буквы от А до N. Ввод текста фиксируется нажатием клавиши перевода строки, после чего выполняются последующие операции: выводится сообщение ВЫ ВВЕЛИ ТЕКСТ и по словам PAD (вывод адреса текстового буфера) 10 (число вводимых знаков) и TYPE выводится десять знаков текста. Таким образом, число 10 ограничивает количество выводимых из текстового буфера знаков. Отметим, что ячейки текстового буфера хранят коды знаков текста.

Действие слов KEY, EMIT и INKEY поясняет следующий пример:

```
Ad=35352      Ar=46134
: DKEY KEY EMIT SPACE INKEY .
Compiling DKEY ;
A 65
Stack 0      ---OK
```

При компиляции слова DKEY ПЭВМ останавливается на слове KEY и ждет нажатия любой клавиши. После ее нажатия слово EMIT поглощает с вершины стека код этой клавиши и выводит на печать ее символ. Затем слово SPACE создает пробел, а слова INKEY и . выводят на печать код нажатой клавиши (он запоминается). При компиляции второй части примера (слова INKEY .) выводится код 255.

FSP88 обеспечивает задание текстов в виде последовательности кодов знаков в любую свободную область ОЗУ. Для этого используется слово EXPST. Вывод текстов осуществляется словом TYPE.

Однако эти слова оперируют с текстами фиксированной длины, задаваемой числом n после адреса. Между тем нередко нужно выделить только содержательную часть текста, введенного словом EXPECT. Для этого в конце текста можно ввести пробел (или пробелы). Слово —TRAIL вычисляет число знаков n_r в тексте длиной в n_1 знаков, исключая все конечные пробелы. Следовательно, слово после действия слова —TRAIL приведет к распечатке содержательной части текста с адреса A длиной n_2 . Приведем пример:

```

Compiling 40000 10 EXPECT ;
AB CDE
Stack 0 ---OK
Compiling 40000 15 TYPE ."end" ;
AB CDE      ?????end
Stack 0 ---OK
Compiling 40000 10 -TRAIL TYPE .
"end" ;
AB CDEend
Stack 0 ---OK

```

Здесь с адреса 40000 словом EXPECT задана строка

AB_CDE_____

где знак _ обозначает пробел. Слова 40000 15 TYPE распечатывают пятнадцать символов, включая десять символов, введенных словом EXPECT, и пять вопросительных знаков (они указывают на то, что последующие пять знаков не были введены). Далее слова —TRAIL TYPE распечатывают содержательную часть текста: AB CDE. Конечные пробелы при этом не выводятся. Обратите, однако, внимание, на то, что промежуточный пробел (между буквами B и C) сохранен.

При работе с текстовыми переменными FSP88 использует слово

DELETE A n _ _ _

(delete — уничтожать), которое стирает n символов в цепочке, начиная с начального адреса A . Стирание производится вводом в n ячеек кода 32 (пробел).

Программа

```

Ad=35384      Ar=48035
: DDEL ."ВВЕДИТЕ ТЕКСТ (10 ЗНАКОВ) : " CR PAD 10 EXPECT CR ."РЕЗУЛ
BTAT PAD 5 DELETE:" CR PAD 5 DEL
ETE PAD 10 TYPE ;
Compiling DDEL ;
ВВЕДИТЕ ТЕКСТ (10 ЗНАКОВ):
1234567890
РЕЗУЛЬТАТ PAD 5 DELETE:
67890
Stack 0 ---OK

```

иллюстрирует действие слова DELETE на текст, введенный в текстовый буфер. Вначале в него введен текст 1234567890. В результате

действия слов PAD 5 DELETE первые пять знаков текста заменены пробелами. Если дать слова PAD 10 CDUMP (см. § 2.6), то, просмотрев содержимое первых десяти ячеек текстового буфера, можно убедиться в том, что в первые пять ячеек вместо знаков 1, 2, 3, 4 и 5 занесены коды пробела (т. е. 32).

Отметим, что слово DELETE в системе FSP88 аналогично слову BLANKS в версии fig-FORTH. При работе с текстами в FSP88 используется слово

WORDS — — — n

которое оставляет на вершине стека код первого символа слова, вводимое словом QUERY в текстовый буфер (в других версиях языка Форт это слово имеет иное назначение). Если буфер не заполнен, слово WORD оставляет на вершине стека код нуля. Действие слова WORD в FSP88 несколько отличается от принятого в других версиях языка Форт.

Слова ввода-вывода по обработке текстов обеспечивают эффективные возможности организации словесного диалога и создания текстовых редакторов.

§ 2.5. Слова для управления стеком

Стек FSP88 организован так, что он содержит свыше 100*) ячеек по 5 байт в каждой (рис. 2.3). Таким образом, он может хранить большое число десятичных чисел с плавающей запятой. Слова управления стеком являются одними из самых быстрых. Поэтому специфика программирования на языке FSP88 (и других версиях Форты) заключается в широком применении стека для временного хранения проме-

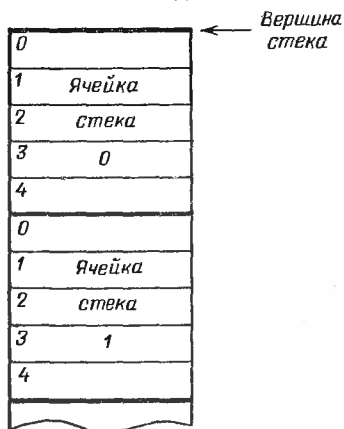


Рис. 2.3. Структура арифметического стека FSP88 и его отдельной ячейки

для временного хранения промежуточных результатов. Поэтому специфика программирования на языке FSP88 (и других версиях Форты) заключается в широком применении стека для временного хранения проме-

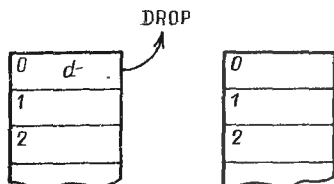


Рис. 2.4. Операция уничтожения числа на вершине стека DROP

*) До 500 в системе FSP88M.

жуточных результатов вычислений. Для эффективного использования стека применяются слова управления стеком, перечисленные ниже (с примерами). Эти слова присущи и другим версиям языка Форт. Описание слов на базе FSP88 удобно тем, что после каждой операции выводится число заполненных ячеек стека.

Ввод чисел в стек не требует особых слов; они вводятся набором и разделяются пробелами:

```
Ad=35392    Ar=48144
: STACK0 1 2 3 4 5 . SPACE . SPA
CE . SPACE . SPACE ;
Compiling STACK0 ;
5 4 3 2 1
Stack 0    ---OK
```

В этом примере вначале введено пять чисел: 1, 2, 3, 4 и 5. Затем словами . и SPACE они выводятся в обратном порядке и разделяются пробелом.

Слово

DROP *d* _ _ _

уничтожает число *d* на вершине стека (остальные числа перемещаются вверх на одну ячейку); см. программу ниже и рис. 2.4:

```
Ad=35400    Ar=48212
: STACK1 1 2 3 4 5 DROP . SPACE
. SPACE . SPACE . ;
Compiling STACK1 ;
4 3 2 1
Stack 0    ---OK
```

Здесь вначале в стек введено пять чисел. Число 5 уничтожено словом DROP, остальные числа выведены четырехкратным использованием слов . и SPACE. Отсутствие чисел в стеке после использования четырех слов вывода показывает, что число 5 действительно уничтожено.

Слово

DUP *d* _ _ _ *d d*

дублирует число на вершине стека. Остальные числа (вместе с исходными) опускаются на одну ступеньку вниз (рис. 2.5):

```
Ad=35408    Ar=48277
: STACK2 1 2 3 DUP . SPACE . SPA
CE . SPACE . ;
Compiling STACK2 ;
3 3 2 1
Stack 0    ---OK
```

Слово

OVER $d_1 d_2 \dots d_1 d_2 d_1$

помещает первое от вершины стека число (подвершину) на вершину (рис. 2.6):

```
Ad=35416    Ar=48326
: STACK3 1 2 3 OVER . SPACE . SP
ACE . SPACE . ;
Compiling STACK3 ;
2 3 2 1
Stack 0    ---OK
```

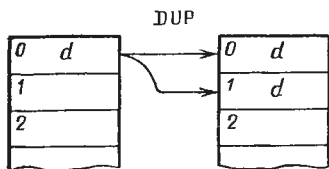


Рис. 2.5. Операция дублирования числа на вершине стека DUP

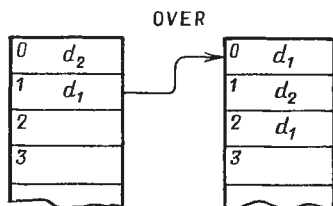


Рис. 2.6. Операция OVER

Слово

?DUP $d \dots d$ (если $d \neq 0$)
 $d \dots$ (если $d = 0$)

дублирует число d на вершине стека, если оно не равно нулю. Если оно равно нулю, состояние стека не меняется (рис. 2.7):

```
Ad=38192    Ar=57670
: STACK4 2 1 0 ?DUP . SPACE . SP
ACE . ;
Compiling STACK4 ;
0 1 2
Stack 0
```

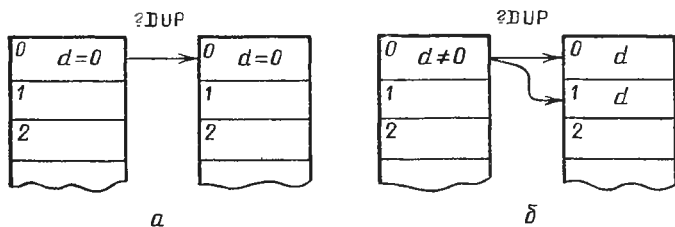


Рис. 2.7. Операция ?DUP при $d=0$ (a) и $d \neq 0$ (б)

Таким образом, при $d \neq 0$ слово ?DUP по действию на стек аналогично слову DUP.

Слово

ROT $d_1 d_2 d_3 \dots d_2 d_3 d_1$

выполняет циклическое перемещение (*rotation* — вращение) чисел в трех ячейках стека (рис. 2.8):

```
Ad=38200 Ar=57713
: STACK5 1 2 3 4 ROT . SPACE . S
FACE . SPACE . ;
Compiling STACK5 ;
2 4 3 1
Stack 0
```

Слово

PICK $n \dots (d_n)$

помещает копию n -го от вершины стека числа на вершину (рис. 2.9):

```
Ad=38216 Ar=57771
: STACK6 1 2 3 4 5 4 PICK . SPAC
E . SPACE . SPACE . SPACE . SPAC
E . ;
Compiling STACK6 ;
2 5 4 3 2 1
Stack 0
```

Числа в ячейках стека можно рассматривать как вектор или одномерный массив. Таким образом, операция n PICK открывает доступ

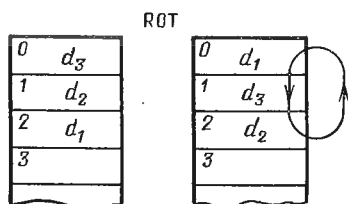


Рис. 2.8. Вращение чисел в 3-х ячейках стека — операция ROT

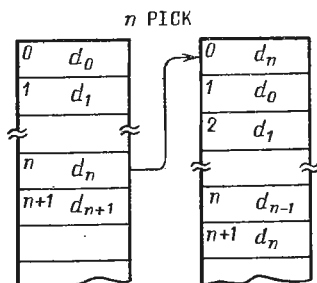


Рис. 2.9. Копирование n -го числа — операция n PICK

к любому числу — компоненту массива в стеке в любом порядке (в отличие от операции ROT, дающей строго последовательный доступ для чисел в трех ячейках стека).

Слово

SWAP $d_1 d_2 \dots d_2 d_1$

обеспечивает обмен содержимым между двумя ячейками стека, включая вершину (рис. 2.10):

```
Ad=38216 Ar=57855
: STACK7 1 2 3 4 SWAP . SPACE .
SPACE . SPACE . ;
Compiling STACK7 ;
3 4 2 1
Stack 0
```

Слово

DEPTH _ _ _ _ n

определяет количество чисел n в стеке и помещает его на вершину стека (рис. 2.11):

```
Compiling 12 34 56 ;

Stack 3
Compiling DEPTH . ;
3
Stack 3
```

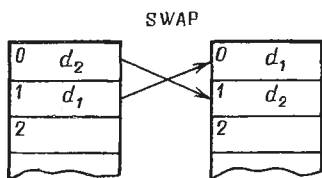


Рис. 2.10. Обмен содержимым двух ячеек стека — слово SWAP

DEPTH

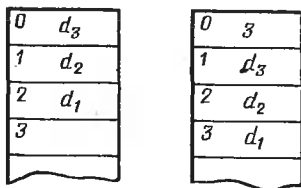


Рис. 2.11. Операция DEPTH

Слово

ABORT $d_1 d_2$ _ _ _ _

уничтожает все числа в стеке (засылает нули во все ячейки стека) — см. рис. 2.12.

Действие слов DEPTH и ABORT иллюстрирует следующая программа:

```
Ad=38224 Ar=57912
: STACK8 1 2 3 . SPACE . SPACE .
SPACE 1 2 3 DEPTH . ABORT DEPTH
. ;
Compiling STACK8 ;
3 2 1 3
Stack 0
```

Слово

ROLL n _ _ _ _ d_n

выполняет циклическое перемещение (вращение) в n ячейках стека (см. рис. 2.13 для $n=4$) и следующий пример:

```

Ad=35464    Ar=48699
: STACK9 1 2 3 4 5 4 ROLL : SPAC
E . SPACE . SPACE . SPACE : ;
Compiling STACK9 ;
2 5 4 3 1
Stack 0    ---OK
    
```

При $n=2$ это слово по действию аналогично слову SWAP, а при $n=3$ — слову ROT. Слово n ROLL дает удобный аппарат для последовательного доступа к n числам в стеке.

ABORT

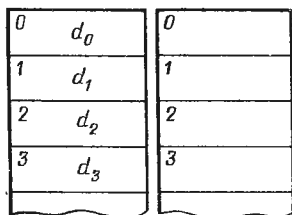


Рис. 2.12. Операция ABORT

n ROLL

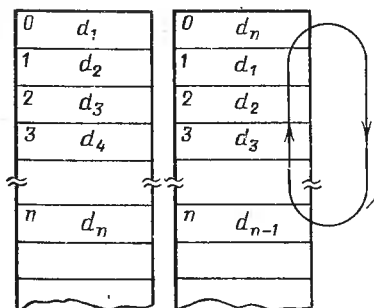


Рис. 2.13. Вращение в n ячейках стека (слово n ROLL)

Помимо арифметического стека (или просто стека) форт-системы содержат стек возврата, следящий за выполнением циклических операций. Имеются два слова для операций со стеками.

Слово

$>R n _ _ _ _$

переносит число n с вершины стека в стек возврата.

Слово

$R > _ _ _ _ n$

переносит число n из стека возврата на вершину стека.

В FSP88 предусмотрен оперативный контроль за количеством чисел в стеке. Однако в составе базовых слов нет средств для просмотра содержимого стека. Тем не менее они легко создаются с помощью имеющихся слов (см. § 2.11, 3.8).

§ 2.6. Слова для адресации, управления памятью, задания переменных и констант

Система FSP88 предоставляет пользователю обширные средства для использования памяти ОЗУ ПЭВМ: адресацию, модификацию адресов, управление памятью и просмотр содержимого ячеек памяти как отдельно (по байтам), так и по группам.

Физически все пространство ОЗУ является ячейками, способными хранить 1 байт информации, т. е. $2^8=256$ значений содержимого каждой ячейки (от 0 до 255). Этим содержимым, к примеру, может быть код любого знака. Как следует из рис. 2.1, часть ячеек заполнена системными кодами FSP88. Однако между ними имеются достаточно протяженные свободные области, которыми пользователь может распоряжаться по своему усмотрению.

Выделенное для этого пространство ОЗУ можно представить в виде определенного числа ячеек с начальным адресом A_d (рис. 2.14). Это пространство можно использовать под хранение массивов чисел, машинных кодов или кодов текстовой информации.

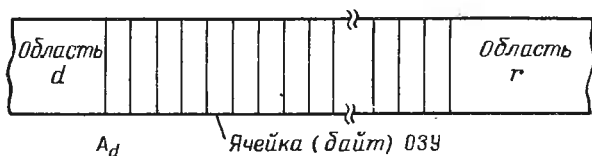


Рис. 2.14. Структура свободного пространства ОЗУ, выделенного под нужды пользователя

Управление памятью выполняется с помощью ряда слов, перечисленных ниже.

Слово

@ $A _ _ _ d$

заменяет адрес A числом d с плавающей запятой, занимающим 5 байт, начиная с начального адреса A и кончая конечным адресом $A+4$.

Слово

! $d _ _ _ A$

заменяет число d с плавающей запятой его начальным адресом A .

Число d представляется в виде

$$d = M \cdot 10^E,$$

где M — нормализованная мантисса (первая до запятой цифра от 0 до 9), E — порядок. Мантисса задается восемью действующими циф-

рами, порядок E — двумя цифрами. Числа d лежат в пределах от $d_{\min}=4 \cdot 10^{-39}$ до $d_{\max}=1 \cdot 10^{38}$. Они определяют разрядную сетку FSP88. Выход d за эти пределы ведет к индикации ошибки.

Для хранения d со знаками порядка и мантииссы нужно 5 байт или 5 смежных ячеек ОЗУ (рис. 2.15). Таким образом, слова @ и ! оперируют с десятичными числами с плавающей точкой d и адресами их первой из пяти ячеек.

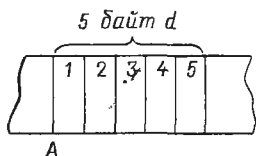


Рис. 2.15. Структура ячеек ОЗУ для хранения числа d с плавающей запятой

Слово
C _ _ _ A

где C — имя переменной (буквы от A до Z), помещает на вершину стека адрес переменной C , т. е. номер первой из пяти ячеек ОЗУ этой переменной.

В ОЗУ ПЭВМ в версии FSP88 выделено 23×5 ячеек для хранения числовых значений d 23 переменных (A, B, C, \dots, Z). Таким образом, сочетание слов $d C !$ используется для придания переменной C числового значения d , а сочетание слов $C @$ — для вызова этого значения на вершину стека. Имеется также слово

? C _ _ _

которое ведет к выдаче числового значения d переменной C на печать или на терминал. Это слово аналогично словам @ . , т. е.

: ? @ . ;

Следующий пример иллюстрирует операции с переменной C :

```
Compiling C . ;
32778
Stack @ ---OK
Compiling 1.23E+25 C ! ;

Stack @ ---OK
Compiling C @ . ;
1.23E+25
Stack @ ---OK
```

Вначале слова $C .$ выводят адрес $A=32778$ переменной C . Затем слова $1.23E+25 C !$ придают переменной C значение $d=1,23 \cdot 10^{25}$. В заключение слова $C @ .$ выводят на индикацию значение d .

Целочисленные операции в FSP88 (они помечены знаком %) оперируют с положительными целыми числами, имеющими значения от 0 до $2^{16}-1=65535$. Эти числа, в частности, используются для ука-

зания адресов ячеек ОЗУ (адресная шина имеет 16 разрядов). Целые числа занимают в ОЗУ две ячейки (рис. 2.16). При выводе на индикацию словами % и %? они представляются в виде чисел с разделительной запятой (например, 12, 345). Это позволяет отличать такие числа от обычных. Хотя для хранения целых чисел нужно всего 2 байта, при присвоении целочисленных значений переменным А, В и др. каждая из них по-прежнему занимает 5 байт.

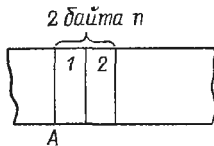


Рис. 2.16. Структура ячеек ОЗУ для хранения целого числа n

Для операции с целыми числами в FSP88 предусмотрен ряд слов (см. ниже).

Слово

%@ A — — — n

заменяет начальный адрес A сдвоенной ячейки на вершине стека его целочисленным содержимым n .

Слово

%! n A — — —

помещает число n в ОЗУ по адресу A (в ячейки с адресами A и $A+1$).

Слово

%? A

вызывает на индикацию целое число n , хранящееся в ячейках памяти с адресами A и $A+1$.

Следующая программа поясняет присвоение переменной C целочисленного значения и вызов его на индикацию:

```

Compiling
C % ;
32,778
Stack @ ---OK
Compiling 123 C %! ;

Stack @ ---OK
Compiling C %@ . ;
123
Stack @ ---OK

```

При указании адреса тысячи от единиц отделяются запятой.

Слово

' _ _ _ _ A

оставляет в стеке начальный адрес слова (или переменной), указанного вслед за ним. Структура заголовка слова дана на рис. 2.17.

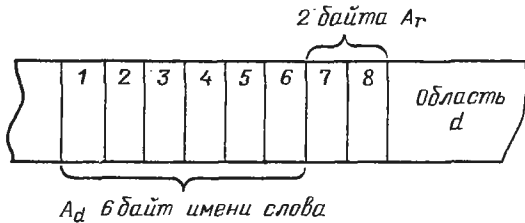


Рис. 2.17. Структура ячеек ОЗУ для хранения слов

Глобальные переменные A, B, ..., Z системы FSP88 при указании их имени дают значения начального адреса пяти ячеек, т. е. спятеренных ячеек ОЗУ (в области *d*). Можно задать дополнительные переменные, зарезервировав под них 5 байт в любой другой области, в том числе и в г. Ниже показано задание такой переменной ALPHA и ее использование:

```
Ar=57951      Ad=37336
: ALPHA 57790 ;
Compiling 123.45 ALPHA ! ;

Stack 0
Compiling ALPHA @ 2 * . ;
246.9
Stack 0
```

Для задания дополнительных переменных нужно после ввода имени указать адрес $A_r + 9$ и, выйдя в Бейсик, увеличить значение переменной *r* на 5 (резервирование 5 байт). Подобным образом можно задавать и массивы.

Содержимое ячеек памяти можно просмотреть, используя следующие слова.

Слово

DUMP A n _ _ _ _

обеспечивает просмотр содержимого *d* спятеренных *n* ячеек памяти, начиная с ячейки с адресом A.

Следующий пример:

```
Compiling 37000 12 DUMP ;
37,000 1
37,005 2
37,010 3
37,015 4
```



```

Compiling 10000 COUNT . CR COUNT
. CR COUNT . CR COUNT . CR COUN
T . CR . ;
0005
0006
0003
0004
40
10005
Stack 0 ---OK

```

Здесь вначале слова 10000 5 CDUMP обеспечивают просмотр содержимого пяти ячеек памяти с адреса 10000 (в данном случае это ПЗУ ПЭВМ, а не ОЗУ). Затем с помощью пятикратного исполнения слов COUNT . CR выводится только содержимое пяти ячеек, а в конце слово . выводит адрес 10004 последней ячейки

Слово

```
ERASE A n _ _ _
```

заносит код нуля в n ячеек, начиная с адреса A , для уничтожения их содержимого.

В следующем примере:

```

Compiling 1 38000 ! 2 38005 ! 3
38010 ! 38000 3 DUMP ;
38,000 1
38,005 0
38,010 3

Stack 0 ---OK

Compiling 38000 5 ERASE 38000 3
DUMP ;
38,000 0
38,005 0
38,010 3

Stack 0 ---OK

```

числа $d=1, 2, 3$ заносятся в специально отведенные ячейки с адресами $A=38000, 38005, 38010$. С помощью слова DUMP просматривается содержимое памяти (и подтверждается занесение указанных d). Затем слова 38005 5 ERASE заносят нули в пять ячеек, начиная с адреса 38005. Последующий просмотр содержимого ячеек показывает, что число $d=2$ уничтожено и заменено на $d=0$. Таким образом, можно избирательно уничтожить числа в определенных областях ОЗУ.

Слово

```
FILL A n b _ _ _
```

заносит значение b (от 0 до 255) в n ячеек ОЗУ, начиная с адреса A . Так, слова $A n 0$ FILL аналогичны словам $A n$ ERASE, т. е. заносят нули в n ячеек с адреса A . При $b=32$ слова $A n 32$ FILL заносят в n ячеек ОЗУ коды пробелов (т. е. формируют n пробелов в текстовом массиве).

В FSP88 имеются средства по перемещению областей памяти в пределах адресного пространства ОЗУ. Это реализуется следующими словами.

Слово

MOVE $A_1 A_2 n$ — — —

помещает содержимое n спятеренных ячеек ОЗУ (т. е. n чисел с плавающей запятой), расположенных с начального адреса A_1 , в область ОЗУ той же длины, но расположенную с адреса A_2 .

Слово

%M $A_1 A_2 m$ — — —

помещает содержимое m вдвоенных ячеек ОЗУ (т. е. m целых чисел), расположенных с начального адреса A_1 , в область ОЗУ той же длины, но расположенную с адреса A_2 .

Слово

SMOVE $A_1 A_2 n$ — — —

помещает содержимое n одинарных ячеек ОЗУ (т. е. n байт), расположенных в ОЗУ с адреса A_1 , в область ОЗУ той же длины, но расположенную с адреса A_2 .

В приведенном примере

```
Compiling 1 38000 ! 2 38005 ! 3  
38010 ! 38000 3 DUMP ;
```

```
38,000 1  
38,005 2  
38,010 3
```

```
Stack 0 ---OK
```

```
Compiling 38000 40000 3 MOVE 400  
00 3 DUMP ;
```

```
40,000 1  
40,005 2  
40,010 3
```

```
Stack 0 ---OK
```

по адресам 38000, 38005 и 38010 задан ввод трех чисел 1, 2 и 3. Слова 38000 3 DUMP показывают, что действительно в ОЗУ сформировалась область длиной 3-5 байт, хранящая в трех спятеренных ячейках эти три числа. Затем слова 38000 40000 3 MOVE переносят содержимое этих ячеек в область ОЗУ с начальным адресом 40000. Слова 40000 3 DUMP вызывают адреса и содержимое этих ячеек на индикацию и показывают, что числа 1, 2 и 3 действительно скопированы в новой области ОЗУ.

Отметим еще ряд слов для работы с памятью.

Слово

+! $d A$ — — — ($d_A \leftarrow d_A + d$)

увеличивает на число d с плавающей запятой содержимое спятеренной ячейки ОЗУ с начальным адресом A .

Слово

$\% +! n A \text{ --- } (n_A \leftarrow n_A + n)$

увеличивает на целое число n содержимое сдвоенной ячейки ОЗУ с начальным адресом A .

Слово

$C! b A \text{ --- }$

записывает байт (число от 0 до 255) в ячейку с адресом A .

Слово

$C@ A \text{ --- } b$

помещает на вершину стека байт b , хранящийся в ячейке ОЗУ с адресом A .

Действие слов $+!$, $C!$ и $C@$ иллюстрирует следующий пример:

```
Compiling 1.234 40000 ! 2.123 40
000 +! 40000 @ . ;
3.357
Stack 0 ---OK
Compiling 40000 ? ;
3.357
Stack 0 ---OK
Compiling 65 39000 C! 39000 C@ .
;
65
Stack 0 ---OK
```

В первом случае к числу 1,234, помещенному в ячейку ОЗУ по адресу 40000, добавляется число 2,123 (слова 2.123 40000 $+!$).

В результате имеем $1,234 + 2,123 = 3,357$. Второй случай иллюстрирует ввод кода 65 словом $C!$ по адресу 39000 и затем вызов его словом $C@$. Слова $P!$ и $P@$ служат для общения с портами микропроцессора.

§ 2.7. Арифметические операции

В форт-системе FSP88 предусмотрены арифметические операции двух классов: с десятичными числами с плавающей точкой и с десятичными целыми числами. Первые специально не обозначают каким-либо знаком (поэтому их можно спутать с целочисленными операциями FORTH-79, fig-FORTH, FORTH-83 и др.). Чтобы не путать виды операций, перед целочисленными операциями в FSP88 ставится специальный знак $\%$. Форт-система FSP88 имеет обширный базовый набор слов для выполнения арифметических операций:

$+ d_1 d_2 \text{ --- } d_1+d_2$ (сложение)
 $- d_1 d_2 \text{ --- } d_1-d_2$ (вычитание)
 $* d_1 d_2 \text{ --- } d_1 \cdot d_2$ (умножение)
 $/ d_1 d_2 \text{ --- } d_1/d_2$ (деление)
 $1+ d \text{ --- } d+1$ (прибавление 1 к d)
 $2+ d \text{ --- } d+2$ (прибавление 2 к d)
 $1- d \text{ --- } d-1$ (вычитание 1 из d)
 $2- d \text{ --- } d-2$ (вычитание 2 из d)
 $ABS N \text{ --- } |N|$ (модуль $|N|$ числа N)

Аналогично для целочисленных операций:

$\% + n_1 n_2 \text{ --- } n_1+n_2$ (целочисленное сложение)
 $\% - n_1 n_2 \text{ --- } n_1-n_2$ (целочисленное вычитание)
 $\% * n_1 n_2 \text{ --- } n_1 \cdot n_2$ (целочисленное умножение)
 $\% / n \text{ --- } n_1/n_2$ (целочисленное деление)
 $\% 1+ n \text{ --- } n+1$ (прибавление 1 к целому n)
 $\% 2+ n \text{ --- } n+2$ (прибавление 2 к целому n)
 $\% 1- n \text{ --- } n-1$ (вычитание 1 из целого n)
 $\% 2- n \text{ --- } n-2$ (вычитание 2 из целого n)

Кроме того, имеются две специфические для целочисленных действий операции.

Слово

$\% MOD n_1 n_2 \text{ --- } n_0$

оставляет на вершине стека остаток от деления двух целых чисел (n_1/n_2) — число n_0 .

Слово

$\% MOD n_1 n_2 \text{ --- } n n_0$

заменяет n_1 и n_2 на результат деления n и остаток n_0 .

Слово ABS может относиться к любым числам (d или n).

Действие слов $*$, $/$ и ABS иллюстрируется примером

```

compiling 2.3 1.75E-12 * . ;
4.025E-12
stack 0 ---OK
compiling 1.2 3E+20 / . ;
4E-21
stack 0 ---OK
compiling -12 ABS . ;
12
stack 0 ---OK
  
```

Действие слов $1-$, $2+$, $2-$ иллюстрируется примером

```

compiling 4.5 1- . ;
3.5
stack 0 ---OK
  
```

```

Compiling 2.1 2+ . ;
4.1
Stack 0 ---OK
Compiling 4.567 2- . ;
2.567
Stack 0 ---OK

```

Действие слов $\%+$, $\%-$, $\%*$ иллюстрируется примером

```

Compiling 12 34 %+ %. ;
46
Stack 0 ---OK
Compiling 124 52 %- %. ;
72
Stack 0 ---OK
Compiling 123 4 %* %. ;
492
Stack 0 ---OK

```

Пример

```

Compiling 38 3 %/ %. ;
12
Stack 0 ---OK
Compiling 38 3 %MOD %. ;
2
Stack 0 ---OK
Compiling 13 %1+ %. ;
14
Stack 0 ---OK

```

демонстрирует целочисленные операции, реализуемые словами $\%/$, $\%MOD$ и $\%1+$.

§ 2.8. Арифметические функции

FSP88 в расчетных приложениях выгодно отличается от целочисленных версий языка Форт обширным набором арифметических функций: их даже больше, чем у типовых версий языка Бейсик. В то же время вычисление большинства функций происходит путем обращения к подпрограммам ПЗУ Бейсика и, следовательно, не связано с большими дополнительными расходами памяти, обусловленными реализацией достаточно сложных численных методов их вычислений.

Лишь слово

NEGATE N — — — — N

как и описанное далее слово SIGN, выдает результат, не зависящий от вида числа (целого или с плавающей запятой). Все остальные

функции являются функциями аргумента в виде чисел с плавающей точкой; в этой форме они и выдают результат. Функции можно разбить на следующие группы: алгебраические, тригонометрические и обратные тригонометрические.

Слова, реализующие алгебраические функции:

SIGN N	_____	1	(если $N > 0$)
N	_____	0	(если $N = 0$)
N	_____	-1	(если $N < 0$)
INT d	_____	int d	(целая часть int d числа d)
$\uparrow d_1 d_2$	_____	$d_1^{d_2}$	(возведение d_1 в степень d_2)
EXP d	_____	e^d	(возведение e в степень d)
SQR d	_____	\sqrt{d}	(корень квадратный из d)
LN d	_____	ln d	(натуральный логарифм числа d)
RND	_____	d	(случайное число d на отрезке $[0, 1]$)
MIN $d_1 d_2$	_____	$d_{\text{мин}}$	(минимальное $d_{\text{мин}}$ из двух чисел d_1 и d_2)
MAX $d_1 d_2$	_____	$d_{\text{макс}}$	(максимальное $d_{\text{макс}}$ из двух чисел d_1 и d_2)

Слова, реализующие тригонометрические функции:

SIND d	_____	sin d	(sin d , d в градусах)
SINR d	_____	sin d	(sin d , d в радианах)
COSD d	_____	cos d	(cos d , d в градусах)
COSR d	_____	cos d	(cos d , d в радианах)
TAND d	_____	tg d	(tg d , d в градусах)
TANR d	_____	tg d	(tg d , d в радианах)

Слова, реализующие обратные тригонометрические функции:

ASND d	_____	arcsin d	(arcsin d в градусах)
ASNR d	_____	arcsin d	(arcsin d в радианах)
ACSD d	_____	arccos d	(arccos d в градусах)
ACSR d	_____	arccos d	(arccos d в радианах)
ATND d	_____	arctg d	(arctg d в градусах)
ATNR d	_____	arctg d	(arctg d в радианах)

Следующий пример иллюстрирует применение слов NEGATE, MIN и MAX:

```

Compiling 2 3 / NEGATE . ;
-0.66666667
Stack 0 ---OK
Compiling -2.145 4E+30 MIN . ;
-2.145
Stack 0 ---OK
Compiling -2.145 4E+30 MAX . ;
4E+30
Stack 0 ---OK

```

Действие слов SIND, TANR и LN иллюстрирует пример

```
Compiling 30 SIND . ;
0.5
Stack 0 ---OK
Compiling 1 TANR . ;
1.5574077
Stack 0 ---OK
Compiling 2 LN . ;
0.69314718
Stack 0 ---OK
```

При использовании слов, реализующих арифметические функции, следует помнить, что как значения аргумента, так и значения функций должны попадать в пределы разрядной сетки ПЭВМ. Например, нельзя вычислить e^{100} , так как результат $2,688 \cdot 10^{43}$ выходит за пределы $d_{\text{макс}} = 1 \cdot 10^{38}$. Следует учитывать также естественные математические границы для задания аргументов ряда функций.

В целом набор функций FSP88 отличается функциональной полнотой. С помощью имеющихся функций можно легко задавать многие другие функции (см. следующую главу).

§ 2.9. Логические операции

Логические операции в FSP88 выполняются над аргументами, являющимися числами или предикатами. Предикаты — это утверждения типа ДА и НЕТ (или ИСТИННО и ЛОЖНО). В качестве предикатов рассматриваются логический нуль (НЕТ) и логическая единица (ДА). Результат логической операции с предикатами также предикат. Значения предиката (0 или 1) приписываются флагу f . Если $f=0$ (флаг опущен), имеем значение НЕТ; если $f=1$, имеем значение ДА (флаг поднят).

Логические операции версии FSP88 представлены тремя словами — AND, OR и NOT. Их достаточно для решения любых логических задач.

Слово

AND $d_1 d_2$ — — — d (логическое умножение)

Пусть d_1 , d_2 и d — предикаты; действие слова AND показано в следующей таблице:

Предикаты		Результат
d_1	d_2	
0	0	0
0	1	0
1	0	0
1	1	1

Слово

OR $d_1 d_2 \text{ --- } d$ (логическое сложение)

Пусть d_1 , d_2 и d — предикаты; действие слова OR показано в следующей таблице:

Предикаты		Результат
d_1	d_2	d
0	0	0
0	1	1
1	0	1
1	1	1

Слово

NOT $d_1 \text{ --- } d$ (логическое отрицание)

Пусть d_1 и d — предикаты; действие слова NOT показано в следующей таблице:

Предикат	Результат
d_1	d
0	1
1	0

Специфика логических операций на языке FSP88 заключается в том, что выходные параметры логических операций могут быть и числами (0 и любое $d \neq 0$). При этом 0 трактуется как логический ноль, а любое $d \neq 0$ — как логическая единица. Это относится и к результату, который имеет два значения: 0 и $d \neq 0$. Если ожидается логический результат ДА, то он будет представлен числом $d \neq 0$, находящимся на вершине стека. Если на вершине стека число, равное 0, а результат должен быть ДА, то на вершину стека заносится число 1 (см. программу):

```
Compiling 3 5 AND . ;
5
Stack 0 ---OK
Compiling 3 0 OR . ;
1
Stack 0 ---OK
Compiling 0 3 OR . ;
3
Stack 0 ---OK
```

С помощью этих слов могут создаваться слова, реализующие и другие логические операции.

§ 2.10. Условные выражения

Условные выражения порождают действия, характер которых зависит от некоторых условий. У FSP88 условные выражения, как и у большинства языков программирования, задаются словами IF (если), THEN (тогда) и ELSE (иначе).

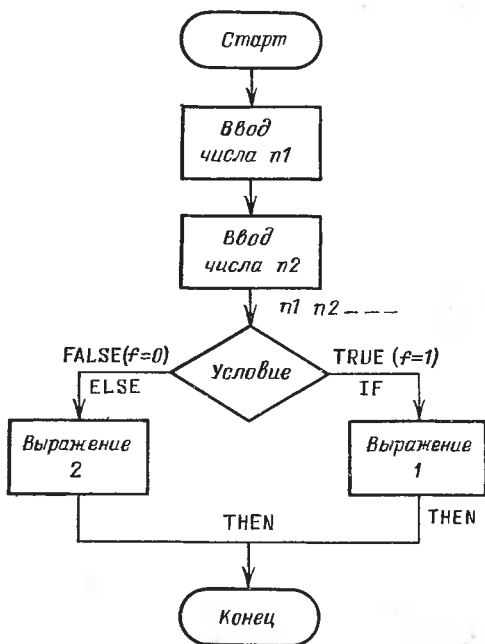


Рис. 2.18. Алгоритм условного выражения IF—ELSE—THEN

Условные выражения с применением этих слов образуют предложение:

f IF <Выражение 1> ELSE <Выражение 2> THEN

Если $f=1$ (флаг поднят), выполняется выражение 1; в противном случае выполняется выражение 2 (рис. 2.18).

Специфика FSP88 заключается в том, что слова IF, ELSE и THEN должны обязательно использоваться совместно. Недопустима конструкция

f IF <Выражение> THEN

Она заменяется конструкцией

f IF <Выражение> ELSE THEN

(выражение выполняется, если флаг поднят). Возможна и такая конструкция:

`f IF ELSE <Выражение> THEN`

Здесь выражение выполняется, если флаг опущен ($f=0$). Состоянием флага управляют следующие слова, реализующие условия (для числа с плавающей запятой):

`0=` `d` `---` `f` ($f=1$, если $d=0$; $f=0$, если $d \neq 0$)
`0<` `d` `---` `f` ($f=1$, если $d < 0$; $f=0$, если $d \geq 0$)
`0>` `d` `---` `f` ($f=1$, если $d > 0$; $f=0$, если $d \leq 0$)
`=` `d1` `d2` `---` `f` ($f=1$, если $d_1 = d_2$; $f=0$, если $d_1 \neq d_2$)
`<` `d1` `d2` `---` `f` ($f=1$, если $d_1 < d_2$; $f=0$, если $d_1 \geq d_2$)
`>` `d1` `d2` `---` `f` ($f=1$, если $d_1 > d_2$; $f=0$, если $d_1 \leq d_2$)

Аналогично для целочисленных операций:

`%0=` `n` `---` `f` ($f=1$, если $n=0$; $f=0$, если $n \neq 0$)
`%=` `n1` `n2` `---` `f` ($f=1$, если $n_1 = n_2$; $f=0$, если $n_1 \neq n_2$)
`%<` `n1` `n2` `---` `f` ($f=1$, если $n_1 < n_2$; $f=0$, если $n_1 \geq n_2$)
`%>` `n1` `n2` `---` `f` ($f=1$, если $n_1 > n_2$; $f=0$, если $n_1 \leq n_2$)
`%<=` `n1` `n2` `---` `f` ($f=1$, если $n_1 \leq n_2$; $f=0$, если $n_1 > n_2$)
`%>=` `n1` `n2` `---` `f` ($f=1$, если $n_1 \geq n_2$; $f=0$, если $n_1 < n_2$)

Ниже поясняется механизм управления флагом с помощью условий для целых чисел:

`2 2 %= ... 1` `2 3 %< ... 1` `2 3 %> ... 0`

В первом случае $n_1 = n_2 = 2$ и условие `%=` дает $f=1$. Во втором случае $n_1 = 2$, $n_2 = 3$ и условие `%<` дает $f=1$. Наконец, в третьем случае $n_1 = 2$, $n_2 = 3$ и условие `%>` неверно, что дает $f=0$.

Программа X?

```
Ad=35472    Ar=48778
: X? 1 < IF ."ЧИСЛО МЕНЬШЕ 1" CR
." ELSE ."ЧИСЛО БОЛЬШЕ ИЛИ РАВНО 1
." CR THEN ;
Compiling 0.5 X? ;
ЧИСЛО МЕНЬШЕ 1

Stack 0    ---OK
Compiling 1.5 X? ;
ЧИСЛО БОЛЬШЕ ИЛИ РАВНО 1

Stack 0    ---OK
```

анализирует введенное число: если оно меньше 1, выдается текст ЧИСЛО МЕНЬШЕ 1; если оно больше 1, выдается текст ЧИСЛО БОЛЬШЕ ИЛИ РАВНО 1. В этой программе используется условное выражение

`IF ELSE THEN` в полном виде.

§ 2.11. Организация циклов

На языке FSP88 возможна организация всех известных видов циклов, необходимых для структурного программирования. *Циклом* называют систему команд, обеспечивающих выполнение определенной серии слов (тела цикла) заданное число раз или до тех пор, пока выполняется (или не выполняется) некоторое заданное условие. Оно может быть в конце или в начале цикла, а также в его теле.

Простейшим является цикл DO — LOOP. Он реализован словами

$n_k + 1$ n_n DO (заголовок цикла)

⟨Тело цикла⟩

LOOP (конец цикла)

Слово

DO $n_k + 1$ n_n — — —

задает некоторой управляющей переменной (в нашем случае I) начальное значение n_n и запоминает число $n_k + 1$, где n_k — конечное значение управляющей переменной. Далее числа $n_k + 1$ и n_n убираются из стека.

Слово

LOOP — — — (I ← I+1, Контроль I)

в стеке никаких действий не производит. Оно увеличивает I на +1 и проверяет I на равенство $n_k + 1$. Если этого равенства нет, управление передается в заголовок цикла и цикл повторяется. Если I превышает $n_k + 1$, то происходит выход из цикла. Внутри цикла I меняется как n_n , $n_n + 1$, $n_n + 2$, ..., n_k . Графически алгоритм цикла DO — LOOP показан на рис. 2.19, а пример его реализации дан ниже:

```
Ad=36352   Ar=56677
: ЦИКЛ (ИМЯ ПРОЦЕДУРЫ)
5 (КОНЕЧНОЕ ЗНАЧЕНИЕ I+1)
1 (НАЧАЛЬНОЕ ЗНАЧЕНИЕ I)
DO (ЗАГОЛОВЕК ЦИКЛА)
I (УПРАВЛЯЮЩАЯ ПЕРЕМЕННАЯ)
. (ПЕЧАТЬ I)
LOOP (ОКОНЧАНИЕ ЦИКЛА)
;
Compiling ЦИКЛ ;
1234
Stack 0 ---OK
```

Если убрать все комментарии (они не компилируются и в ОЗУ не размещаются), то программа задания цикла будет иметь весьма простой вид:

```
Ad=35480   Ar=48849
: DC1 5 1 DO I . SPACE LOOP ;
Compiling DC1 ;
1 2 3 4
Stack 0 ---OK
```

Эти две программы выводят на печать значения управляющей

переменной цикла I . В данном случае $n_k + 1 = 5$, $n_n = 1$ и I принимает значения 1, 2, 3 и 4.

Иногда нужно изменение I с шагом, отличным от $+1$. Для этого используется окончание цикла вида

$+LOOP d$ — — — ($I \leftarrow I + d$, сравнение I с $n_k + 1$)

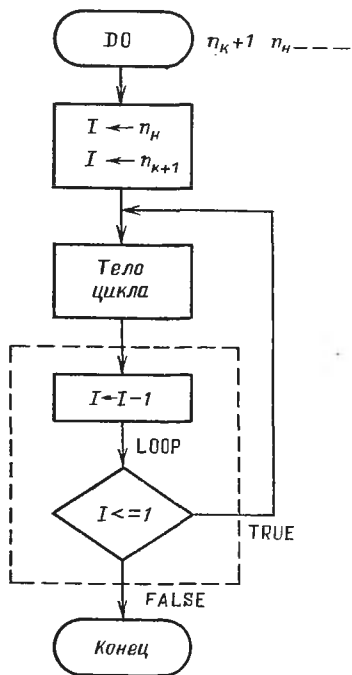


Рис. 2.19. Алгоритм выполнения цикла DO — LOOP

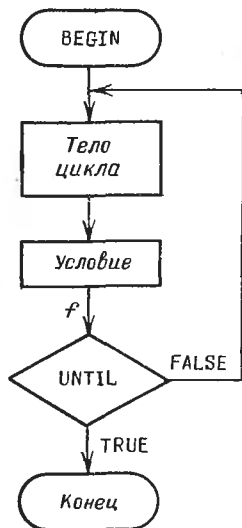


Рис. 2.20. Алгоритм выполнения цикла BEGIN — UNTIL

За исключением того, что I получает теперь приращение d , выполнение цикла происходит аналогично ранее рассмотренному:

```
Ar=57268      Ad=37224
: DC2 11 -4 DO I . CR 2 +LOOP ;
Compiling DC2 ;
-4
-2
0
2
4
6
8
10
```

Stack 0

Тот же цикл без комментария реализован в программе

```
Ar=57308      Ad=37232
: DC3 11 -4 DO I . CR 2 +LOOP ;
Compiling DC3 ;
-4
-2
0
2
4
6
8
10
```

Stack 0

У многих версий языка Форт можно задавать число перед +LOOP отрицательным, что дает отрицательное приращение шагу изменения I. Однако у FSP88 этого нельзя делать. Для получения отрицательного приращения I используется специальное слово

-LOOP *d* — — — ($I \leftarrow I - d$, сравнение I с $n_k + 1$)

При этом число *d*, задающее приращение I, должно быть положительным.

Специфика FSP88 проявляется в том, что шаг *d* может быть любым числом (не только целым):

```
Ar=57348      Ad=37240
: DC4 1 -1 DO I . CR 0.25 +LOOP
;
Compiling DC4 ;
-1
-0.75
-0.5
-0.25
0
0.25
0.5
0.75
```

Stack 0

Циклы DO — LOOP, DO — +LOOP и DO — — LOOP используются, если нужно получить заданное число повторений тела цикла.

Другая конструкция цикла реализуется по схеме (рис. 2.20)

BEGIN (заголовок цикла)

<Тело цикла>

Условие UNTIL (проверка условия)

Здесь слово BEGIN (в переводе «начинать») открывает цикл, а слово UNTIL (в переводе «до») закрывает его. Тело цикла выполняется до тех пор, пока истинно условие перед словом UNTIL. Отличительная черта этого цикла — проверка условия в конце тела цикла. Если условие не выполняется, цикл повторяется. Как только условие

начинает выполняться, цикл прерывается. Характерно, что если условие выполняется сразу, то тело цикла будет выполнено один раз.

В следующей программе исходное число 1 в теле цикла каждый раз удваивается и выводится на печать. В конце цикла оно сравнивается с числом 100:

```
Ar=57388      Ad=37248
: DC5 1 BEGIN 2 * DUP DUP . CR 1
00 > UNTIL DROP ."end" ;
Compiling DC5 ;
2
4
8
16
32
64
128
end
Stack 0
```

Нетрудно заметить, что цикл прервался после того, как результат — число 128 — впервые превысил значение 100.

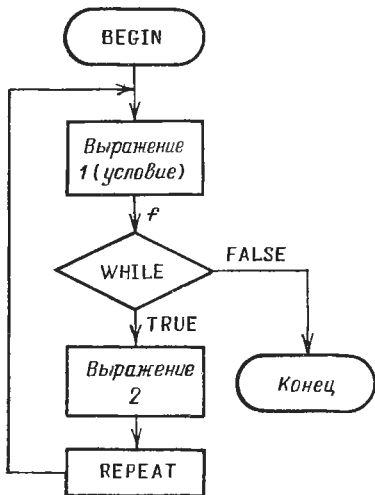


Рис. 2.21. Алгоритм выполнения циклов BEGIN—WHILE—REPEAT

Наконец, третья конструкция цикла (рис. 2.21) имеет вид

BEGIN (заголовок цикла)

<Выражение 1>

Условие WHILE (проверка условия)

<Выражение 2>

REPEAT (конец цикла)

Здесь значение выражения 1 проверяется (слово WHILE) на условие. Если условие выполняется, то вычисляется выражение 2. Иначе происходит выход из цикла.

Действие цикла последней конструкции иллюстрирует следующая программа:

```
Ar=57446      Ad=37256
: DC6 DUF 100 > NOT WHILE 2 * DU
P . CR REPEAT DROP ."end" ;
.Compiling
1 DC6 ;
2
4
8
16
32
64
128
end
Stack 0
```

У FSP88 возможна организация вложенных друг в друга циклов. Программа

```
Ad=38368      Ar=58977
: DC7 3 1 DO 3 1 DO 5 1 DO 1 . .
" " J . ." " " K . CR LOOP LOOP LD
DP ;
```

демонстрирует создание трех циклов. Во внутреннем цикле информацию о значениях управляющих переменных несут следующие переменные: I — внутренний цикл, J — средний цикл и K — внешний цикл. Это видно из распечатки их значений:

```
Compiling DC7 ;
1 1 1
2 1 1
3 1 1
4 1 1
1 2 1
2 2 1
3 2 1
4 2 1
1 1 2
2 1 2
3 1 2
4 1 2
1 2 2
2 2 2
3 2 2
4 2 2

.Stack 0 ---Ok
```

Однако статус переменных I, J, K более сложный, чем описано. Прежде всего следует отметить, что при выходе из цикла эти переменные ведут себя как обычные переменные, сохраняя свои значения,

полученные до входа в цикл. Кроме того, специфичен и статус их в цикле: ближайшей к заголовку цикла всегда является переменная I, вслед за ней может использоваться J и, наконец, K. Таким образом, переменная I может нести информацию о значениях управляющих переменных сразу нескольких вложенных циклов. Пример этого дан ниже:

```
Ar=57604      Ad=37272
: DCB 3 0 DO ."I=" I . CR 6 1 DO
  I . SPACE LOOP CR LOOP ;
Compiling DCB ;
I=0
1 2 3 4 5
I=1
1 2 3 4 5
I=2
1 2 3 4 5

Stack 0
```

Помимо указанных слов для организации циклов служат слова:

```
%DO  $n_k + 1$   $n_n$  _ _ _
```

задает начало цикла с целочисленной управляющей переменной.

```
%LOOP _ _ _ _
```

задает конец цикла с целочисленной управляющей переменной, шаг изменения которой равен +1.

```
%+ LOOP  $n$  _ _ _
```

задает конец цикла с целочисленной управляющей переменной, шаг изменения которой равен n .

```
%I
```

задает значение управляющей целочисленной переменной цикла.

Приведенные ниже примеры

```
Compiling 0 9 DO I . 2 -LOOP ;
97531
Stack 0
Compiling 6 1 %DO %I %. %LOOP ;
12345
Stack 0
Compiling 9 0 %DO %I %. 2 %+LOOP
;
02468
Stack 0
```

иллюстрируют применение этих слов. Они показывают также, что циклы могут исполняться в непосредственном режиме работы, т. е. без объявления знаком (:).

Целочисленные циклы выполняются быстрее, чем циклы, у которых управляющая переменная имеет числовые значения с плавающей запятой. Однако эта разница нередко теряется, если выполнение тела

цикла требует значительного времени. В таких случаях нет особой необходимости использовать целочисленные циклы, зались слов которых более громоздка (они помечены знаком %).

До сих пор в циклах рассматривалось их естественное окончание. У FSP88 имеется два управляющих слова, предназначенных для организации немедленного выхода из цикла:

LEAVE — — — (Выход из цикла DO — LOOP)

%LEAVE — — — (Выход из цикла %DO — %LOOP)

Отличие их лишь в том, что первое используется для выхода из обычного, а второе — из целочисленного цикла. Как только в цикле встречается слово LEAVE (или %LEAVE), управляющая переменная принимает конечное значение и цикл завершается досрочно. При этом выполняются следующие за циклом слова.

Слова LEAVE и %LEAVE обычно используются совместно со словами условных выражений. Это обеспечивает выход из цикла при определенных условиях. Следующие примеры:

```
Ar=57676      Ad=37280
: DLEAVE 5 1 DO CR I . SPACE 2 *
  DUP DUP . 50 > IF LEAVE ELSE TH
EN LOOP SPACE ."end" ;
Compiling 1 DLEAVE 20 DLEAVE ;
```

```
1 2
2 4
3 8
4 16 end
1 40
2 80 end
Stack 2
```

иллюстрируют такое применение. Здесь в цикле организовано умножение каждый раз на 2 исходного числа. Если результат превышает 50, слово LEAVE преждевременно завершает цикл. В первом примере цикл завершается естественным образом, во втором — после исполнения слова LEAVE.

Циклы широко используются при создании различных дополнительных системных функций. Так, ниже показан листинг слова. S, обеспечивающий вывод всех ранее введенных в стек чисел с очисткой стека:

```
Ar=57762      Ad=37288
: .S DEPTH 0 %DO . CR %LOOP ;
Compiling 1 2 3 4 5 ;
```

```
Stack 5
Compiling .S ;
5
4
3
2
1
Stack 0
```


Число циклов здесь задается количеством чисел в стеке и вычисляется с помощью слова DEPTH в заголовке цикла.

В другом листинге (слова ?S):

```
Ad=35488      Ar=48881
: ?S DEPTH %1+ 1 %DO %I PICK . S
R %LOOP ;
Compiling 1 2 3 4 5 ;
```

```
Stack 5      ---OK
```

```
Compiling ?S ;
```

```
5
4
3
2
1
```

```
Stack 5      ---OK
```

числа выводятся из стека на индикацию с сохранением их в стеке. Последовательный вызов чисел выполняется с помощью слова PICK.

Для ускорения этих операций циклы заданы целочисленными. Предполагается, что состояние стека нормальное, т. е. он пуст или в нем имеется одно или более чисел (об особом случае заполнения стека в обратном порядке и коррекции слов .S и ?S говорится в § 3.2).

§ 2.12. Слова для управления

Слова управления служат для передачи управления от одной части программы к другой, а также для передачи управления от одного устройства на другое. К этим командам относится часть ранее описанных слов: IF, THEN, ELSE, BEGIN, UNTIL, WHILE, REPEAT и LEAVE, а также слова для реализации операции сравнения. Ниже приводится описание нескольких дополнительных слов, относящихся к этой группе.

Слово (от *exit* — выход)

```
EXIT _ _ _
```

обеспечивает останов счета с возвращением контроля к терминалу. Используется обычно в командах управления (но не циклов).

Действие этого слова демонстрирует следующий пример:

```
Ad=35496      Ar=48914
: DEXIT DUP 0< IF EXIT ELSE ."LN
(x) = " LN . THEN ." end" ;
```

```
Compiling 2 DEXIT ;
```

```
LN(x)=0.69314718 end
```

```
Stack 0      ---OK
```

```
Compiling -2 DEXIT ;
```

```
Stack 1      ---OK
```

Здесь введенное в стек число x сравнивается с 0. Если оно больше 0, то вычисляется $\ln x$, а в конце выводится сообщение "end". Если $x < 0$, исполняется слово EXIT, что ведет к немедленному прекращению работы и возврату к управлению терминалом (обратите внимание, что слово "end" в этом случае не выводится, т. е. прерывание выполнения программы произошло до ее окончания).

Слово (от *quit* — покидать)

QUIT _ _ _

очищает стек возврата и передает управление терминалу. Если в приведенном выше примере заменить слово EXIT на QUIT, то результат выполнения программы будет тот же.

Слово

EXITLP _ _ _

используется при работе с принтером для останова печати.

Слово

EXIT%L _ _ _

обеспечивает выход из программы, в том числе из целочисленных циклов. Это хорошо иллюстрирует следующий пример:

```
Ad=35504   Ar=48955
: DEXIT%L 5 1 %DO XI %. EXIT%L %
LOOP ;
Compiling DEXIT%L ;
1
Stack 1    ---OK
```

в котором наблюдается выход из цикла уже при первом ходе.

Слово

WAIT _ _ _

обеспечивает останов вычислений. Работа возобновляется при нажатии клавиши Y (см. также § 2.4, где дан пример на использование этого слова).

Слово (от *execute* — выполнение)

EXECUT A_r _ _ _

играет особо важную роль. Оно обеспечивает выполнение слова, имеющего начальный адрес A_r . Тут особенно важно то, что адрес может относиться к слову, которое пока не определено (т. е. отсутствует в словаре). Это не препятствует компиляции программы, имеющей фрагмент A_r EXECUT. Следующий пример иллюстрирует применение слова EXECUT:

```

Compiling 1 SINR .;
0.84147098
Stack 0 ---OK
Compiling SINR .;
44233
Stack 0 ---OK
Compiling 1 44233 EXECUT .;
0.84147098
Stack 0 ---OK

```

Здесь вначале вычисляется значение $\sin 1$ (угол в радианах) с применением функции SINR. Затем определяется адрес 44233 этого слова. Наконец, в конце фрагмент программы 1 44233 EXECUT ; показывает, что при этом вновь вычисляется $\sin 1$ без явного указания слова SINR.

Таким образом, слово EXECUT обеспечивает создание структурированных сверху вниз программ, в состав которых включаются фрагменты, составление которых пользователь откладывает на более позднее время.

Слово

ABORT $d_1 d_2 d_3$ — — —

(*abort* — прерывание) очищает стек возврата и арифметический стек и возвращает управление терминалу (см. также § 2.5). Следующий пример наглядно иллюстрирует это:

```

Ad=35512   Ar=48987
: DABORT 1 2 3 ABORT 4 5 6 ;
Compiling DABORT ."end" ;

Stack 0 ---OK

```

Обратите внимание на то, что введенные в стек числа 1, 2 и 3 уничтожены, а числа 4, 5 и 6 оказались не введенными, так как работа была прервана словом ABORT.

§ 2.13. Задание слов в машинных кодах и общение с другими языками

Как отмечалось, компилятор FSP88 — достаточно эффективное средство, создающее машинное представление (в кодах) команд, которые лишь незначительно превышают по длине аналогичные команды, записанные непосредственно в машинных кодах или на языке ассемблера. Только весьма опытный программист способен создавать более эффективные программы. Таким образом, при подготовке программ на языке FSP88 необходимость в задании команд в машинных кодах особо не ощущается. Тем не менее такая возможность предусмотрена (средствами как самого языка, так и встроенного в FSP88 монитора).

Для задания фрагментов программ в машинных кодах используется выражение

mc $c_1 c_2 \dots c_n - 1$

Здесь заголовок mc (*machine code* — машинные коды) указывает на то, что все последующие числа c_1, c_2, \dots, c_n интерпретируются не как операнды, а как машинные коды. Число -1 указывает на окончание цепочки кода. В мнемоническом листинге программы число -1 заменяется словом end.

Программы в машинных кодах, введенные с помощью FSP88, могут исполняться как на FSP88, так и на других языках, например на Бейсике или прямо в машинных кодах.

Рассмотрим следующий пример. Пусть в двойные регистры BC микропроцессора нужно загрузить число 50, затем прибавить к нему 2, используя дважды операцию инкремента. Программа на языке ассемблера и в машинных кодах будет иметь вид

Ассемблер	Машинные коды	Пояснение
LD BC, 50	1 50 0	Ввод числа 50 в регистр BC
INC BC	3	Увеличение числа на 1
INC BC	3	Увеличение числа на 1
RET	201	Возврат из подпрограммы

Таким образом, десятичные машинные коды этой подпрограммы — числа 1, 50, 0, 3, 3 и 201.

Программа DEMOMC (демонстрация машинных кодов) поясняет ввод кодов:

```
Ad=35520      Ar=49039
: DEMOMC mc 001 050 000 003 003
201 end ;
Compiling DEMOMC 49039 6 CDUMP ;
40,039      1
40,040      50
40,041      0
40,042      3
40,043      3
40,044      201
Stack 0      ---OK
52
```

После ввода слова DEMOMC заданы его исполнение и распечатка памяти. Из распечатки видно, как машинные коды распределяются по ячейкам ОЗУ.

Далее зададим команду b (возврат в Бейсик), а затем исполним ту же подпрограмму на Бейсике, задав слова

```
PRINT USR A
```

где $A=49039$ — начальный адрес подпрограммы в машинных кодах. Получим число 52, т. е. $50+2$. Таким образом, в данном случае подпрограмма, введенная на языке FSP88, исполнена затем на другом языке — Бейсике.

§ 2.14. Команды графики и синтеза звука

В версиях FP50 и FSP88 имеется ряд слов для задания цветных графических изображений. Прежде чем описать их, приведем некоторые определения, относящиеся к машинной графике.

Экран — люминесцентная поверхность электроно-лучевой трубки дисплея, на которой формируется изображение. При описании графики под ним имеется в виду полное изображение (растр), наблюдаемое пользователем (рис. 2.22).

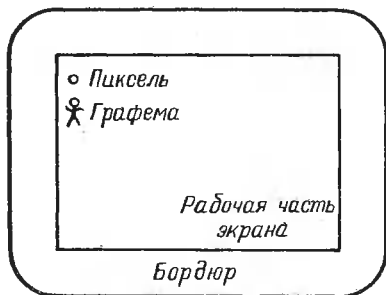


Рис. 2.22. Экран дисплея ПЭВМ

Рабочая часть экрана — ограниченная прямоугольной областью часть экрана. Ограничение используется для исключения краевых областей экрана сильными геометрическими искажениями.

Бордюр — окаймление рабочей части экрана, охватывающее его краевые участки.

Коды цвета — числовые коды, задающие цвет рабочей части экрана (страницы и знаков) бордюра.

Атрибуты графики — совокупность кодов координат цвета различных частей экрана. Атрибуты обычно также являются кодами, создаваемыми по особым правилам.

Пиксель — элементарная точка экрана, воспринимаемая как неделимый элемент изображения. Имеет координаты x и y , задаваемые в растровых единицах.

Графема — графический элемент, занимающий одно знакоместо. Графемами могут быть как знаки алфавита ПЭВМ, так и задаваемые пользователем знаки.

Знакоместо — область экрана, занимаемая одним знаком. Задается координатами знакоместа. Экран делится знакоместами на ряд строк (от 16 до 25) и столбцов (от 32 до 80). Следует отличать координаты точки от координат более крупного элемента — графемы.

Матрица знака — число пикселей по горизонтали и вертикали в одном знакоместе (обычно 8×8).

Разрешение графики — число пикселей, приходящихся по горизонтали и вертикали на рабочую часть экрана. Лежит в пределах от 256×176 для ПЭВМ среднего класса до примерно 600×400 для ПЭВМ высокого класса.

Основные слова машинной графики FSP88 даны ниже (c — коды цвета, x и y — координаты).

PAPER	c _ _ _ _	(Задаёт цвет страницы)
BORDER	c _ _ _ _	(Задаёт цвет бордюра)
INK	c _ _ _ _	(Задаёт цвет знаков)
BRIGHT	c _ _ _ _	(Задаёт яркость знака)
FLASH	c _ _ _ _	(Задаёт режим мерцания)
PROVER	c _ _ _ _	(Задаёт печать одного знака поверх другого знака)
AT	x y _ _ _ _	(Выводит знак на место (x, y))
ATTR	x y _ _ _ _ n	(Выдаёт атрибуты знака (x, y))
PLOT	x y _ _ _ _	(Строит точку (x, y))
POINT	x y _ _ _ _ n	(Выявляет наличие точки (x, y))
DRAW	Δx Δy _ _ _ _	(Строит отрезок прямой)
CIRCLE	x y r _ _ _ _	(Строит окружность с центром (x, y) и радиусом r)
CLS	_ _ _ _	(Очищает экран)

Для слов BRIGHT и FLASH код c равен 0 или 1. Код $c=0$ означает нормальную яркость и отсутствие мерцания знаков. Код $c=1$ даёт повышенную яркость и режим мерцания. При $c=0$ в слове PROVER вывод нового знака происходит на чистое знакоместо, а при $c=1$ — на старый знак без его уничтожения. В словах PLOT, POINT, DRAW и CIRCLE координаты x и y растровые (в пикселях), а для слов AT и ATTR они задаются в знакоместах (y — номер строки, x — номер столбца). За $x=0$ и $y=0$ растровых координат выбран левый нижний угол экрана.

При начальной загрузке FSP88 автоматически устанавливаются белый цвет страницы и бордюра, чёрный цвет знаков, отсутствие мерцания и нормальная яркость знаков. Для слов PAPER, BORDER и INK значения кодов c определяются конкретными реализациями ПЭВМ.

Следующая программа иллюстрирует задание точки словом PLOT и затем выявление ее наличия словом POINT:

```

Ad=3713E      Ar=59333
: DPOINT 120 50 PLOT 120 50 POIN
T . CR 121 50 POINT . ;
DPOINT ;
1
@

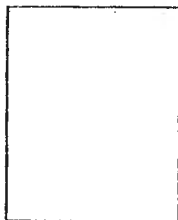
```

Слово 120 50 PLOT строит точку с координатами $x=120$ и $y=50$. Функция 120 50 POINT вырабатывает $n=1$, так как в этом месте

есть ранее построенная точка. Если ее нет — второй случай (121 50 POINT), выработывается значение $n=0$.

Следующая программа

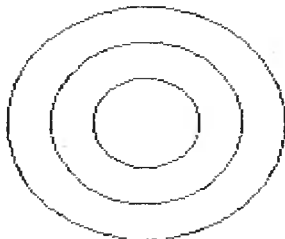
```
Ad=37068      Ar=59078
: BOX 125 10 PLOT 0 120 DRAW 60
0 DRAW 0 -120 DRAW -60 0 DRAW ;
BOX ;
```



иллюстрирует применение слов PLOT и DRAW для построения прямоугольника.

Наконец, третья программа

```
Ad=37128      Ar=59251
: DCIRC 150 70 25 CIRCLE 150 70
45 CIRCLE 150 70 65 CIRCLE ;
DCIRC ;
```



строит три concentric окружности с помощью слова CIRCLE. Изображение имеет вид окружностей на экране дисплея. Распечатка принтером вносит некоторые геометрические искажения — изображение несколько растягивается по оси x и окружности выглядят эллипсами.

Действие других слов легко проверить, наблюдая за изображением на экране дисплея. Графические возможности FSP88 достаточны для построения сложных графиков и создания машинных фильмов (в последнем случае широко используются графемы и фрагменты программ в машинных кодах). Как отмечалось, задание графем на языке FSP88 обеспечивается редактором (см. § 2.2).

Синтез звуков, как и на языке Бейсик, обеспечивается словом

BEEP d_1 d_2 — — —

где число d_1 задает длительность звукового сигнала (в 1/50 с), а d_2 — высоту тона (период колебаний). Звуковые колебания имеют

форму прямоугольных импульсов. Во время d_1 - звучания другие слова не выполняются.

FSP88 имеет также дополнительное слово для синтеза звука
BLEEP $d_1 d_2$ — — —

Здесь d_1 задает число периодов звукового колебания с периодом (в микросекундах).

Слова BEEP и BLEEP задают одноголосное звучание. На время генерации звука другие действия не выполняются. Так, выполнение программы

: DBLEEP 5000 1000 BLEEP . "КОНЕЦ" ;

показывает, что слово КОНЕЦ появляется только после окончания пятисекундного интервала звучания тона с частотой 1000 Гц (период 1000 мкс).

РАСШИРЕНИЕ ФОРТ-СИСТЕМ С ОПЕРАЦИЯМИ НАД ЧИСЛАМИ С ПЛАВАЮЩЕЙ ТОЧКОЙ

§ 3.1. Константы и расширение графики

Объем ОЗУ, занимаемый программами на языке Форт, во многом зависит от рациональной системы задания слов, исключающей по возможности повторы операций. В этой связи желательно расширить базовый набор операций (слов) некоторыми новыми словами. В частности, это относится к вызову широко распространенных констант, вычислению часто встречающихся арифметических выражений и т. д. Реализация таких операций должна предусматривать наиболее быстрые методы их проведения. В целом полная система FSP88 получена вводом всех слов, описанных в данной книге.

Начнем с операции вывода констант. Константа в мнемоническом виде — просто число. Зададим слово CONST, выводящее константу 1, и выполним его декомпиляцию:

```
Ad=3E304   Ar=5E516
: CONST 1 ;
  CONST DC ;
number 0 0 1 0 0 ret
  CONST WC ;
205,11,172,0,0,1,0,0,201,
БАЙТ=9
```

Результат декомпиляции поразителен! Оказывается, что единица, занимающая в мнемоническом виде место всего одного символа, после компиляции занимает целых 8 байт. Из них три — обращение к слову number, а пять — сама константа, довольно бездарно разложенная по байтам. Любое число в FSP88, будучи константой, отнимает в ОЗУ 8 байт (будь то 0, 1, 10 и т. д.). Итак, константа — весьма неприятное исключение из хваленого правила Форты — длина скомпилированной словарной статьи меньше, чем мнемонической. Еще одно неприятное обстоятельство — числовая константа нарушает естественный мнемонический листинг декомпилируемой статьи. А если один из ее байтов имеет значение 201 или 205 (что очень маловероятно), может нарушиться работа декомпиляторов.

Таким образом, весьма желательно задать наиболее ходовые константы всего один раз. Тогда их вызов будет занимать всего 3 байт: один — код 205 (call), а два других — адрес A_r. Для обозначения кон-

стант будем использовать букву С (константы, применяемые часто) и букву с (константы, применяемые редко). Даже если константа используется дважды, ее выгодно задать таким образом.

Лист 3.1

Простые константы

0 : C0 0 ;
1 : C1 1 ;
2 : C2 2 ;
3 : c3 3 ;
4 : c4 4 ;
5 : C5 5 ;
6 : c6 6 ;
7 : c8 8 ;
8 : c10 10 ;
9 : c12 12 ;
10 : c24 24 ;
11 : C90 90 ;
12 : C180 180 ;
13 : c201 201 ;
14 : c205 205 ;
15 : C.5 0.5 ;

На листе 3.1 представлено задание 16 простых констант, часто используемых в последующих словарных статьях.

Лист 3.2

Адресные и специальные константы

0 : c33798 33798 ;
1 : c38496 38496 ;
2 : c38498 38498 ;
3 : c38500 38500 ;
4 : c38550 38550 ;
5 : 2PI 6.2831853 ;
6 : PI/2 1.5707963 ;
7 : LN10 2.3025851 ;
8 : 180/PI 57.29578 ;
9 : EPS 1E-8 ;
10 : CE 0.57721567 ;
11 : c5/9 0.55555556 ;
12 : c8/9 0.88888889 ;
13 : SQR.6 0.77459667 ;
14 : Arg 23627 % @ DU 9 %+ SW 15 %+ ;
15 : %5* C5 %* ;

На этом листе задан ряд адресных и специальных констант. В число последних входят константы 2PI, PI/2 (где PI есть число π), LN10, 180/PI, EPS ($\epsilon=1 \cdot 10^{-8}$), постоянная Эйлера CE, константы c5/9, c8/9 и SQR.6 ($\sqrt{0,6}$), используемые в операциях численного интегрирования методом Гаусса.

Слово

Argd — — — A_d A_r

оставляет в стеке адреса переменных d и r текстового интерпретатора, обеспечивая их доступность (с помощью слов %! и % @).

Слово

$\%5*n$ — — — $5\cdot n$

обеспечивает целочисленное умножение (оно не относится к константам, а просто дополняет лист).

Лист 3.3

Цветовые константы и расширения системы

```
0 : BLACK C0 ;
1 : BLUE C1 ;
2 : RED C2 ;
3 : MGNTA c3 ;
4 : GREEN c4 ;
5 : GYAN C5 ;
6 : YELLOW c6 ;
7 : WHITE 7 ;
8 : XPLOT 23677 C@ ;
9 : YPLOT 23678 C@ ;
10 : 2DU DU DU ;
11 : 2ROT ROT ROT ;
12 : CPLOT INK PLOT ;
13 : DLINE INK YPLOT - SW XPLOT - SW DRAW ;
14 : LINE DU INK 2ROT PLOT DLINE ;
15 : FRAMES 23672 C@ 23673 C@ 23674 C@ ;
```

На листе 3.3 в строках 0-7 задана выдача кодов для восьми цветов графики: BLACK (черный), BLUE (синий), RED (красный), MGNTA (сокращение от *magenta* — оранжевый), GREEN (зеленый), GYAN (голубой), YELLOW (желтый) и WHITE (белый). Использование этих констант делает более понятными программы цветной графики.

Следующие два слова: 2DU (двойное DU) и 2ROT (двойное ROT) — расширяют набор операций со стеком. Слова XPLOT и YPLOT выдают координаты x и y последней построенной точки.

Слово

CPLOT x y c — — —

строит точку с координатами (x, y) и цветом, заданным кодом c . Вместо c можно задавать цветовую константу. Например,

100 50 RED CPLOT ;

строит красную точку с координатами (100; 50).

Слово DRAW, как отмечалось, строит отрезок от заданной точки (x, y) до точки $(x + \Delta x, y + \Delta y)$.

Слово

DLINE x y c — — —

(от слов *draw line*) строит отрезок прямой, соединяющий ранее построенную точку (ее координаты хранятся в ячейках ОЗУ с адресами 23677 и 23678 и выдаются словами XPLOT и YPLOT) с новой точкой (x, y) . Цвет отрезка задается кодом c . Слово DLINE очень удобно

для построения ломанных линий с точной стыковкой сопрягаемых отрезков (из-за погрешностей вычисления $(x + \Delta x)$ и $(y + \Delta y)$) слово DRAW не дает точной стыковки).

Слово

LINE x_2 y_2 x_1 y_1 c _ _ _

строит отрезок прямой линии, проходящей через точки с координатами (x_1, y_1) и (x_2, y_2) и имеющей цвет с кодом c . После построения последней точка имеет координаты (x_2, y_2) .

Слово

FRAMES _ _ _ $n1$ $n2$ $n3$

служит для создания «электронных часов». Оно вырабатывает три непрерывно меняющихся со значениями 0, 1, 2, ..., 255 числа $n1$, $n2$ и $n3$. Число $n1$ меняется на 1 с интервалом $1/50$ с, число $n2$ — реже в 256 раз, а $n3$ — реже в 256^2 раза.

§ 3.2. Системные функции, включая реализацию монитора и декомпиляторов

На трех следующих листах представлено системное расширение версии FSP88. Можно убедиться в том, сколь просто FSP88 реализует специальные системные функции.

Лист 3.4

Некоторые системные функции

```
0 : c96 96 ;
1 : c123 123 ;
2 : c38495 38495 ;
3 : n c38498 c33798 ;
4 : ARD n %I DU %I %@ % = IF %I c6 %- %LV ELSE THEN c
8 %+J SW DR ;
5 : DW ARD c38496 %! ;
6 : cw DU c6 TYPE ."СКРЫТО" CR 2DU c8 %+ 2ROT c38495
SW %- CMOVE ;
7 : CW ARD cw ;
8 : cws c96 > A @ C@ c123 < AND IF A @ cw ELSE c8 A
+! THEN ;
9 : wc 2DU c205 + SW ;
10 : wce SW - 1+ CR ."БАЙТ=" . ;
11 : CWS ARD A ! BEGIN A @ C@ cws A @ c38495 > UNTIL
;
12 : WC wc [ I DU C@ DU . ." ," c201 = IF LV ELSE DR
THEN ] wce ;
13 : 2^ DU * ;
14 : 3^ 2DU * * ;
15 : %2* DU %+ ;
```

Слова c96, c123 и c38495 задают константы. Слово

n _ _ _ 38498 33798 .

задает заголовок цикла и определяет адреса поиска A_d по значениям A_r (см. ниже).

Слово

ARD A_r — — — A_d

преобразует адрес A_r в адрес A_d . Для этого в цикле просматривается адресное поле заголовков словаря в области d . Если адрес A_r обнаружен, то шестая ниже его ячейка имеет адрес A_d .

Слово

DW A_r — — — ($A_d \rightarrow 38496$)

заносят в служебную ячейку ОЗУ с адресом 38496 значение A_d для слова с адресом компиляции A_r . Значение A_d используется текстовым интерпретатором при реализации замены имени слова.

Слово CW служит для скрытия слова в словаре. Оно использует вспомогательное слово sw. По указанному A_r определяется адрес A_d . Затем вся расположенная выше область заголовков словом CMOVE смещается вниз на 8 байт, исключая тем самым слово из этой области.

Слово CWS (и вспомогательные слова cws, wc) выполняет в цикле поиск имен всех слов, первый символ имени которых имеет код от 97 до 122 (т. е. строчная латинская буква). Если такое слово обнаружено, CWS обращается к слову CW и скрывает слово в словаре. Иначе поиск идет дальше — до конца области заголовков.

Слово

WC A_r — — — (печать латинских кодов)

реализует функции декомпилятора машинных кодов. Оно по значению A_r выводит машинные коды, хранимые в ячейках A_r , A_{r+1} ..., и проверяет их на равенство константе 201. Если равенство обнаружено, цикл прерывается и на печать выводится подсчитанное число байтов словарной статьи.

Три слова

$2 \uparrow d$ — — — d^2

$3 \uparrow d$ — — — d^3

$\%2 * n$ — — — $2 \cdot n$

дополняют лист 3.4 и реализуют дополнительные арифметические операции.

Лист 3.4а

Дополнительные системные функции

```
Ø : SPR DU 44Ø24 %! ARD
```

```
44Ø2Ø %! ;
```

```
1 : .S DPT DU CØ %<= IF DR ELSE CØ %I . SP %J THEN ;
```

```
2 : ?S DPT DU CØ %<= IF DR ELSE %I+ C1 %I %I PICK .  
SP %J THEN ;
```

3 : WA ARD SW ARD %I %I DU c6 TYPE SP . SP %I c6 %+
 X? CR c8 %+] ;
 4 : %N! N %! ;
 5 : %N@ N %@ ;
 6 : X! X ! ;
 7 : X@ X @ ;
 8 : Y! Y ! ;
 9 : Y@ Y @ ;
 10 : Z! Z ! ;
 11 : Z@ Z @ ;
 12 : 2* DU + ;
 13 : 2/ c.5 * ;
 14 : PI 3.1415926 ;
 15 : LG LN LN10 / ;

Слово (от *save programm*)

SPR A'_r — — —

заносит в ячейки ОЗУ 44020, 44022, 44024 и 44026 значения A_d , A'_d , A_r и A'_r для текущего состояния системы и при записи ее части, начиная с указанного слова (см. описание команды *sp*).

Слово

S $N_1 N_2 \dots N_n$ — — —

выводит на печать в строку содержимое стека, разделяя выведенные числа пробелом. Стек при этом очищается.

Действие слова

?S $N_1 N_2 \dots N_n$ — — — $N_1 N_2 \dots N_n$

аналогично описанному для слова *.S*, но числа в стеке сохраняются.

Анализ словарных статей слов *.S* и *?S* хорошо иллюстрирует полезность циклов. Вначале слово *DPT* оставляет на вершине стека количество чисел n . Далее организуется цикл вывода, если $n > 0$. При $n \leq 0$ такой цикл не проводится. Специфика FSP88 (и FP50) заключается в том, что n может быть отрицательным, например при применении команды *.* при пустом стеке. Поэтому и необходим анализ n .

Слово

WA $A_{r1} A_{r2}$ — — —

печатает адреса A_d и A_r и имена всех слов от слова с адресом A_{r1} до слова с адресом A_{r2} . Пример его применения приводился ранее.

Слова в строках 4—11 очевидны и сокращают слова присвоения и вызова числовых значений переменных. Остальная группа слов пополняет набор арифметических операций:

$2* d \dots 2d$
 $2/ d \dots d/2$
 PI — — — π
 LG $d \dots \lg d = \ln d / \ln 10$.

Обратите внимание, что в слове 2* операция умножения $2d$ заменена операцией сложения, т. е. $2d = (d + d)$, а в слове 2/ операция деления заменена умножением на константу 0,5. Это уменьшает время выполнения операций.

Лист 3.5

Монитор и символьный декомпилятор

```

0 : c26 26 ;
1 : c31 31 ;
2 : c23 23 ;
3 : c165 165 ;
4 : mon0 c23 c6 %[ Z@ C@ %I TAB . C1 Z +! c4 %+J c26
  TAB ;
5 : mon1 Z@ C@ 2DU c165 < SW c31 > AND ;
6 : mon C5 C0 %[ mon1 IF EMIT ELSE DR SP THEN C1 Z +
  ! %J ;
7 : MON SW DU ROT + 1+ SW %[ %I Z! %I . mon0 %I Z! m
  on CR C5 %+J ;
8 : dcn ARD c6 -TRAIL TYPE SP ;
9 : dcf c6 %- dcn C1 X! C0 ;
10 : dcw C0 X! n %[ Y@ %I %@ %= IF %I dcf %LV ELSE T
  HEN c8 %+J ;
11 : dc0 X@ 0= IF ."call " Z@ 1+ %@ . SP ELSE THEN ;
12 : dc1 Z@ C@ . SP C1 Z +! ;
13 : dc Z@ C@ c205 = IF Z@ 1+ %@ Y! dcw dc0 c3 Z +!
  ELSE dc1 THEN ;
14 : DC Z! C180 C0 %[ Z@ C@ c201 = IF ."ret" %LV ELS
  E dc THEN %J ;
15

```

Можно лишь поражаться, что две достаточно сложные программы — монитор с форматированным выводом и символьный декомпилятор — помещаются в пределах лишь одного листа системы FSP88. Это говорит о поразительной компактности программы даже в мнемоническом виде.

Слово

MON A n — — —

организует циклический просмотр содержимого n ячеек памяти, начиная с A . Содержимое выводится на форматированную печать в виде адресов A , $A+5$, $A+10$ и т. д. и строк по пять чисел (значений байтов). Затем эти числа анализируются на выполнение неравенств < 31 и > 165 . Если одно из неравенств выполняется, в строке символов печатается пробел, иначе — символ по ASCII. Таким образом, выдача задана в стандартном для мониторов виде.

Функции символьного декомпилятора подробно описывались в § 2.2. Читателю можно порекомендовать детально проанализировать строки 8—14 листа 3.5, реализующие программу декомпиляции.

§ 3.3. Табуляция произвольных функций одной переменной

Нормально форт-система FSP88 поддерживает идеологию программирования снизу вверх. Другими словами, прежде чем реализовать какое-то либо новое слово, нужно иметь заданными все входящие в состав его словарной статьи слова. Это напоминает постройку дачи после того, как подобраны все необходимые материалы.

Мы знаем, что есть и иной путь — вначале создать проект дачи, а затем подбирать к нему нужные материалы. Именно так реализуется стиль программирования сверху вниз. При нем можно создавать главную программу, а потом подключить к ней вспомогательные программы. Некоторые версии Форты (например, FORTH-PC или близкая по духу система ДССП-80 [5, 9]) поддерживают и такой стиль.

Если дачу можно построить из попавшихся под руки материалов, то завод так всерьез не построишь. И действительно, чем сложнее программа, тем явнее выявляются преимущества стиля программирования (или скорее, мышления) сверху вниз.

Однако в этом мы можем убедиться и на гораздо более простых примерах. Допустим мы хотим создать слово FTAB, выполняющее табуляцию любой функции $f(x)$ аргумента x . Но не задав вычисление $f(x)$, мы не сможем приступить к созданию и словарной статьи для слова FTAB. А составив слова для вычисления $f(x)$ и FTAB, мы затем не сможем использовать FTAB для других функций. Получается замкнутый круг?

Система FSP88 довольно легко обходит такую трудность. В трех первых строках листа 3.6 содержатся слова FA!, FA@ и F@ для реализации структурного программирования сверху вниз.

Лист 3.6

Табуляция и вычисление некоторых функций и их производных

```
0 : FA! c38496 %! ;
1 : FA@ c38496 %@ ;
2 : F@ FA@ EXECUT ;
3 : FTAB %1+ C1 %I DU ROT DU F@ SW DU . SW c12 TAB .
  CR + SW %J ;
4 : F(X) 2^ NEG EXP 1.1283792 * ;
5 : SINX/X DU SIN SW / ;
6 : f' Z! Y! Z@ Y@ 2* - F@ Z@ Y@ - F@ c8 * - ;
7 : F' f' Z@ Y@ + F@ c8 * + Z@ Y@ 2* + F@ - c12 / Y@
  / ;
8 : HSN EXP DU C1 SW / - 2/ ;
9 : HCS EXP DU C1 SW / + 2/ ;
10 : HTN HSN 2DU * 1+ SQR / ;
11 : AHS 2DU * 1+ SQR + LN ;
12 : AHC 2DU * 1- SQR + LN ;
13 : AHT DU 1+ SW C1 SW - / SQR LN ;
14 : N! 1+ C1 C1 X! %I %I X@ * X! %J X@ ;
15 : POL X! C0 DPT C2 SW [ I PICK + X@ * C1 -] C2 PI
  CK + ;
```


Слово FA! используется в конструкции

'Имя FA! ;

Оно заносит в системную ячейку ОЗУ с номером 38496 адрес A_r любого слова (например, определяемой в дальнейшем функции).

Слово

FA@ _ _ _ A_r

оставляет на вершине стека содержимое ячейки 38496.

Наконец, главное слово

F@ _ _ _ $f(x)$

оставляет на вершине стека вычисленное значение $f(x)$ для функции, заданной с помощью слова FA!. В слове F@ использована конструкция A_r EXECUT, выполняющая словарную статью с адресом компиляции A_r .

Теперь нет препятствий к применению слова F@ в любой программе, где необходимо вычислить какую-либо функцию. Просто надо не забыть перед выполнением вычислений занести в ячейку 38496 (используя слово FA!) адрес компиляции A_r нужной нам функции. Забывчивость в определении функции тут непростительна — наступает самое неожиданное поведение системы.

В строке 3 листа 3.6 задано слово FTAB, используемое в виде

$a \Delta x n$ FTAB ;

Здесь a — нижняя граница аргумента x , Δx — шаг изменения x и n — число строк таблицы.

В следующем примере показаны задание внешней функции $F(x) = e^{-x^2/\sqrt{2\pi}}$ для нормального распределения вероятности и результат ее табуляции:

```

Ad=37120   Ar=59221
: F(x) 2^-2 / EXP PI 2* SQR / ;
' F(x) FA! 0 0.25 2 FTAB ;
0           0.39894228
0.25       0.38666812

3 FTAB ;
0.5         0.35206533
0.75       0.30113743
1           0.24197072

```

Слово FTAB составлено таким образом, что в его словарной статье не применяется ни одна из глобальных переменных форт-системы. Для хранения операндов используется только стек. При каждом обращении к слову FTAB печатается n строк таблицы значений x и $f(x)$, причем на вершине стека остаются значения $b = a + n\Delta x$ и Δx . Поэтому, задав новое n , можно продолжить печать таблицы, начиная с нового $a \leftarrow b$.

§ 3.4. Вычисление производной функции одной переменной

Вычисляемые с помощью слова F@ функции могут быть самыми разнообразными. Для примера в строках 4 и 5 заданы два слова, используемые в дальнейшем:

Слово

$$P(X) \ x \ _ \ _ \ _ \ 2e^{-x^2}/\sqrt{\pi}$$

Слово

$$\text{SINX/X} \ x \ _ \ _ \ _ \ (\sin x)/x$$

Над функциями могут проводиться различные операции. Часть из них описана в гл. 5. Остановимся на одной — операции численного дифференцирования (вычисления первой производной), ибо она также сводится к простому вычислению по формулам.

Численное дифференцирование функций, заданных таблично, используется редко. Поэтому включать в словарь эту операцию нецелесообразно (в [10, 11] подробно описаны алгоритмы численного дифференцирования табличных данных, которые при необходимости несложно реализовать средствами языка Форт).

Более полезна операция численного дифференцирования функций $F(x)$, заданных аналитически. Она исключает необходимость поиска в справочниках значений производной $F'(x)$ и организацию дополнительных вычислений $F'(x)$ по самым разнообразным формулам. Результаты вычисления значения производной $F'(x)$ для $F(x)$ в заданной точке $x=x_0$ в случае гладких функций удается получить с высокой точностью (6—7 верных знаков), что обычно вполне достаточно для практики. Необходимость оперативного вычисления $F'(x)$ возникает часто, например для построения графика производной $F'(x)$ или оценки чувствительности $F(x)$ к изменению x .

Для численного дифференцирования обычно используются формулы для 3, 5 и 7 узлов с центральным узлом при $x=x_0$. Формулы для 3 узлов обычно дают невысокую точность из-за грубой аппроксимации $F(x)$, а формулы для 7 узлов слишком сложны. Точность последних не намного выше, чем при 5 узлах, из-за возрастания погрешностей вычисления $F(x)$ при близко расположенных узлах.

В связи с этим для численного дифференцирования ограничимся применением формулы для 5 узлов, расположенных через интервал $\Delta x=h$:

$$F'(x) = (F_{-2} - 8F_{-1} + 8F_{+1} - F_{+2})/12h. \quad (3.1)$$

Здесь значения $F(x)$ вычисляются при $x_{-2}=x_0-2h$, $x_{-1}=x_0-h$, $x_{+1}=x_0+h$ и $x_{+2}=x_0+2h$. Они обозначены условно соответственно как F_{-2} , F_{-1} , F_{+1} и F_{+2} . Таким образом, численное дифференцирование

сводится к заданию указанных выше текущих значений x и вычислению F_{-2}, F_{-1}, F_{+1} и F_{+2} и $F'(x)$ по формуле (3.1). Эти вычисления реализует слово F' (см. листинг листа 3.6, строки 6 и 7).

Ниже приведен пример вычисления производной для функции $F(x)$, использованной при проверке слова FTAB:

```
Ad=37120   Ar=59221
: F(x) 2^ C2 NEG / EXP PI 2* SQR
/ ;
' F(x) FA! .05 0.5 F . ;
-0.1760322
.001 0.5 F' . ;
-0.1760326
```

Отметим, что точное значение $F'(x) = -0,17603266$ и при $h=0,01$ верны семь знаков результата.

Результаты численного дифференцирования зависят от величины шага h . При больших h узлы расположены друг от друга далеко, что снижает точность аппроксимации (формулы дифференцирования получены аппроксимацией $F(x)$ полиномом). Однако при малом h значений ординат $F(x)$ мало отличаются друг от друга, что ведет к росту погрешности вычисления $F'(x)$ по формуле (3.1). Это демонстрирует второй пример — вычисление $F'(x)$ при $h=0,001$ (последняя цифра результата 2 отличается от точной 6). Рационально выбирать $h \approx 0,01$ для большинства достаточно гладких функций.

§ 3.5. Вычисление гиперболических и обратных гиперболических функций

В научных и технических расчетах широкое применение находят гиперболические функции действительного переменного x . Их можно представить в виде экспоненциальных функций, имеющих в составе базовых версий языка FSP88 с плавающей точкой:

$$\begin{aligned} \operatorname{sh} x &= (\exp x - 1/\exp x)/2, \\ \operatorname{ch} x &= (\exp x + 1/\exp x)/2, \\ \operatorname{th} x &= \operatorname{sh} x / \sqrt{1 + \operatorname{sh}^2 x}. \end{aligned}$$

Эти функции определены для любых x (в пределах разрядной сетки ПЭВМ) и реализованы словами HSN, HCS и HTN (см. лист 3.6, строки 9, 10 и 11):

```
HSN x _ _ _ sh x
HCS x _ _ _ ch x
HTN x _ _ _ th x
```

Обратные гиперболические функции также имеют простые аналитические представления:

$$\operatorname{arsh} x = \ln(x + \sqrt{1 + x^2}) \text{ при любом } x,$$

$$\operatorname{arsh} x = \ln(x + \sqrt{x^2 - 1}) \text{ при } x > 1.$$

$$\operatorname{arth} x = \ln \sqrt{(x+1)/(x-1)} \text{ при } -1 < x < 1.$$

Эти функции реализованы словами (строки 11, 12 и 13 листа 3.6):

AHS x — — — $\operatorname{arsh} x$

AHC x — — — $\operatorname{arch} x$

AHT x — — — $\operatorname{arth} x$

Примеры на вычисление гиперболических и обратных гиперболических функций:

```
1 HSN . ;
1.1752012
1 HCS . ;
1.5430806
1 HTN . ;
0.76159416
1 AHS . ;
0.88137359
2 AHC . ;
1.3169579
0.5 AHT . ;
0.54930614
```

Следует отметить, что описанные выше функции могут вычисляться самостоятельно, например по разложениям в ряд. Однако для уменьшения объема ОЗУ целесообразно вычислять их через базовые функции (слова), хотя в отдельных случаях это ведет к несколько большей погрешности.

§ 3.6. Вычисление значений факториала и полинома

Факториал числа N по определению есть

$$N! = 1, \quad \text{если } N=0 \text{ или } N=1,$$

$$N! = 1 \cdot 2 \cdot \dots \cdot N, \quad \text{если } N \geq 1.$$

Этот естественный алгоритм вычисления факториала удобно реализовать с помощью целочисленного цикла `%DO ... %LOOP` (или `% [...%]` в FSP88) умножением x с начальным значением $x=1$ на значение управляющей переменной цикла `%I`. При этом следует задать $n_n=1$ и $n_x=N+1$. Это реализует слово $N!$ в строке 14 листа 3.6.

При этом алгоритме максимальное значение $N!$ ограничено разрядной сеткой ПЭВМ, т. е. $N! < X_{\max}$. Так, для $X_{\max}=1 \cdot 10^{38}$ имеем $N < 34$.

Описанный здесь прием — применение быстрого целочисленного цикла — обеспечивает ускорение вычислений. Поэтому он целесообразен и при реализации других операций, например, с векторами и матрицами (гл. 4).

Еще одним полезным примером применения цикла служит вычисление значений степенного полинома

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_2 x^2 + a_1 x + a_0.$$

Поскольку вычисление выражений x^i является медленной операцией, вычисление $P(x)$ удобно производить по схеме Горнера

$$P(x) = (\dots (a_n x + a_{n-1}) x + \dots + a_1) x + a_0.$$

При этом медленные операции возведения в степень заменяются существенно более быстрыми операциями умножения. Более того, эти операции имеют явно циклический характер: выражение в круглых скобках умножается на x , к нему прибавляется очередной коэффициент полинома и т. д.

Однако реализация такого циклического алгоритма при различных x предполагает, что все коэффициенты полинома должны сохраняться в памяти ЭВМ. Они образуют вектор — одномерный массив из $n+1$ чисел. Способ организации хранения вектора в ОЗУ ПЭВМ будет рассмотрен далее (см. § 4.2). Пока же отметим следующее.

Естественным хранилищем вектора может служить арифметический стек ПЭВМ, присущий форт-системам. В системе FSP88 в стек можно вводить порядка 100 чисел. Это более чем достаточно при практических задачах с полиномами, так как на практике степень полинома редко превышает значения $n \geq 10$. Количество чисел в стеке может доходить и до 300, если исключить лист редактора (для этого при исполнении слова `cl` в ответ на запрос о номере строки нужно указать число `-16`).

Таким образом, вычисления полинома можно организовать так, что коэффициенты полинома вводятся в стек и хранятся в нем. В приведенном ниже листинге слова `POL` эти коэффициенты вводятся первый раз, затем вводится значение x . При повторных вычислениях вводится только новое значение x , а коэффициенты полинома сохраняются.

Вычисление $P(x)$ реализуется словом `POL` (строка 15 листа 3.6). Рассмотрим листинг этого слова более подробно.

Ввод чисел в стек перед пуском соответствует приведенному ниже:

$$a_n \ a_{n-1} \ \dots \ a_1 \ a_0 \ x \ _ \ _ \ _$$

Слова `X!` `S0` удаляют число x из вершины стека, присваивают его переменной `X` и помещают в стек число 0. Далее слова `DPT` `S2` помещают в стек числа $n+2$ (это количество чисел в стеке — число 0 и $n+1$ коэффициентов полинома) и 2. Затем команда `SW` формирует окончательное распределение чисел вида

$$a_n \ a_{n-1} \ \dots \ a_1 \ a_0 \ 0 \ 2 \ n+2$$

Два последних числа образуют заголовок цикла, необходимый для реализации схемы Горнера. Число 0 — начальное значение $P(x)$.

Далее идет сам цикл (выражение в скобках [...]). Значение управляющей переменной I этого цикла задает вызов словом `PICK` соответствующего коэффициента и суммирование его с текущим числом на вершине стека (это 0 в первый раз, после выполнения слова `]`). Затем это число умножается на значение переменной X (т. е. на x), после чего цикл повторяется. При выходе из цикла (он идет с шагом изменения $I \delta = -1$) слова `2 PICK +` прибавляют к содержимому стека значение a_0 . В итоге имеем распределение чисел в виде

$$- \dots - a_n a_{n-1} \dots a_1 a_0 P(x)$$

Если вывести из стека значение $P(x)$ и ввести новое значение x , получим исходное распределение чисел в стеке с сохранением всех значений a_i полинома.

Работа со словом `POL` требует некоторой осторожности — надо следить, чтобы в стек не попало случайно какое-либо число, так как это приведет к его интерпретации как коэффициента полинома с последующим неверным вычислением $P(x)$. Поэтому более предпочтителен метод вычисления $P(x)$ с хранением a_i в ОЗУ (§ 4.2).

Ниже показаны контрольные примеры на вычисление факториала $N!$ и значений полинома `POL`:

```

0 N! . ;
1
10 N! . ;
3628800
4 3 2 1 0 1 POL . ;
10
2 POL . ;
98

```

ОПЕРАЦИИ С ВЕКТОРАМИ И МАТРИЦАМИ

§ 4.1. Ввод и вывод векторов

Вектором называется последовательность чисел — компонент вектора:

$$v_1 \ v_2 \ v_3 \ \dots \ v_n$$

Количество компонент n характеризует длину вектора. Векторами могут представляться данные эксперимента, колонки и строки таблиц, координаты x и y графиков и т. д. Векторы являются одномерными массивами.

В системе FSP88 легко задаются векторы следующих трех типов.

1. Вектор целых положительных чисел единичной разрядности с десятичным представлением от 0 до 255. Каждое число занимает в ОЗУ 1 байт, т. е. одну ячейку.

2. Вектор целых положительных чисел двойной (с позиций FSP88, но не FORTH-79) разрядности с десятичным представлением от 0 до +65535. Такие числа занимают в ОЗУ 2 байта, т. е. вдвоенную ячейку.

3. Вектор десятичных положительных и отрицательных чисел с плавающей запятой (восемь знаков нормализованной мантиссы и два знака порядка), лежащих в указанных выше (§ 2.1) пределах. Каждое число занимает в ОЗУ 5 байт, или спятеренную ячейку.

Физически вектор в системе FSP88 есть последовательность ячеек памяти ОЗУ, представленная (для трех типов векторов) на рис. 4.1. Во всех случаях вектор характеризуется начальным адресом A и длиной занимаемой в ОЗУ области. Она равна n байтам для вектора типа 1, $2n$ байтам для векторов типа 2 и $5n$ байтам для векторов типа 3.

Задание векторов и манипуляция с их адресами поддерживаются некоторыми системными функциями (базовыми словами) FSP88. Как отмечалось выше (см. § 2.6), имеются команды размещения целых чисел одинарной и двойной разрядности в ОЗУ по заданному A , изменения адреса и контроля содержимого памяти (слова DUMP и % DUMP). В связи с этим рассмотрим подробно дополнительные процедуры задания и вывода векторов десятичных чисел с плавающей точ-

кой в форме, удобной для расчетов в режиме диалога. Они приведены в строках 0—7 листа 4.1. Слова, относящиеся к векторным операциям, отмечены буквой V.

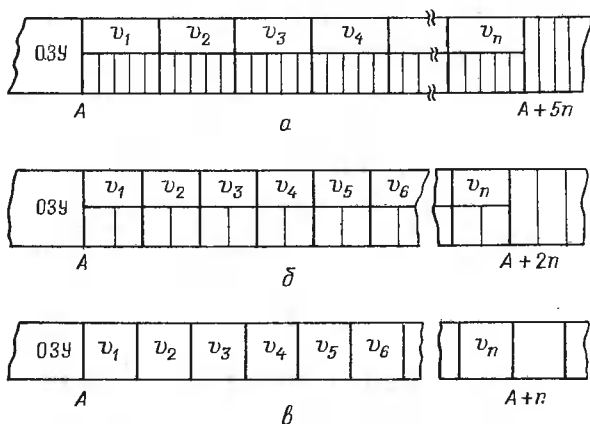


Рис. 4.1. Представление векторов в ячейках ОЗУ FSP88: целые числа от 0 до 255 (а), целые числа — от 0 до 65535 (б) и числа с плавающей точкой (в)

Лист 4.1

Операции с векторами

- 0 : VA0 c38500 c38498 %! ;
- 1 : a0V c38498 %@ ;
- 2 : VN DPT %N! ;
- 3 : VI VN %N@ C0 %I %N@ %I %- ROLL a0V ! C5 c38498 % +! %I ;
- 4 : VIS VA0 VI ;
- 5 : VIF VI VA0 ;
- 6 : VD %N@ C0 %I ."V" %I %I+ . ."=" a0V %I %5* %+ ? CR %I ;
- 7 : V@ %I- %5* a0V %+ @ ;
- 8 : cn1 %N@ %I+ C1 ;
- 9 : VSUM C0 cn1 %I %I V@ + %I ;
- 10 : VPROD C1 cn1 %I %I V@ * %I ;
- 11 : VMAX C1 V@ cn1 %I %I V@ MAX %I ;
- 12 : VAMAX C0 cn1 %I %I V@ ABS MAX %I ;
- 13 : VMIN C1 V@ cn1 %I %I V@ MIN %I ;
- 14 : NV VAMAX cn1 %I DU %I V@ SW / %I %I- %5* a0V %+ ! %I ;
- 15 : VFOL X! C0 C1 %N@ %I V@ + %I * C1 -%I C1 V@ + ;

Ниже перечислены слова ввода-вывода векторов с указанием номеров строк листа 4.1.

0. Слово

VA0 — — —

задает начальный адрес векторов $A_0=38500$ путем занесения его в

две ячейки ОЗУ, начиная с ячейки с адресом 38498. Таким образом, массив с компонентами векторов помещается в ОЗУ между областями d и g .

1. Слово

VN $v_1 v_2 \dots v_n$ — — —

задает переменной N целочисленное значение количества компонент вектора n , введенного в стек.

2. Слово

a0V — — — A_0

оставляет на вершине стека начальный адрес вектора A_0 (содержимое двух ячеек ОЗУ с адресами 38498 и 38499).

3. Слово

VI $\bar{a}_1 \dots n_n$ — — —

фиксирует ввод вектора (от слов *vector input* — ввод вектора), переписывая содержимое n ячеек стека в ОЗУ, начиная с адреса A_0 , вычисляет n , присваивает значение n переменной N , очищает стек и увеличивает A_0 на $5n$.

4. Слово

VIS $a_1 \dots a_n$ — — —

фиксирует ввод первого вектора, задавая вначале $A_0=38500$ (в остальном действует подобно слову VI). Наименование слова образовано первыми буквами слов *vector input start*.

5. Слово

VIF $a_1 \dots a_n$ — — —

фиксирует ввод последнего вектора (как VI) и после этого задает значение $A_0=38500$. Образовано от слов *vector input finish*.

6. Слово

VO — — — (вывод вектора)

выводит на индикацию все компоненты вектора (от слов *vector output* — вывод вектора). Каждая компонента выводится в виде

$V_i = \text{Число}$

где i — индекс вектора (число от 1 до n).

7. Слово

V@ i — — — v_i

заменяет индекс i на вершине стека значением компоненты вектора v_i . Ввод векторов обычно выполняется указанием начального адреса (VA0 при $A_0=38500$), набором компонент вектора v_i и фиксацией

набора словом VI. Если вводится первый вектор, слово VA0 можно опустить, задав фиксацию словом VIS. Последующие векторы фиксируются вводом слова VI, которое автоматически перемещает указатель адреса векторов. В ряде случаев после ввода векторов желательно вновь установить исходное значение A_0 . Если оно равно 38500, то достаточно зафиксировать ввод последнего вектора словом VIF.

При вводе векторов с фиксацией словом VI возможны три варианта ввода.

1. Ввод всех v_i подряд с разделением их пробелами. После ввода v_n указывается слово VI. Этот способ наиболее приемлем для не очень длинных векторов ($n < 100$). Главное его достоинство в том, что до исполнения слова VI (нажатием клавиши перевода строки) действует система редактирования строки, позволяющая изменить значение любой компоненты вектора до её окончательного ввода в ОЗУ.

2. Ввод каждого v_i с фиксацией ввода нажатием клавиши перевода строки. После ввода v_n указывается слово VI. Недостаток способа — введенное v_i уже нельзя исправить до фиксации вектора (но можно после этого с помощью слова Адрес!).

3. Ввод v_i по группам. В пределах группы v_i разделяются пробелами. В конце ввода каждой группы нажимается клавиша перевода строки. Наконец, после ввода последнего компонента последней группы указывается слово VI. Этот способ удобен при вводе длинных векторов по частям, когда их полная проверка утомительна.

Следует помнить, что слова VIS и VI, фиксируя ввод векторов, автоматически перемещают указатель A_0 , т. е. увеличивают A_0 на 5n байт. Это позволяет вводить подряд любое число векторов с любым числом компонент в каждом из них (например, вводить построчно матрицы).

Если ввод был зафиксирован словом VIS, то начальное значение $A_0 = 38500$. Однако исполнением слов

Адрес 38498 %!

можно установить любой начальный адрес (при фиксации ввода словом VI).

Вывод v_i по заданным i предполагает, что установлено определенное значение A_0 . Например, если исполнялось слово VA0, то V@ будет выделять компоненты вектора с начальным адресом $A_0 = 38500$. Сменой A_0 можно обеспечить выделение компоненты любого другого вектора.

В математике векторы принято обозначать названиями. В FSP88 вместо этого используется указание начального адреса и длины вектора n . Однако можно легко ввести векторы с именами. Для этого используется следующая конструкция задания поименованного вектора:

: Имя A_0 38498 %! ;

где A_0 — начальный адрес вектора.

Теперь вектор можно задавать по имени

Имя $v_1 v_2 \dots v_n$ VI ;

и вызывать по имени

Имя VO;

Это иллюстрирует следующий пример:

```
Ad=38416 Ar=59334
: VECTOR 40000 38498 %! ;
VECTOR 1 2 3 4 5 VI ;
```

```
VECTOR VO ;
```

```
V1=1
```

```
V2=2
```

```
V3=3
```

```
V4=4
```

```
V5=5
```

Как отмечалось, ввод всех компонент вектора с указанием их начальных адресов (но без индексов) можно выполнять с помощью базового слова DUMP:

A_0 n DUMP ;

где A_0 и n задаются как числа (его начальный адрес и длина), либо

$a0V N \% @$ DUMP ;

С помощью слова DUMP можно просматривать адресное пространство векторов (обзорно или по частям).

§ 4.2. Операции с векторами

Для облегчения подготовки программ, содержащих операции с векторами, в FSP88 включено восемь слов, реализующих ряд элементарных операций (строки от 8 до 15 листа 4.1). Ниже дается описание этих слов.

8. Слово

sp1 _ _ _ $n+1$ 1

служит для организации цикла, управляющая переменная которого меняется от значения 1 до n .

9. Слово

VSUM _ _ _ $\sum_{i=1}^n v_i$

оставляет на вершине стека сумму всех компонент вектора.

Пример.

1 2 3 4 VIS ; (ввод вектора 1, 2, 3 и 4)

VA0 VSUM . ; (вычисление суммы компонент)

10 (результат)

10. Слово

$$\text{VPROD} \text{ --- } \prod_{i=1}^n v_i$$

оставляет на вершине стека произведение компонент вектора.

Пример.

VA0 1 2 3 4 VIF ; (ввод вектора 1, 2, 3 и 4)

VPROD . ; (вычисление произведения компонент)

24 (результат)

11. Слово

$$\text{VMAX} \text{ --- } v_{i \max}$$

оставляет на вершине стека значение максимальной по величине компоненты.

Пример.

VA0 1 2 3 4.5 VIF ; (ввод вектора)

VMAX . ; (нахождение $v_{i \max}$)

4.5 (значение $v_{i \max}$)

12. Слово

$$\text{VAMAX} \text{ --- } |v_i|_{\max}$$

оставляет на вершине стека значение максимальной по абсолютной величине (модулю) компоненты.

Пример.

1 2 -5 4 3 VIS ; (ввод вектора)

VA0 VAMAX . ; (нахождение $|v_i|_{\max}$)

5 (значение $|v_i|_{\max}$)

13. Слово

$$\text{VMIN} \text{ --- } v_{i \min} \text{ (поиск } v_{i \min})$$

оставляет на вершине стека значение минимальной по величине компоненты.

Пример.

VA0 1 2 -3 4 VIF ; (ввод вектора)

VMIN . ; (поиск $v_{i \min}$)

-3 (значение $v_{i \min}$)

14. Слово

$$\text{NV} \text{ --- } |v_i|_{\max}$$

оставляет на вершине стека значение максимальной по абсолютной

величине компоненты $|v_i|_{\max}$, делит значения v_i всех компонент на $|v_i|_{\max}$, т. е. производит нормирование их, переводя в пределы $[-1, +1]$.

Пример.

VA0 1 2 3 4 VIF ; (ввод вектора)
 NV VO ; (нормировка и вывод вектора)
 V1=0.25 (вывод компонент вектора)
 V2=0.5
 V3=0.75
 V4=1

Стек t _ _ _ ОК (указание о наличии в стеке $|v_i|_{\max}$)
 . (вывод $|v_i|_{\max}$)
 4 (значение $|v_i|_{\max}$)

15. Слово

VPOL x _ _ _ P (x)

вычисляет и оставляет на вершине стека значение полинома

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

при заданном x . Коэффициенты полинома рассматриваются как компоненты ранее введенного вектора, причем $a_i = v_{i-1}$, т. е.

a_i a_0 a_1 $a_2 \dots a_n$
 v_i v_1 v_2 $v_3 \dots v_{n+1}$

Пример.

Вычислить значение полинома

$$P(x) = 0 + 1x + 2x^2 + 3x^3 + 4x^4.$$

Организуем это так:

VA0 0 1 2 3 4 VIF ; (ввод a_i)
 1 VPOL . ; (вычисление $P(x)$ для $x=1$)
 10 (значение $P(x)$ для $x=1$)
 2 VPOL . ; (вычисление $P(x)$ для $x=2$)
 98 (значение $P(x)$ для $x=2$)

Этот способ вычисления $P(x)$ удобнее, чем описанный в § 3.4, так как a_i хранятся в ОЗУ, а не в стеке. Это исключает нарушение процесса вычислений из-за попадания в стек результатов других операций.

§ 4.3. Ввод и вывод матриц

Матрица может рассматриваться как совокупность векторов

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} & \dots & m_{1m} \\ m_{21} & m_{22} & m_{23} & \dots & m_{2m} \\ \dots & \dots & \dots & \dots & \dots \\ m_{n1} & m_{n2} & m_{n3} & \dots & m_{nm} \end{pmatrix}.$$

Компоненты матрицы — числа m_{ji} , где $j=1, 2, \dots, n$ — номер строки, $i=1, 2, \dots, m$ — номер столбца. Матрица является двумерным массивом чисел.

Матрица может быть представлена n векторами с длиной m или m векторами с длиной n (n — номер столбца, m — номер строки). Если их расположить в ряд, то будет получен один полный вектор длиной $m \cdot n$. Именно таким образом хранится матрица в ОЗУ ПЭВМ в системе FSP88 (см. рис. 4.1). Таким образом, матрица задается начальным адресом A_0 , значениями m и n (для квадратной матрицы $m=n$) и компонентами $m_{ji}=v_k$.

При таком представлении ввод матриц не требует специальных операций. В FSP88 предусмотрено выполнение различных операций с квадратными матрицами, как наиболее распространенными. Наиболее удобно расположить полный вектор такой матрицы с адреса $A_0=38500$. Тогда ввод m_{ji} выполняется по схеме:

```
m11 m12 ... m1n VIS ; (ввод строки 1)
m21 m22 ... m2n VI ; (ввод строки 2)
..... (ввод строк до n-1)
mn1 mn2 ... mnn VIF ; (ввод строки n)
```

В конце ввода указатель адреса векторов возвращается в исходное положение ($A_0=38500$).

Для облегчения работы с матрицами и их отдельными компонентами предусмотрен ряд специальных слов ввода-вывода (см. лист 4.2 матричных операций M). Эти слова представлены строками с номерами от 0 до 10.

Лист 4.2

Матричные операции

```
0 : mb %N@ DU %* %5* ;
1 : cmn a0V mb %+ a0V ;
2 : ji SW %1- %N@ %* %+ ;
3 : am ji %1- %5* a0V %+ ;
4 : M0 %N! a0V mb ERASE ;
5 : M1 %N! cmn %I C1 %I ! C5 %+ ] ;
6 : M1D M0 VA0 cmn %I C1 %I ! %N@ %1+ %5* %+ ] ;
7 : M! am ! ;
8 : M@ am @ ;
9 : M? am ? ;
10 : M0 cn1 [ cn1 [ ."M" J . ." I . ."=" J I M? CR
] ] ;
```

```

11 : n1+ %N@ %1+ ;
12 : M+ cmn %[ DU %I @ + %I ! C5 %+ ] DR ;
13 : M* cmn %[ DU %I @ * %I ! C5 %+ ] DR ;
14 : M/ C1 SW / M* ;
15 : MT cn1 [ n1+ I 1+ [ J I Me I J Me J I M! I J M!
    ] ] ;

```

Ниже описаны слова ввода-вывода для задания матриц специального вида.

0. Слово

$mb \text{ --- } 5n^2$

оставляет на вершине стека длину (в байтах) $5n^2$ полного вектора ранее введенной квадратной матрицы.

1. Слово

$стп \text{ --- } A_0 + 5n^2 A_0$

оставляет в двух ближайших к вершине стека ячейках значения $A_0 + 5n^2$ и A_0 . Используется для организации цикла, управляющая переменная которого является указателем адреса компонент вектора.

2. Слово

$ji \text{ } j \text{ } i \text{ --- } k$

оставляет на вершине стека индекс k компонент общего вектора матрицы, вычисляя его по формуле

$$k = i + (j - 1)n$$

по введенным в стек индексам j и i . Таким образом, компоненты двумерного массива m_{ji} преобразуются в компоненты v_k полного вектора матрицы.

3. Слово

$am \text{ } ji \text{ --- } A_0, ji$

по введенным в стек j и i вычисляет адрес

$$A_{0, ji} = A_0 + (K - 1) \cdot 5$$

первой ячейки ОЗУ для пяти ячеек, хранящих m_{ji} , и оставляет этот адрес на вершине стека.

4. Слово

$MO \text{ } n \text{ ---}$

для заданного на вершине стека значения n задает нулевую квадратную матрицу $m_{ji} = 0$ размером $n \times n$. Матрица задается в виде ее общего вектора длиной $5n^2$ байт с применением слова ERASE, очищающего заданную область ОЗУ (это обеспечивает предельно высокую скорость задания $m_{ji} = 0$ без организации циклов).

Пример.

3 M0 ; (задана нулевая матрица 3×3 с текущего адреса A_0)

VA0 3 M0 ; (задана нулевая матрица 3×3 с адреса $A_0=38500$)

5. Слово

M1 n _ _ _

для заданного на вершине стека значения n задает матрицу $n \times n$, у которой все $m_{ji}=1$. Для ускорения задания матрицы используется целочисленный цикл, в ходе которого компонентам полного вектора матрицы $a_k=m_{ji}$ присваивается значение 1.

Пример.

VA0 3 M1 ;

Создается матрица

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

с начальным адресом полного вектора $A_0=38500$.

6. Слово

MID n _ _ _

для заданного на вершине стека значения n задает единичную матрицу $n \times n$, у которой диагональные элементы $m_{ii}=1$, а при $i \neq j$ $m_{ij}=0$. Для ускорения задания матрицы также используются преобразование m_{ij} в a_k и один быстрый целочисленный цикл.

Пример.

VA0 3 MID ;

Создается матрица

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

с начальным адресом полного вектора $A_0=38500$.

7. Слово

M! d j i _ _ _

присваивает компоненте m_{ji} значение d . Это слово очень удобно для обеспечения различных преобразований компонент матриц, а также коррекции их ввода.

Пример.

1.75 2 3 M! ; (задает $m_{23}=1,75$)

8. Слово

M@ j i _ _ _ m_{ji}

оставляет по заданным j и i значение m_{ji} на вершине стека. Для приведенного выше примера

2 3 M@ . ; (вывод m_{23} и индикация)
1.75 (результат $m_{23}=1,75$)

9. Слово

M? j i _ _ _

для заданных j и i выводит на индикацию m_{ji} в виде

$M_{j,i} = m_{ji}$

без занесения m_{ji} на вершину стека (j и i из стека удаляются).
Для приведенного выше примера

2 3 M? (индикация m_{23})
M2,3=1.75 (результат $m_{23}=1.75$)

10. Слово

MO _ _ _

обеспечивает вывод на индикацию n и всех m_{ji} в виде

$M_{j,i} = m_{ji}$.

Пример.

1 2 3 VIS ; (ввод строки 1)
4 5 6 VI ; (ввод строки 2)
7 8 9 VIF ; (ввод строки 3)
MO ; (вывод n и m_{ji} на индукцию)
 $n=3$ (индикация n)
 $M_{1,1}=1$ (индикация $m_{1,1}$)
 $M_{1,2}=2$ (индикация $m_{1,2}$)
 $M_{1,3}=3$
 $M_{2,1}=4$
 $M_{2,2}=5$
 $M_{2,3}=6$
 $M_{3,1}=7$
 $M_{3,2}=8$
 $M_{3,3}=9$ (индикация $m_{3,3}$)
Стек 0 _ _ _ ОК

Как видно из этого примера, знание начальных адресов (или имен) векторов и матриц необходимо программисту на этапе задания типов данных. Пользователю, использующему систему FSP88 с описанным расширением, этого знать не нужно. Достаточно лишь представлять довольно простое функциональное назначение слов.

§ 4.4. Элементарные операции с матрицами

В лист 4.2 (строки 11—15) включен также ряд типовых элементарных операций над уже введенными матрицами.

11. Слово

$n1 + _ _ _ n+1$

оставляет на вершине стека значение $n+1$. Самостоятельного значения слово обычно не имеет и служит для упрощения листинга других слов.

12. Слово

$M + d _ _ _$

увеличивает на величину d значение всех компонент m_{ji} матрицы и удаляет d из стека.

Примеры.

$1 M +$; (ко всем m_{ji} прибавляется 1)

$-1 M +$; (от всех m_{ji} отнимается 1)

Из последнего примера вытекает, что при смене знака d (запись $-d$ или d NEGATE) слово $M+$ можно использовать для вычитания d из всех m_{ji} .

13. Слово

$M * d _ _ _$

умножает все m_{ji} на число d и удаляет d из стека.

14. Слово

$M / d _ _ _$

делит все m_{ji} на число d и удаляет d из стека.

15. Слово

$MT _ _ _$

создает на месте исходной матрицы транспонированную, т. е. матрицу, у которой строки есть столбцы исходной матрицы ($m_{ji}^T = m_{ij}$).

Пример.

1 2 3 VIS 4 5 6 VI 7 8 9 VIF MT

MO ;

$M1, 1=1$

$M1, 2=4$

$M1, 3=7$

$M2, 1=2$

$M2, 2=5$

$M2, 3=8$

$M3, 1=3$

$M3, 2=6$

$M3, 3=9$

Количество матричных операций у FSP88 намеренно ограничено разумным минимумом. Однако следует учесть, что имеющиеся операции над векторами и матрицами позволяют выполнять много дополнительных операций (вспомним, что, заменив n на n^2 , можно использовать операции над полным вектором матрицы). Среди таких операций отметим следующие:

- 1) обнуление ранее введенной матрицы (ввод MO),
- 2) смена знака всех m_{ji} (умножение на -1 с помощью слова M*),
- 3) поиск $m_{j\max}$ по всей матрице (с помощью слова VMAX),
- 4) поиск $|m_{ji}|_{\max}$ по всей матрице (с помощью слова VAMAX),
- 5) поиск $m_{j\min}$ по всей матрице (с помощью слова VMIN),
- 6) нормировка m_{ji} делением их на $|m_{ji}|_{\max}$ (с помощью слова NV).

Пример.

Ввести матрицу ($n=3$)

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

и пронормировать ее компоненты, разделив их на $|m_{ji}|_{\max}$. Ввод матрицы был описан раньше. После ввода задаем $N=9$ как длину полного вектора и, используя слово NV, нормируем компоненты полного вектора. Далее возвращаем величине N значение $N=3$ и, используя команду MO, выводим компоненты нормированной матрицы (в стеке остается $|m_{ji}|_{\max}$). Эти действия поясняет следующий листинг:

```

1 2 3 VIS 4 5 6 VI 7 8 9 VIF 9 N
%! NV 3 N %! MO ;
M1,1=0.11111111
M1,2=0.22222222
M1,3=0.33333333
M2,1=0.44444444
M2,2=0.55555555
M3,1=0.77777778
M3,2=0.88888889
M3,3=1

```

. ;
9

Наличие описанных выше операций над векторами и матрицами существенно облегчает решение более сложных задач линейной алгебры. В § 4.5 рассматриваются две такие важнейшие задачи: обращение матрицы и решение систем линейных уравнений.

§ 4.5. Обращение матриц и решение систем линейных уравнений

Пусть имеется матрица M . Обращенной (обратной или инвертированной) матрицей называют матрицу M^{-1} , удовлетворяющую условию $M \cdot M^{-1} = M_1$, где M_1 — единичная матрица. Обращение матриц

обычно производится одной из модификаций алгоритма Гаусса и сводится к проведению соответствующих преобразований компонентов m_{ij} исходной матрицы. Для экономии памяти компоненты M^{-1} заносятся на место исходных. В этом параграфе — 1 является не показателем степени, а указанием на отношение к операции инвертирования.

В описанном далее листе обращение матрицы выполняется по следующему алгоритму.

1. К диагональным элементам m_{ij} прибавляется 1 ($m_{ij} \leftarrow m_{ij} + 1$).

2. Для M от 1 до N вычисляется $P = M_{MM} - 1$ и проверяется условие

$$P = m_{ij} - 1 = 0.$$

Если оно выполняется, матрица является вырожденной и не может инвертироваться. Поэтому печатается сообщение КОНЕЦ и счет останавливается без выборки результатов. В противном случае компоненты m_{ij} преобразуются по формулам:

$$m(M, i) \leftarrow m(M, i) / P \text{ для } i = 1, 2, \dots, N;$$

$$Q \leftarrow m(j, M) \text{ для } j = 1, 2, \dots, N \text{ и } j \neq M;$$

$$m(j, i) \leftarrow m(j, i) - Q \cdot m(M, i) \text{ для } i = 1, 2, \dots, N.$$

3. Для j от 1 до N получаем

$$m_{ij}^{-1} \leftarrow m_{ij} - 1,$$

т. е. отнимаем 1 от диагональных элементов.

В результате этих преобразований получают компоненты m_{ij}^{-1} обратной матрицы M^{-1} . В процессе инвертирования возникает необходимость в организации трех циклов с вложениями с управляющими переменными M , j и i (см. п. 2 алгоритма).

Важнейшее применение матричных операций — решение систем линейных уравнений

$$AX = B, \tag{4.1}$$

где A — матрица коэффициентов, B — вектор свободных членов и X — вектор неизвестных. Приняв обозначения $A = M$ и $A^{-1} = M^{-1}$, решение (4.1) можно найти в виде

$$X = B \cdot A^{-1} = B \cdot M^{-1}, \tag{4.2}$$

т. е. как результат перемножения вектора B на обратную матрицу M^{-1} коэффициентов исходного уравнения (4.1). Если имеется ряд систем линейных уравнений

$$AX_1 = B_1, AX_2 = B_2, \dots, AX_m = B_m,$$

имеющих разные векторы b_j , но одинаковые матрицы A , то обращение матрицы A достаточно провести один раз и искать решение по формуле (4.2) для всех X_j и B_j .

Вычисление компонент вектора X выполняется по формуле

$$x_i = \sum_{i=1}^N m_{ii}^{-1} b_i,$$

где N — порядок системы (количество уравнений в ней). Приведенный далее лист содержит ряд слов, обеспечивающих обращение матрицы $A=M$, ввод вектора B и решение систем линейных уравнений для заданных A и разных B .

Лист 4.3.

Обращение матрицы и решение систем линейных уравнений

```

0 : MD1+ cn1 %I %I DU M@ 1+ %I DU M! %J ;
1 : MD1- cn1 %I %I DU M@ 1- %I DU M! %J ;
2 : mc 0 @ DU M@ 1- DU F ! 0= IF ."КОНЕЦ" QUIT ELSE
THEN ;
3 : mp/ 0 @ G @ M@ F @ / 0 @ G @ M! ;
4 : mq! L @ 0 @ M@ 0 ! ;
5 : mq mq! cn1 [ L @ I M@ 0 @ 0 @ I M@ * - L @ I M!
] ;
6 : m5 0 ! mc cn1 [ I G ! mp/ ] cn1 ;
7 : m6 L ! L @ 0 @ ;
8 : MI MD1+ cn1 [ I m5 [ I m6 = IF ELSE mq THEN ] ]
MD1- ;
9 : V0 %5* DU a0V SW ERASE c38498 %+! ;
10 : aab c38500 mb %+ DU B %! %N@ %5* %+ A %! ;
11 : VBI aab B %@ c38498 %! VI %N@ V0 VA0 ;
12 : m7 M@ * SW A %@ + DU ROT SW @ + SW ! ;
13 : ij 1- C5 * ;
14 : MV* cn1 [ cn1 [ J ij I ij B %@ + @ J I m7 ] ] ;
15 : XSLE VBI MV* A %@ c38498 %! V0 ;

```

В этот листинг входят следующие слова.

Слово

MD1+ — — — ($m_{ij} \leftarrow m_{ij} + 1$)

обеспечивает прибавление 1 ко всем диагональным элементам m_{ij} матрицы A :

Слово

MD1- — — — ($m_{ij} \leftarrow m_{ij} - 1$)

обеспечивает вычитание 1 от всех диагональных элементов m_{ij} матрицы M .

Слово

mc — — — (контроль m_{ij})

проверяет m_{ij} на равенство нулю. Если $m_{ij} = 0$, останавливает счет с выводом сообщения КОНЕЦ, свидетельствующего о вырождении матрицы и невозможности ее обращения.

Слова mp/, mq!, mq, m5 и m6 не имеют самостоятельного значения и используются для проведения типовых операций инвертирования матрицы M (см. описание алгоритма обращения матрицы).

Слово

MI _ _ _ ($m_{j \leftarrow m_j^{-1}}$)

обеспечивает инвертирование (обращение) матрицы M . Компоненты $m_{j \leftarrow m_j^{-1}}$ обращенной матрицы M^{-1} заносятся на место исходных компонент m_{ji} (если нужно сохранить исходную матрицу, ее можно перевести на новое место перед инвертированием, воспользовавшись словом MOVE).

Слово

aab _ _ _ ($A \leftarrow 38500 + 5MN + 5N$, $B \leftarrow 38500 + 5MN$)

задает переменным A и B указанные выше (в скобках) значения.

Слово

VBI _ _ _ (ввод b_i)

обеспечивает фиксацию ввода свободных частей b_i уравнения (4.1).

Слова m_j и i_j не имеют самостоятельного значения и используются как вспомогательные слова для сокращения длины последующих словарных статей.

Слово

MV* _ _ _

обеспечивает умножение матрицы $M \leftarrow M^{-1}$ на вектор B .

Слово

XSLE _ _ _ (вывод $x_i = v_i$)

фиксирует конец операций при решении системы линейных уравнений и выводит на индикацию вектор решения $v_i = x_i$.

В общем случае решение системы из N линейных уравнений с применением описанных выше слов выполняется по следующей схеме:

$m_{11} \ m_{12} \ \dots \ m_{1N}$ VIS ; Ввод первой строки матрицы M

$m_{21} \ m_{22} \ \dots \ m_{2N}$ VI ; Ввод второй строки матрицы M

$m_{N1} \ m_{N2} \ \dots \ m_{NN}$ VIF ; Ввод N -й строки матрицы M

MI ; Инвертирование матрицы

$b_1 \ b_2 \ \dots \ b_N$ XSLE ; Ввод b_i и задание ввода x_i

Последний пункт можно повторить для новых b_i .

Пример. Решить системы уравнений

$$\begin{bmatrix} 4 & 8 & 0 \\ 8 & 8 & 0 \\ 2 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \\ 6 \end{bmatrix} \text{ и } \begin{bmatrix} 12 \\ 32 \\ 4 \end{bmatrix}.$$

Ниже дан листинг, показывающий решение этих систем уравнений:

```
4 8 0 VIS 8 8 8 VI 2 0 1 VIF MI
4 4 6 XSLE ;
V1=4
V2=-1.5
V3=-2
```

```
12 32 4 XSLE ;
V1=1
V2=1
V3=2
```

Описанные выше слова могут использоваться и для решения других задач линейной алгебры.

§ 4.6. Одномерная статистика вектора

Пусть задан вектор из n компонент:

$$v_1 \ v_2 \ v_3 \ \dots \ v_n$$

Он может представлять собой сводку данных эксперимента (например, измерение какого-либо размера детали, напряжения в электрической сети и т. д.). Тогда большой интерес представляют статистические характеристики такого массива: среднее значение \bar{v}_i , дисперсия, начальные и центральные моменты и т. д. Такая статистическая обработка вектора (одномерная статистика) реализуется с помощью слов, указанных в листинге STAT1.

Лист 4.4.

Одномерная статистика вектора

```
0 : n/ %N@ / ;
1 : m12 C0 cn1 %[ %I V@ 2^ + %] n/ DU U ! VSUM n/ DU
  T ! ;
2 : m3 C0 cn1 %[ %I V@ 3^ + %] n/ DU V ! ;
3 : m4 C0 cn1 %[ %I V@ 2^ 2^ + %] n/ DU W ! ;
4 : M2 U @ T @ 2^ - DU X ! ;
5 : M3 V @ c3 T @ * U @ * - C2 T @ 3^ * + DU Y ! ;
6 : M4 W @ c4 V @ * c6 U @ * c3 T @ 2^ * - T @ * - T
  @ * - DU Z ! ;
7 : d0ae X@ %N@ * %N@ %1- / Y@ X@ 3^ SQR / Z@ X@ 2^
  / c3 - ;
8 : .P . CR ;
9 : .m m12 ."m1=" .P ."M2=" .P m3 ."m3=" .P m4 ."m4="
  " .P ;
10 : .M M2 ."D=M2=" .P M3 ."M3=" .P M4 ."M4=" .P ;
11 : .stat .m .M d0ae ROT ."D0=" .P SW ."A=" .P ."E="
  " .P ;
12 : U3 %N@ %1- c6 * %N@ %1+ / %N@ c3 %+ / SQR ;
13 : u4 N %@ 2DU %2- %* SW 3 %- %* 24 %* ;
14 : U4 u4 %N@ %1- 2^ %N@ c3 %+ * %N@ c5 %+ * / SQR
  ;
15 : STAT1 ."N=" N %? CR .stat U3 ."U3=" .P U4 ."U4="
  " .P ;
```

0. Слово

n/ v _ _ _ v/N

заменяет число v на вершине стека значением v/N ($N=n$).

1. Слово

m12 _ _ _ m1 m2

вычисляет начальные моменты первого и второго порядка

$$m_1 = \frac{1}{N} \sum_{i=1}^N v_i \quad \text{и} \quad m_2 = \frac{1}{N} \sum_{i=1}^N v_i^2,$$

помещает их в стек и присваивает переменным T и U значения m_1 и m_2 соответственно.

2. Слово

m3 _ _ _ m3

вычисляет начальный момент третьего порядка

$$m_3 = \frac{1}{N} \sum_{i=1}^N v_i^3$$

помещает m_3 на вершину стека и присваивает переменной V значение m_3 .

3. Слово

m4 _ _ _ m4

вычисляет начальный момент четвертого порядка

$$m_4 = \frac{1}{N} \sum_{i=1}^N v_i^4,$$

помещает m_4 на вершину стека и присваивает переменной W значение m_4 .

4. Слово

M2 _ _ _ M2

вычисляет центральный момент второго порядка M_2 (и дисперсию $D=M_2$)

$$D = M_2 = \frac{1}{N} \sum_{i=1}^N (v_i - \bar{v}_i)^2 = m_2 - m_1^2,$$

помещает его значение на вершину стека и присваивает переменной X значение M_2 .

5. Слово

M3 _ _ _ M3

вычисляет центральный момент третьего порядка

$$M_3 = m_3 - 3m_1m_2 + 2m_1^3,$$

помещает его значение на вершину стека и присваивает переменной Y значение M3.

6. Слово

M4 _ _ _ M4

вычисляет центральный момент четвертого порядка

$$M_4 = m_4 - 4m_1m_3 + 6m_1^2m_2 - 3m_1^4,$$

помещает M4 на вершину стека и присваивает переменной Z значение M4.

7. Слово

d0ae _ _ _ E A D0

вычисляет несмещенную дисперсию

$$D_0 = M_2N/(N-1),$$

коэффициент асимметрии

$$A = \frac{1}{ND^{3/2}} \sum_{i=1}^N (v_i - \bar{v}_i) = \frac{M_3}{M_2^{3/2}}$$

и коэффициент эксцесса

$$E = \frac{1}{ND^2} \sum_{i=1}^N (v_i - \bar{v}_i)^4 = \frac{M_4}{M_2^2} - 3.$$

помещает вычисленные значения в стек.

8. Слово

.P d _ _ _ (Вывод на печать d с переводом строки)

9. Слово

m _ _ _

вычисляет m1, m2, m3 и m4 и выводит их на индикацию.

1С. Слово

M _ _ _

вычисляет m1, m2, m3, m4, D=M2, M3 и M4 и выводит их на индикацию.

11. Слово

stat _ _ _ _

вычисляет $m_1, m_2, m_3, m_4, D=M_2, M_3, M_4, D_0, A$ и E и выводит их на индикацию.

12. Слово

U3 _ _ _ _ U_3

Вычисляет коэффициент

$$U_3 = \sqrt{\frac{6(N-1)}{(N+1)(N+3)}}$$

и выводит его на вершину стека.

13. Слово

u4 _ _ _ _ p

вычисляет вспомогательную величину

$$p = 24(N-2)(N-3)$$

и выводит ее на вершину стека.

14. Слово

U4 _ _ _ _ U_4

вычисляет коэффициент

$$U_4 = \sqrt{\frac{p}{(N-1)^2(N+3)(N+5)}}$$

и выводит его на вершину стека.

15. Слово

STATI _ _ _ _

вычисляет все перечисленные выше статистические параметры вектора и выводит их на индикацию в указанном порядке.

Практически весь объем описанных вычислений реализуется с помощью трех слов:

VA0 — установка начального адреса вектора, после чего следует ввод компонент его,

VI — фиксация ввода,

VA0 STATI — возврат начального адреса и команда выдачи результатов.

Пример ввода на обработку вектора из десяти чисел:

VA0 9 8 10 9 11 12 10 10 9 11 VI

;

Результат вычислений имеет вид

```
VA0 STAT1 ;  
N=10  
m1=9.9  
M2=99.3 1  
m3=1008.9  
m4=10379.7  
D=M2=1.2900001  
M3=0.28799868  
M4=3.737751  
D0=1.4333334  
A=0.19656487  
E=-0.75389057  
U3=0.61450987  
U4=0.92244359
```

Аналогичным образом могут задаваться слова для других статистических расчетов. В следующей главе, в частности, описано вычисление функции и интеграла вероятности для нормального распределения.

РЕАЛИЗАЦИЯ ОСНОВНЫХ ЧИСЛЕННЫХ МЕТОДОВ ОБЩЕГО НАЗНАЧЕНИЯ

§ 5.1. Вычисление определенного интеграла функции одной переменной

В этой главе описываются численные методы общего назначения, обычно требующие применения специальных алгоритмов для своей реализации. Специфика системы FSP88 в том, что и эти методы доведены до слов, входящих в расширенную версию системы (имеющую тем не менее достаточный объем ОЗУ для дальнейшего ее наращивания). Включение всех этих возможностей в систему FSP88 делает ее, по существу, языком сверхвысокого уровня, ориентированным на сложные расчетные операции как в непосредственном режиме, так и в режиме вычислений по программам. Рассмотрение этих возможностей начнем с вычислений определенных интегралов.

Определенный интеграл функции одной переменной $F(x)$ имеет вид

$$I = \int_A^B F(x) dx.$$

Вычисление определенного интеграла — одна из наиболее типовых вычислительных задач.

В состав FSP88 включены два слова (FS и FSE), реализующие вычисление определенного интеграла по двум наиболее распространенным алгоритмам: Симпсона (при задании $F(x)$ с равномерным расположением узлов) и Гаусса (при неравномерном расположении узлов).

При интегрировании методом Симпсона отрезок $[A, B]$ разбивается на m отрезков равной длины. Интеграл вычисляется по составной формуле Симпсона

$$I = \frac{H}{3} \left[F(A) + F(B) + 2 \sum_{l=1}^{N-1} F(A + 2lH) + 4 \sum_{l=1}^N F(A - H + 2lH) \right],$$

где $N = m/2$, $H = (B - A)/m$. Вычисления реализованы словом:

FS A B N — — — I (интеграл по Симпсону)

При интегрировании методом Симпсона абсциссы $F(x)$ в узловых точках легко вычисляются, причем они включают пределы A и B . Если

$F(x)$ имеет особенности в промежутке между A и B (но не при $x=A$ или $x=B$), то выбор N должен быть таким, чтобы значения абсциссы не приняли значений, характерных для точек с особенностями.

Во многих случаях $F(x)$ имеет особенности при $x=A$ или $x=B$. Примером может служить функция $F(x)=\sin x/x$. Интеграл

$$I = \int_A^B (\sin x/x) dx$$

при $A=0$ нельзя вычислить методом Симпсона, так как при вычислении $F(A)$ имеет место деление на 0, что ведет к остановке счета (хотя $\sin x/x \rightarrow 1$ при $x \rightarrow 0$).

От этого недостатка свободен метод Гаусса. В FSP88 используется метод Гаусса второго порядка. Отрезок $[A, B]$ разбивается на m отрезков. В пределах каждого текущие границы обозначим как a и b , причем $H=b-a=(B-A)/m$. На каждом i -м отрезке x приводится в отрезок $[-1, 1]$ с помощью соотношения

$$x_j = \frac{b+a}{2} + \frac{b-a}{2} t_j.$$

При $j=1, 2, 3$ имеем соответственно $t_1=\sqrt{0,6}$, $t_2=0$, $t_3=-\sqrt{0,6}$. Затем вычисляется

$$I_i = \int_{-1}^{+1} F(t) dt = \sum_{j=1}^3 A_j F(t_j),$$

где при $j=1, 2, 3$ имеем соответственно $A_1=5/9$, $A_2=8/9$, $A_3=5/9$. Тогда

$$I = \sum_{i=1}^m I_i.$$

Для ускорения вычисления значения $\sqrt{6}$, $5/9$ и $8/9$ целесообразно вычислить заранее. В FSP88 реализована процедура вычисления интеграла методом Гаусса с заданной погрешностью E . Для этого задается вычисление I для ряда $m=2, 4, 8$ и т. д. Если разность между вычисленными смежными значениями I менее $60E$, то продолжают вычисления. В противном случае они прекращаются.

Вычисление определенного интеграла с заданной точностью методом Гаусса задается словом

FSE A B E — — — I (интеграл по Гауссу)

Вычисления двумя описанными методами реализованы строками 0—12 листа FSE (см. ниже).

Вычисление определенного интеграла методами Симпсона и Гаусса

```

0 : fs0 C2 %* Z@ * X@ + ;
1 : fs1 C0 %N@ C1 %I %I fs0 F@ + %J C2 * ;
2 : fs2 C0 %N@ %I+ C1 %I %I fs0 Z@ - F@ + %J c4 * ;
3 : fs3 %N! Y! X! Y@ X@ - 2/ %N@ / Z! ;
4 : FS fs3 X@ F@ Y@ F@ + fs1 + fs2 + Z@ * c3 / ;
5 : fs4 60 * Y! B ! X! C1 %N! C0 P ! SQR.6 T ! ;
6 : fs5 %N@ C2 %* %N! X @ A ! C0 S ! B @ A @ - %N@ /
  H ! ;
7 : fs6 A @ 2* H @ + 2/ DU C ! A @ - D ! c5/9 D @ *
  E ! ;
8 : fs7 c8/9 D @ * L ! D @ T @ * D ! C @ D @ - F@ ;
9 : fs8 @ * S @ + S ! ;
10 : fs9 E fs8 C @ F@ L fs8 C @ D @ + F@ E fs8 A @ H
  @ + A ! ;
11 : fsc %N@ %I+ C1 %I fs6 fs7 fs9 %J P @ L ! S @ P
  ! ;
12 : FSE fs4 BEGIN fs5 fsc S @ L @ - ABS Y@ <= UNTIL
  S @ ;
13 : ffse FA! FSE ;
14 : ERF C0 SW EPS ' P(X) ffse ;
15 : Si C0 SW EPS ' SINX/X ffse ;

```

Метод Симпсона реализован в строках 0—4 листа. Из них основной является строка 4 со словами FS (остальные строки вспомогательные и просто дробят программу на небольшие части). Метод Гаусса реализован в строках 5—12. Из них строка 12 основная и задает слово FSE (остальные строки вспомогательные).

Для применения слов FS и FSE необходимо задать функцию $F(x)$ и определить ее начальный адрес. Ниже показаны задание функции $F(x)=\sqrt{2x+1}$ и вычисление интеграла

$$I = \int_0^1 \sqrt{2x+1} dx$$

двумя описанными методами:

```

Ad=38424   Ar=59354
: FX 2* 1+ SQR ;
' FX FA! ;

```

```

0 1 8 FS . ;
1.3987172
0 1 1E-6 FSE . ;
1.3987175

```

Следует отметить, что на практике могут встречаться случаи, когда численное интегрирование с заданной погрешностью (слово FSE) невозможно. Это может случиться, когда заданная погрешность E слишком мала, а $F(x)$ вычисляется с большей погрешностью. В этом случае процесс умножения m на 2 может продолжаться до бесконечности. В FSP88 не введен контроль над m , что может в подобных случаях

привести к заикливание операции интегрирования методом Гаусса. Поэтому в сомнительных случаях рекомендуется вначале вычислять интеграл с не очень малым E (например, 0,01 или 0,001). При необходимости контроль над m легко ввести в описанную процедуру интегрирования, останавливая счет, если m достигает больших значений (например, $m \geq 64$).

Численное интегрирование часто используется для вычисления специальных функций. Здесь мы ограничимся вычислением интеграла вероятности

$$\text{erf}(x) = \int_0^x p(x) dx,$$

где $p(x) = (2/\sqrt{\pi}) e^{-x^2}$, и интегрального синуса

$$\text{Si}(x) = \int_0^x ((\sin x)/x) dx,$$

подынтегральные функции которых были ранее заданы. Для этого в строках 13—15 листа 5.1 заданы слова:

```
ERF X _ _ _ erf(X)
Si X _ _ _ Si(X)
```

Вычисления реализованы с применением численного интегрирования методом Гаусса (т. е. слова FSE) при $E = \text{EPS} = 1 \cdot 10^{-8}$. Нижний предел A задан равным 0, а верхний предел B — равным X . Затем вычисляется адрес A_r нужной функции ($P(X)$ для ERF и SIN X/X для Si) и исполняется слово ffse, т. е. фиксируется адрес компиляции и идет обращение к слову FSE.

Следующий пример иллюстрирует вычисление функций ERF и Si:

```
1.25 ERF . ;
0.92290015
1 Si . ;
0.94608307
```

В § 5.8 рассмотрено вычисление других специальных функций.

§ 5.2. Вычисление корней и экстремумов нелинейных функций

Пусть имеется система нелинейных зависимостей $F(X_i)$, где X_i — вектор аргументов. Важнейшее место в анализе такой системы занимает вычисление корней системы нелинейных уравнений

$$F(X_i) = 0, \tag{5.1}$$

а также поиск экстремумов $F(X_i)$. Существует огромное количество численных методов для решения этих задач [8, 11, 19]. Однако многие

практически важные задачи можно решать с помощью всего двух методов, реализованных в системе FSP88.

Первый метод — вычисление всех действительных корней одного нелинейного уравнения

$$f(x)=0.$$

поразрядным изменением аргумента x . Алгоритм этого метода следующий.

1. Задаем отрезок поиска $[A, B]$, погрешность вычислений E и начальный шаг H изменения аргумента x .

2. Задаем $C=H$, $X=A$, $K=0$ и вычисляем $W=\text{sign} E(x)$, где sign — функция определения знака, реализованная словом SGN.

3. Проверяем условие $X < B$. Если оно выполняется, идем дальше; в противном случае заканчиваем вычисления и организуем останов ПЭВМ.

4. Задаем $X=X+C$, вычисляем $F(X) \cdot W/C$ и проверяем условие $F(X) \cdot W/C > 0$. Если оно выполняется, идем к п. 3, иначе — к п. 5.

5. Задаем $C=-C/R$, где R — показатель разрядности (уменьшения шага C), взятый равным 4. Изменение знака C обеспечивает изменение направления поиска. Проверяем выполнение условия $|C| > E/R$. Если оно выполняется, идем к п. 3, иначе — к п. 6.

6. Выводим на печать (индикацию) значение K -го корня $X_K=X$, задаем $K=K+1$, $C=H$ и $W=-W$, после идем к п. 3.

Этот алгоритм реализован в строках 0—6 листа 5.2.

Лист 5.2

Вычисление корней уравнения $f(x)=0$ и минимума функции ряда переменных $F(x_i)$

```

0 : f0 H ! E ! B ! A ! H @ C ! A @ X ! F @ SGN W ! C1
N ! ;
1 : f1 X @ B @ < ;
2 : f2 C @ X + ! F @ W @ * C @ / 0 > ;
3 : f3 C @ NEG c4 / DU C ! ABS E @ c4 / > ;
4 : f4 CR ."X" N ? ."=" X ? H @ C ! W @ NEG W ! C1 N
+ ! ;
5 : f5 f3 IF ELSE f4 THEN ;
6 : XF=0 f0 BEGIN f1 WHILE f2 IF REPEAT ELSE THEN f5
REPEAT ;
7 : V ! SW %1- %5* a0V %+ ! ;
8 : o1 H ! E ! VIS VA0 ;
9 : o2 DU V @ Z ! F @ V ! Z @ H @ - V ! F @ W ! ;
10 : o3 Z @ 2* DU H @ + W @ * SW H @ - U @ * + Z @ V @
* c4 * - ;
11 : o4 o3 W @ V @ 2* - U @ + / 2 / DU Z @ - ABS E @ >
;
12 : o5 Z @ H @ + V ! F @ U ! o4 IF C1 S ! ELSE THEN ;
13 : o6 C0 S ! cn1 %I %I o2 %I o5 %I SW V ! %I CR ."F
=" V ? ;
14 : FMIN o1 BEGIN o6 S @ 0= UNTIL CR CR V0 CR ."Fmi
n=" F @ . ;
15 : FXI C1 V @ C2 V @ c3 V @ + + EXP C1 V @ C2 V @ 2^ c3
V @ 3^ * * / ;

```


Из семи слов, реализующих данный алгоритм, основным является $XF=0$ (вычисление x_i , при которых $f(x_i)=0$). Оно используется следующим образом:

$A \ B \ E \ H \ XF=0$

Перед этим необходимо задать слово, вычисляющее функцию $f(x_i)$, и определить начальный адрес этого слова с помощью слов 'Имя $f(x_i)$ FA! ;

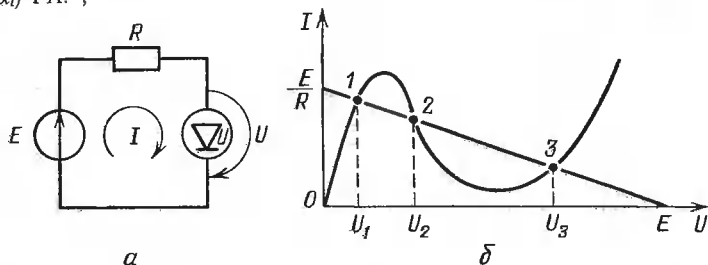


Рис. 5.1. Цепь с туннельным диодом (а), имеющим N -образную вольт-амперную характеристику (б)

Рассмотрим пример. Имеется цепь (рис. 5.1) с туннельным диодом, вольт-амперная характеристика которого описывается уравнением

$$I(U) = A_1 U e^{-\alpha U} + D(e^{\beta U} - 1). \quad (5.2)$$

Чтобы определить падение напряжения на диоде, надо решить нелинейное уравнение

$$f(U) = (E_n - U)/R - I(U) = 0. \quad (5.3)$$

Зависимость (5.2) N -образная, поэтому в общем случае решение может дать три точки пересечения $I(U)$ линий нагрузки резистора R .

Возьмем (опуская единицы измерения вольт, ампер и ом) $E_n = 1$, $A_1 = 0,2718$, $\alpha = 10$, $D = 1 \cdot 10^{-8}$, $\beta = 20$ и $R = 125$. Тогда, считая $X = U$, запишем уравнение (5.3) в виде

$$f(X) = (1 - X)/125 - 0,2718 X e^{-10X} + 1 \cdot 10^{-8} (e^{20X} - 1).$$

На листе 5.3 показано задание функции $f(X)$, определение ее начального адреса и решение уравнения $f(X) = 0$.

Лист 5.3

Пример решения уравнения $f(X) = 0$

Ad=38432 Ar=59364

: F(X) 1 X@ - 125 / -10 X@ * EXP

X@ * 0.2718 * - 20 X@ * EXP 1-

EPS * - ;

F(X) FA! ;

0 1 .001 0.1 XF=0 ;

X1=.04375

X2=0.234375

X3=0.625

Как видно, при введенных данных решение действительно даёт три корня (заданы $A=0$, $B=1$, $E=0,001$ и $H=0,1$).

Отметим, что отрезок $[A, B]$, погрешность E и начальный шаг H следует задавать с позиций разумных приближений. Так, задание большого H может привести к тому, что один или даже все корни окажутся невычисленными, так как аргумент X сразу проскочит область, в которой они находятся. Задание слишком малых E может породить неустойчивость решения. Эти проблемы в той или иной мере характерны для всех численных методов решения нелинейных уравнений.

Решение системы нелинейных уравнений $F(X_i)=0$ — задача значительно более сложная, чем решение одного уравнения. Чаще всего это решение выполняется методом Ньютона — Рафсона при задании компонентам вектора X_i начальных значений [8, 11]. В принципе, этот метод несложно реализовать на языке FSP88, поскольку главная его часть заключается в составлении матрицы приращений (Якоби) и ее обращения. Однако с целью экономии памяти и повышения универсальности реализованных алгоритмов целесообразно свести решение уравнения $F(x_i)=0$ к поиску минимума функций

$$f(x_i) = \sum_{i=1}^n |F_i(x_i)| \quad \text{или} \quad f(x_i) = \sum_{i=1}^n (F_i(x_i))^2.$$

Таким образом, эта задача сводится к другой важной задаче анализа нелинейных функций — поиску их экстремумов. Поскольку поиск максимума $F(X)$ аналогичен поиску минимума $-F(X_i)$, ограничимся иллюстрацией решения задачи поиска минимума функции $f(x_i) = f(x_1, x_2, \dots, x_n)$ n переменных x_i . Здесь мы вновь сталкиваемся с проблемой выбора численного метода — имеется множество методов минимизации функций ряда переменных, учитывающих те или иные их особенности.

Ограничимся реализацией метода координатного спуска с квадратичной интерполяцией функции при спуске по каждой координате. Считаем, что функция вблизи области поиска, намеченной начальными значениями переменных x_{i0} , имеет один минимум и не содержит разрывов. В этом случае данный метод достаточно эффективен и обеспечивает сокращение времени поиска в 1,5—2 раза в сравнении с методами координатного и спирального спуска [6, 11].

В данном методе ищется последовательно минимум $f(x_1, x_2, \dots, x_n)$ вначале по переменной x_1 , затем x_2 и т. д. до x_n . На каждом таком цикле переменной S присваивается значение 0 и вычисляются три значения функции:

$$V = f(x_{i0}), \quad W = f(x_{i0} - H) \quad \text{и} \quad U = f(x_{i0} + H),$$

где в скобках указано значение изменяемой (с шагом H) переменной x_{i0} . Затем с помощью аналитической формулы для квадратичной ин-

терполяции вычисляется значение

$$\tilde{x}_{im} = \frac{W(2x_{i0} + H) + 4Vx_{i0} + U(2x_{i0} - H)}{W - 2V - U},$$

т. е. x_i в точке аналитического минимума. Далее проверяется условие

$$|\tilde{x}_{im} - x_{i0}| > E,$$

где E — погрешность вычисления. Если это условие выполняется, переменной S с начальным значением $S=0$ присваивается значение 1, что указывает на необходимость повторения циклов минимизации. Если данное условие выполнено, т. е. погрешность вычисления всех x_i в точке минимума меньше заданной, счет останавливается и выводятся значения вектора $V_i = x_i$. Для контроля сходимости в конце каждого цикла следует выводить значение $f(x_i)$, а по окончании минимизации $F_{\min} = f(x_{i\min})$.

Этот алгоритм реализован в строках 8—14 листа 5.2. Для минимизации функции $f(x_i)$ необходимо:

1. Задать вычисление функции $f(x_i)$ в виде слова с определенным именем (имя функции).

2. Задать начальный адрес функции

'Имя функции FA! ;

3. Исполнить слово FMIN:

$x_{10} x_{20} \dots x_{n0}$ E H FMIN;

В строке 15 листа 5.2 словом FXI задано вычисление типовой контрольной функции

$$f(x_1, x_2, x_3) = e^{x_1 + x_2 + x_3} / (x_1 x_2^2 x_3^3), \quad (5.4)$$

имеющей минимум при $x_1=1$, $x_2=2$ и $x_3=3$.

Эту функцию можно использовать для проверки программы минимизации функций ряда переменных, указанной выше.

Результат выполнения контрольного примера имеет следующий вид:

FXI FA! 1 1 1 .001 .01 FMIN ;

F=15.767692

F=8.767287

F=5.792765

F=4.3903284

F=3.8381877

F=3.738154

F=3.7354532

F=3.7354518

V1=1.0000349

V2=2.0000299

V3=3.0000299

Fmin=3.7354518

При задании малой погрешности E метод может не сходиться. Поэтому рекомендуется контроль за значениями $f(x_i)$ по ходу выполнения программы. Если видно, что сходимости нет, следует остановить вычисления (в данной реализации FSP88 для этого используется клавиша BREAK, нажимаемая при появлении надписи "Scrool ?").

§ 5.3. Решение систем обыкновенных дифференциальных уравнений

Решение многих задач моделирования физических процессов, расчета электрических цепей и т. д. сводится к решению систем обыкновенных дифференциальных уравнений

$$\begin{aligned} dy_1/dx &= F_1(x, y_1, \dots, y_n), \\ dy_2/dx &= F_2(x, y_1, \dots, y_n), \\ &\dots \\ dy_n/dx &= F_n(x, y_1, \dots, y_n). \end{aligned} \tag{5.5}$$

Для простоты рассмотрим одно уравнение

$$dy/dx = F(x, y).$$

Чаще всего для его решения используется хорошо себя зарекомендовавший метод Рунге — Кутты 4-го порядка. При нем ищется решение $y(x)$ при начальных $x = x_0, y = y_0$ и шаге изменения x , равном h . Каждая точка решения находится по известной предыдущей точке с помощью формул

$$\begin{aligned} K_1 &= hF(x, y), \quad K_2 = hF(x + h/2, y + K_1/2), \\ K_3 &= hF(x + h/2, y + K_2/2), \\ K_4 &= hF(x + h, y + K_3), \\ y_{i+1} &= y_i + (K_1 + 2K_2 + 2K_3 + K_4)/6. \end{aligned}$$

Если решается система уравнений, то все эти вычисления приходится повторять n раз, где n — число уравнений. При этом вместо функции $F(x, y)$ используется n функций $F_i(x, y_1, y_2, \dots, y_n)$ и в каждом цикле нужно выполнять вычисления $K_1 - K_4$ и y_{i+1} . Этот метод реализован в системе FSP88 с помощью слов, помещенных в лист 5.4.

Лист 5.4.

Слова для решения системы из $n \leq 10$ обыкновенных дифференциальных уравнений

```
0 : r1 H ! Z! X! VIS VA0 a0V c38550 C5 MOVE CR ;
1 : ww 30 %+ V@ H @ * DU ;
2 : r0 20 %+ SW V! ;
3 : r1 c10 %+ V@ + ;
4 : k1 F@ cn1 %I %I ww %I r0 2/ %I r1 %I SW V! %I ;
5 : cx H @ 2/ X +! F@ ;
```

```

6 : r3 20 %+ V@ + ;
7 : k2 cx cn1 %C %I ww 2* %I r3 %I r0 2/ %I r1 %I SW
V! %J ;
8 : k3 cn1 %C %I ww 2* %I r3 %I r0 %I r1 %I SW V! %J
;
9 : x? cx ."X=" X ? CR ;
10 : y? 2DU ."Y" . ."=" V@ DU . CR SW c10 %+ SW V! ;
11 : k4 %C %I ww DR %I r3 c6 / %I r1 %I SW V! %I y?
%J ;
12 : RK ri BEGIN k1 k2 k3 x? cn1 k4 X@ Z@ H @ - >= U
NTIL ;
13 : VY %1- %5* a0V %+ ;
14 : VF %1- %5* 38650 %+ ;
15 : df 2 VY @ 1 VF ! 1 VY @ X@ / 2 VY @ - X@ / 1 VY
@ - 2 VF ! ;

```

Большинство слов служебные. Основным словом является слово RK $y_{10}, y_{20} \dots y_{n0} x_0 b h \text{ --- } \text{---}$

Его входными параметрами являются вектор начальных значений $y_{10}, y_{20}, \dots, y_{n0}$, начальное значение $x—x_0$, конечное значение x и шаг изменения $x—h$. Результат на каждом шаге выдается в форме

```

X=Значение
Y1=Значение
Y2=Значение
.
.
YN=Значение

```

Однако перед применением слова RK нужно задать слово для вычисления функции $F_i=dy_i/dx$. Для облегчения этого в строках 13 и 14 листа заданы переменные

```

VY i --- Адрес  $y_i$ 
VF i --- Адрес  $F_i$ 

```

Они позволяют записывать словарную статью для всех функций, используя следующие операции вызова значений компонентов массивов y_i и F_i и записи в них:

Считывание	Запись в массив
i VY @ (дает y_i)	y_i i VY ! ($y(i) \leftarrow y_i$)
i FY @ (дает F_i)	F_i i FY ! ($F(i) \leftarrow F_i$)

В строке 15 листа 5.4 приведена запись на языке FSP88 следующей системы дифференциальных уравнений:

$$F_1=y_2 \text{ и } F_2=(y_1|x-y_2)|x-y_1.$$

Она представлена словом DF.

Лист 5.5 иллюстрирует процесс решения при $y_{10}=0,099500833$, $y_{20}=0,49235$, $x_0=0,2$, $b=0,6$, $h=0,1$.

Лист 5.5

Пример решения системы дифференциальных уравнений

```

Ad=38440 Ar=59456
: DF 2 VY @ 1 VF ! 1 VY @ X@ / 2
VY @ - X@ / 1 VY @ - 2 VF ! ;
: DF FA! .099500833 0.49235 0.2
0.6 0.1 RK ;

```

```

x=0.3
Y1=0.14833527
Y2=0.48309507
x=0.4
Y1=0.19606167
Y2=0.47022677
x=0.5
Y1=0.2423236
Y2=0.45384862
x=0.6
Y1=0.28677658
Y2=0.434098

```

Набор слов листа 5.4 рассчитан на решение систем, содержащих от 1 до 10 дифференциальных уравнений. При этом массивы $K_{ii} - K_{ki}$ и y_i заданы содержащими до десяти компонент. Это уменьшает затраты времени на вычисление индексов массивов. При необходимости эти слова легко доработать под решение систем, содержащих большее количество уравнений. Для этого достаточно заменить числа 10, 20 и 30 на другие (например, 20, 40 и 60 при решении систем из $n \leq 20$ дифференциальных уравнений). Эту замену необходимо проделать для всех слов, в которых встречаются эти числа.

§ 5.4. Спектральный анализ

Как известно, любая периодическая функция $y(t)$ может быть разложена на отдельные гармонические составляющие — гармоники. Каждая гармоника характеризуется амплитудой и фазой. Сложение бесконечно большого числа гармоник восстанавливает исходную зависимость $y(t)$ точно, а при ограничении числа гармоник — приближенно. Вычисление амплитуд и фаз гармоник для заданной зависимости $y(t)$ составляет основную цель спектрального анализа. В системе FSP88 реализованы три метода спектрального анализа.

Первый метод относится к анализу спектра усеченных косинусоид (рис. 5.2). Он широко используется при анализе радиотехнических устройств (усилителей мощности, умножителей частоты и т. д.). В этом случае фазы всех гармоник равны нулю и вычисляются их относительные амплитуды — коэффициенты Берга. Они представлены аналитическими выражениями

$$\alpha_0 = y_0/y_M = (\sin \theta - \theta \cos \theta)/\pi (1 - \cos \theta),$$

$$\alpha_1 = y_{M1}/y_M = (\theta - \sin \theta \cos \theta)/\pi (1 - \cos \theta),$$

$$\alpha_n = \frac{y_{Mn}}{y_M} = \frac{2}{\pi} \cdot \frac{\sin(n\theta) \cos \theta - n \cos(n\theta) \sin \theta}{n(n^2 - 1)(1 - \cos \theta)}.$$

Таким образом, для вычисления заданного коэффициента α_i нужно задавать только два параметра — угол отсечки θ и номер гармоники i . При $i=0$ вычисляется относительная постоянная составляющая α_0 , при $i=1$ — относительная амплитуда первой гармоники α_1 , при $i=n \geq 2$ — относительные амплитуды высших гармоник

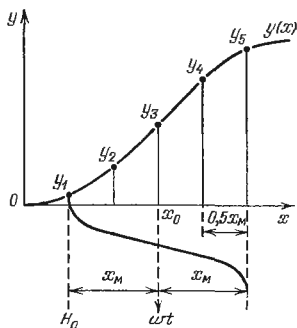
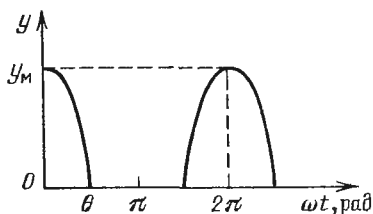


Рис. 5.2. Усеченная косинусоида (к анализу спектра методом Берга)

Рис. 5.3. К анализу спектра методом 5 ординат

α_n (везде y_{M1} — амплитуда гармоники, y_M — амплитуда усеченной косинусоиды).

Эти вычисления реализованы словами, включенными в лист 5.6 в строках 0—5.

Лист 5.6

Спектральный анализ методом Берга и методом 5 ординат

```

0 : a00 Q @ SIN Q @ DU COS * - C1 Q @ COS - / PI / ;
1 : a1 Q @ 2DU SIN SW COS * - C1 Q @ COS - / PI / ;
2 : an. N @ Q @ * DU SIN Q @ COS * SW COS N @ * Q @
SIN * ;
3 : an an. - 2* PI / N @ / N @ 2^ 1- / C1 Q @ COS -
/ ;
4 : b0 N ! 180/PI / Q ! N @ 0= IF a00 ELSE THEN ;
5 : BERG b0 N @ C1 = IF a1 ELSE THEN N @ C1 > IF an
ELSE THEN ;
6 : y0 A @ E @ + B @ D @ + 2* + c6 / ;
7 : ym1 E @ A @ - D @ + B @ - c3 / ;
8 : ym2 A @ E @ + 2/ C @ - 2/ ;
9 : ym3 E @ A @ - D @ B @ - 2* - c6 / ;
10 : ym4 A @ E @ + B @ D @ + c4 * - C @ c6 * + c12 /
;
11 : ordi E ! D ! C ! B ! A ! ;
12 : SORD ordi ym4 ym3 ym2 ym1 y0 ;
13 : Sord1 y0 ."Y0=" .P ym1 DU ."Ym1=" .P ym2 DU ."Y
m2=" .P ym3 ;
14 : Sord2 Sord1 DU ."Ym3=" .P ym4 DU ."Ym4=" .P ;
15 : SORD. ordi Sord2 2^ .SW 2^ + SW 2^ + SQR SW / ."
KF=" . ;

```

Слово A00 вычисляет α_0 , слово A1 вычисляет α_1 и слова AN и AN вычисляют α_n . Слова BERG0 и BERG служат для задания но-

мера гармоник, выбора расчетной формулы и вывода результата расчета. Из этих слов основным является слово

$$\text{BERG } \theta \ i \ _ _ _ \alpha_i(\theta)$$

Листинг 5.7 иллюстрирует спектральный анализ методом Берга — вычисляются $\alpha_0(30)$, $\alpha_1(30)$ и $\alpha_3(30)$.

Лист 5.7

Примеры вычисления коэффициентов Берга

```
30 0 BERG . ;
0.1105984
30 1 BERG . ;
0.21522321
30 3 BERG . ;
0.17146562
```

Второй метод — метод пяти ординат — используется для вычисления постоянной составляющей Y_0 и амплитуд четырех гармоник Y_1 — Y_4 для систем со слабой нелинейностью передаточной характеристики $y(x)$. Воздействие на систему считается гармоническим:

$$x(t) = x_M \cos \omega t + x_0,$$

где x_0 — постоянная составляющая воздействия, x_M — амплитуда. Фазовый сдвиг гармоник в этом случае также отсутствует. Рисунок 5.3 иллюстрирует суть метода.

Амплитуды гармоник вычисляются по формулам

$$Y_0 = [(y_1 + y_5) + 2(y_2 + y_4)]/6,$$

$$Y_{M1} = (y_1 - y_5 + y_2 - y_4)/3,$$

$$Y_{M2} = (y_1 - y_5 - 2y_3)/4,$$

$$Y_{M3} = [(y_1 - y_5) - 2(y_2 - y_4)]/6,$$

$$Y_{M4} = [(y_1 + y_5) - 4(y_2 + y_4)]/12.$$

Кроме того, вычисляется коэффициент гармоник

$$K_r = \sqrt{(Y_{M2})^2 + (Y_{M3})^2 + (Y_{M4})^2} / Y_{M1}.$$

Эти вычисления реализованы в строках 6—12 листа 5.7. Первые пять слов имеют имена, соответствующие обозначениям результата: y_0 , y_{m1} , y_{m2} , y_{m3} и y_{m4} . Слова 5Ord1 и 5Ord2 служат для задания пяти ординат и обращения к отмеченным выше словам. Основным является слово

```
5ORD y1 y2 y3 y4 y5 _ _ _ YM4 YM3 YM2 YM1 Y0
```

обеспечивающее вычисление Y_0 и Y_{M1} — Y_{M4} . Другое слово:

```
5ORD. y1 y2 y3 y4 y5 _ _ _
```

обеспечивает вывод результатов на индикацию, включая $K_r = K_r$. Пример вычислений дан ниже.

Пример вычислений методом пяти ординат

```

0.2 1 2.5 4.1 4.5 SORT. ;
Y0=2.48333333
Ym1=2.46666667
Ym2=-.0750000001
Ym3=-0.316666667
Ym4=-.0583333333
KГ=0.13403267

```

Более общий случай (третий метод) спектрального анализа предполагает, что функция $y(t)$ разбита на n равных промежутков времени и представлена $n+1$ отсчетами. Такую функцию, если она периодическая, можно представить рядом Фурье в одной из двух форм:

$$y(t) = \frac{a_0}{2} + \sum_{K=1}^{\infty} [a_K \cos(2\pi K f_1 t) + b_K \sin(2\pi K f_1 t)]$$

или

$$y(t) = \frac{a_0}{2} + \sum_{K=1}^{\infty} M_K \cos(2\pi K f_1 t + \varphi_K),$$

где $M_K = \sqrt{a_K^2 + b_K^2}$ и $\varphi_K = -\arccos(b_K/a_K)$ при $b_K \geq 0$ и $\varphi_K = \arccos(b_K/a_K)$ при $b_K < 0$,

$$a_K = \frac{2}{T} \int_0^T y(t) \sin(2\pi f_1 t) dt, \quad b_K = \frac{2}{T} \int_0^T y(t) \cos(2\pi f_1 t) dt.$$

Если функция непериодическая, но полностью определена на конечном интервале отрезка $[0, t_0]$, то она характеризуется комплексной спектральной плотностью

$$S(i\omega) = S_C(\omega) + jS_S(\omega) = S(\omega)e^{j\varphi(\omega)},$$

где

$$S(\omega) = \sqrt{[S_C(\omega)]^2 + [S_S(\omega)]^2},$$

$$\varphi(\omega) = -\arccos[S_C(\omega)/S_S(\omega)]$$

при $S_S(\omega) > 0$ и

$$\varphi(\omega) = \arccos[S_C(\omega)/S_S(\omega)]$$

при $S_S(\omega) < 0$. Здесь ($\omega = 2\pi f$)

$$S_C(\omega) = \int_0^{t_0} y(t) \cos(\omega t) dt, \quad S_S(\omega) = \int_0^{t_0} y(t) \sin(\omega t) dt.$$

Вычисляя интегралы методом прямоугольника при решетчатой аппроксимации (РША) функции $y(t)$, получим [10]

$$A_{K0} = \frac{S_C}{\Delta t} = \frac{a_K n}{2} = \sum_{i=0}^{n-1} y_i \cos(2\pi f_i \Delta t), \quad (5.6)$$

$$B_{K0} = \frac{S_S}{\Delta t} = \frac{b_K n}{2} = \sum_{i=0}^{n-1} y_i \sin(2\pi f_i \Delta t), \quad (5.7)$$

где $f = kf_1$, k — номер гармоники и f_1 — частота построения (для периодических колебаний). Таким образом, спектр периодических колебаний решетчатый (определен для дискретных $f = kf_1$). Для финитных колебаний f может принимать любые значения ($f > 0$), и их спектр является сплошным.

Версия FSP88 реализует последовательный спектральный анализ, вычисляя по заданному ряду y_i значения $S = S(\omega)$, $Q = \varphi(\omega)$, $A_{K0} = A_{K0}$, $B_{K0} = B_{K0}$, a_K и b_K . При этом y_i представляется в виде исходного вектора. Очевидно, что при $y_i = 0$ вычисление выражений под знаками сумм в (5.6) и (5.7) становится ненужными (члены дают 0). Поэтому предусмотрена возможность исключения задания нулевых отсчетов y_i в начале и конце вектора y_i . Вводятся только действующие отсчеты, начиная с начального y_a и до последнего y_m . Номера этих отсчетов обозначим как a и m (нулевые промежуточные отсчеты, однако, следует вводить).

Последовательный спектральный анализ на FSP88 реализуется рядом слов, представленных на листе 5.9.

Лист 5.9

Последовательный спектральный анализ

```

0 : SAI T ! O %! M %! A %! VIS VA0 ;
1 : sa1 F ! C0 DU S ! C ! PI F @ * T @ * DU P ! 2* W
! ;
2 : sa2 %1+ V@ DU Z@ COS * C +! Z@ SIN * S +! ;
3 : sa3 sa1 M %@ %1+ A %@ %I %I W @ * Z! %I sa2 %I ;
4 : sar C @ 2^ S @ 2^ + SQ R ! ;
5 : saf C @ R @ / ACS NEG PI T @ * F @ * - Q ! ;
6 : sak S @ Q< IF Q @ NEG Q ! ELSE THEN P @ SIN P @
/ L ! ;
7 : sac Q @ R @ T @ * L @ R @ * T @ * L @ 2^ R @ * T
@ * ;
8 : saq DU ."Q(rad)=" .P ."Q(deg)=" 180/PI * .P ;
9 : saab ."Ak0=" C ? CR ."Bk0=" S ? CR ;
10 : saab. ."ak=" C @ 2* O %@ / .P ."bk=" S @ 2* O %
@ / .P ;
11 : sap CR ."KJA S2=" .P ."CTA S1=" .P ."PWA S0=" .
P saq ;
12 : sas sa3 sar saf sak sac ;
13 : SAS. sas sap saab saab. ;
14
15

```

Из этих слов только три имеют самостоятельное значение.

Слово

SAT $y_a y_{a+1} \dots y_m a m n \Delta t$ — — —

(Spectral Analysis Input) вводит исходный массив y_i , номера начального a и конечного m действующих отсчетов, число интервалов разбиения n и время $\Delta t = T/n = t_0/n$.

Слово

SAS f — — — Q S0 S1 S2

выводит на вершину стека основные результаты спектрального анализа: спектральную плотность S2 при кусочно-линейной аппроксимации (КЛА) $y(t)$; S1 при ступенчатой аппроксимации (СТА) $y(t)$; S0 = $S(\omega)$ при решетчатой аппроксимации (РША) $y(t)$. При СТА S0 умножается на множитель $K_f = \sin(\pi f \Delta t) / (\pi f \Delta t)$, а при КЛА — на K_f^2 . Это позволяет повысить точность вычисления спектральной плотности для реальных видов $y(t)$. Выводится также фазовый угол $Q = \varphi$ (в радианах).

Слово

SAS. f — — —

используется, если результаты анализа должны быть выведены на печать (или индикацию). В этом случае слово SAS не используется (оно входит в состав словарной статьи слова SAS). На печать выводится фазовый угол Q в радианах и в градусах, а также коэффициенты Фурье a_k и b_k . Пример спектрального анализа дан ниже.

Лист 5.10

Пример спектрального анализа для прямоугольного импульса

```
1 1 1 1 1 1 1 1 0 7 32 1.25E-7
SAT ;
5000000 SAS. ;
КЛА S2=6.3253705E-7
СТА S1=6.3661978E-7
РША S0=6.4072667E-7
Q (rad) = -1.5707963
Q (deg) = -89.999998
Ak0=1.00000001
Bk0=5.0273396
ak = .062500009
bk = 0.31420672
```

В данном примере прямоугольный импульс задан восемью отсчетами $y_0 \div y_7 = 1$, $a=0$, $m=7$, $n=32$ и $\Delta t = 0,125 \cdot 10^{-6}$ с (т. е. импульс имеет длительность $8\Delta t = 1 \cdot 10^{-6}$ с и период колебаний $32 \cdot \Delta t = 4 \cdot 10^{-6}$ с). В примере вычислены его спектральные характеристики для $f = 5 \cdot 10^5$ Гц.

Процедура одновременного вычисления ряда гармоник в данном случае не предусмотрена. Но ее очень легко ввести, организовав циклическое изменение f и применив слово SAS или SAS. в цикле. Отметим,

что последовательный спектральный анализ удобен для вывода спектрограмм в виде графиков. При необходимости быстрого вычисления одновременно большого числа гармоник может использоваться быстрое преобразование Фурье (при необходимости соответствующую программу [10] можно ввести в состав FSP88).

§ 5.5. Операции с комплексными числами

Комплексные числа находят широкое применение в ряде научно-технических расчетов. Достаточно отметить использование их в электротехнических приложениях и в проектировании и анализе радиоэлектронных цепей. В связи с этим система FSP88 расширена введением операций над комплексными числами.

Арифметические операции с комплексными числами в алгебраической форме $Z_1 = x_1 + jy_1$ и $Z_2 = x_2 + jy_2$ выполняются по схеме

$$x_1 \ y_1 \ x_2 \ y_2 \ \text{---} \ \text{---} \ x \ y$$

где x и y — действительная и мнимая части результата $Z = x + jy$. Здесь $j = \sqrt{-1}$ — мнимая единица.

Для составляющих x и y результата имеются следующие соотношения:

$$\text{Сложение: } x = x_1 + x_2, \ y = y_1 + y_2.$$

$$\text{Вычитание: } x = x_1 - x_2, \ y = y_1 - y_2.$$

$$\text{Умножение: } x = x_1x_2 - y_1y_2, \ y = x_1y_2 + y_1x_2.$$

$$\text{Деление: } x = (x_1x_2 + y_1y_2) / (x_2^2 + y_2^2), \ y = (y_1x_2 - x_1y_2) / (x_2^2 + y_2^2).$$

Наряду с отмеченной алгебраической формой представления комплексных чисел $Z = x + jy$ используется показательная форма $Z = Me^{j\varphi}$. Преобразование числа Z из алгебраической формы в показательную выполняется по формулам

$$M = \sqrt{x^2 + y^2}, \quad \varphi = \arccos(x/M), \text{ если } y \geq 0,$$

$$\text{и } \varphi = -\arccos(x/M), \text{ если } y < 0.$$

Обратное преобразование (из показательной формы в алгебраическую) производится по формулам

$$x = M \cos \varphi, \quad y = M \sin \varphi.$$

В показательной форме удобно выполнять умножение и деление чисел $M_1e^{j\varphi_1}$ и $M_2e^{j\varphi_2}$. Результат (в виде числа $Me^{j\varphi}$) представляется в виде

$$M = M_1M_2, \quad \varphi = \varphi_1 + \varphi_2$$

для умножения и в виде

$$M = M_1/M_2, \quad \varphi = \varphi_1 - \varphi_2$$

для деления.

Основные операции с комплексными числами

```

0 : Z. SW . ." + j * ( " . . )" ;
1 : Z + ROT + 2ROT + SW ;
2 : Z - ROT SW - 2ROT - SW ;
3 : z i Z ! Y ! X ! W ! ;
4 : Z * z i W @ Y @ * X @ Z @ * - W @ Z @ * X @ Y @ * + ;
5 : M2 ^ 2 ^ SW 2 ^ + ;
6 : z / 1 z i Y @ Z @ M2 ^ M ! W @ Y @ * X @ Z @ * + M @ / ;
7 : Z / z / 1 X @ Y @ * W @ Z @ * - M @ / ;
8 : z q X @ Z @ / ACS Y @ 0 < IF NEG ELSE THEN ;
9 : ZAT Y ! X ! X @ Y @ M2 ^ SQ R Z ! Z @ z q ;
10 : ZT. SW ." M = " . . " Q = " . . " RAD " ;
11 : ZTA Y ! X ! Y @ COS X @ * Y @ SIN X @ * ;
12 : ZT * ROT + 2ROT * SW ;
13 : ZT / ROT SW - 2ROT / SW ;
14 : z y x Y ! X ! X @ 2 ^ Y @ 2 ^ ;
15 : Z2 ^ z y x - X @ Y @ * 2 * ;

```

Основные операции с комплексными числами (назначение слов см. ниже, слова начинаются с буквы Z), реализуются следующим набором слов:

Z.	xy — — —	(печать $x + jy$)
Z+	$x_1 y_1 x_2 y_2$ — — —	xy (для сложения)
Z-	$x_1 y_1 x_2 y_2$ — — —	xy (для вычитания)
Z*	$x_1 y_1 x_2 y_2$ — — —	xy (для умножения)
M↑2	xy — — —	$(x^2 + y^2) = M^2$ (вычисление M^2)
Z/	$x_1 y_1 x_2 y_2$ — — —	xy (для деления)
ZAT	xy — — —	$M\phi$ (преобразование из алгебраической формы в показательную)
ZT.	$M\phi$ — — —	(печать числа в показательной форме, т. е. M и $\phi \equiv Q$)
ZTA	$M \phi$ — — —	xy (преобразование из показательной формы в алгебраическую)
ZT*	$M_1 \phi_1 M_2 \phi_2$ — — —	$M\phi$ (умножение в показательной форме)
ZT/	$M_1 \phi_1 M_2 \phi_2$ — — —	$M\phi$ (деление в показательной форме)

Примеры выполнения арифметических операций над комплексными числами:

```

3 2 Z. ;
3 + j * (2)
1 2 3 4 Z + Z. ;
4 + j * (6)
1 2 3 4 Z - Z. ;
-2 + j * (-2)
1 2 3 4 Z * Z. ;
-5 + j * (10)
1 2 3 4 Z / Z. ;
0.44 + j * (.08)
3 2 ZAT ZT. ;
M = 3.6055513 Q = 0.5880026 RAD
3.6055513 0.5880026 ZTA Z. ;

```

3+j*(2)

3 0.4 2 0.6 ZT* ZT. ;

M=6 Q=1 RAD

3 90 2 45 ZT/ ZT. ;

M=1.5 Q=45 RAD

Приведенные примеры охватывают все рассмотренные выше операции.

Ниже показан пример комбинированных вычислений:

5 -3 3 2 Z* 5 3 Z/ 2 -4 Z/ 0.5 1

Z+ ZAT ZT. ;

M=1.8676818 Q=0.90146605 RAD

В данном случае вычисляется арифметическое выражение

$$Z_0 = \frac{(5-j3)(3+j2)}{(5+j3)(2-j4)} + (0,5+j1).$$

Результат вычислений представлен в показательной форме.

§ 5.6. Функции с комплексным аргументом

Чтобы набор операций с комплексными числами был функционально полным, в FSP88 добавлено вычисление значений элементарных функций комплексного аргумента. Для всех функций $F(Z)$, где $Z=x+jy$, соответствующее слово имеет в сетке значения x и y для аргумента Z и преобразует их в x_p и y_p для результата $Z_p=F(Z)=x_p+jy_p$:

$x_p \quad y_p$

Для всех функций имеются соответствующие формулы преобразования. Например, для функции Z^2 (слово Z2↑ в конце листа 5.14)

$$x_p = (x^2 - y^2), \quad y_p = 2xy.$$

На листах 5.12 и 5.13 представлено задание слов для вычисления функций комплексного аргумента.

Лист 5.12

Вычисление функций комплексного аргумента (пакет слов 1)

0 : Z-1^ zyx + Z! X@ Z@ / Y@ NEG Z@ / ;

1 : Z-2^ zyx + 2^ Z! X@ 2^ Y@ 2^ - Z@ / X@ Y@ * C2 N EG * Z@ / ;

2 : ZSQR zyx + SQR Z! X@ Z@ + 2/ SQR Z@ X@ - 2/ SQR ;

3 : ZEXP Y! EXP DU Y@ COS * SW Y@ SIN * ;

4 : ZLN zyx + LN 2/ Y@ X@ / ATN ;

5 : ZSIN Y! X! X@ SIN Y@ HCS * X@ COS Y@ HSN * ;

6 : ZCOS Y! X! X@ COS Y@ HCS * X@ SIN Y@ HSN * NEG ;

7 : ZTAN 2* Y! 2* X! X@ COS Y@ HCS + Z! X@ SIN Z@ / Y@ HSN Z@ / ;

8 : zab Y! X! X@ 1+ 2^ Y@ 2^ + SQR DU X@ 1- 2^ Y@ 2^ + ;

9 : ZAB zab SQR DU ROT + 2/ 2ROT - 2/ ;

```

10 : ZASN ZAB ASN SW DU 2^ 1- SQR + LN ;
11 : ZACS ZAB ACS SW DU 2^ 1- SQR + LN NEG ;
12 : zat Y! DU 2^ X! 2* 1 X@ - Y@ 2^ - / ATN 2/ ;
13 : ZATN zat X@ Y@ 1+ 2^ + X@ Y@ 1- 2^ + / LN 4 / ;
14 : ZHSN Y! X! X@ HSN Y@ COS * X@ HCS Y@ SIN * ;
15 : ZHCS Y! X! X@ HCS Y@ COS * X@ HSN Y@ SIN * ;

```

Лист 5.13

Вычисление функций комплексного аргумента (пакет слов 2)

```

0 : zht DU + Y! DU + X! X@ HCS Y@ COS + Z! ;
1 : ZHTN zht X@ HSN Z@ / Y@ SIN Z@ / ;
2 : ZAHS NEG SW ZASN SW NEG ;
3 : ZAHC ZACS NEG SW ;
4 : ZAHT NEG SW ZATN SW NEG ;
5 : I1 H ! Z! Y! X! ;
6 : Y1 Z@ - Y@ X@ - * H @ / X@ + ;
7 : I2 H ! Z! C ! Y! X! ;
8 : PX Z@ - H @ / P ! ;
9 : ss P @ DU 1- * 2/ X@ * 1 P @ 2^ - Y@ * P @ DU 1+
  * 2/ C @ * ;
10 : Y2 PX ss + + ;
11 : fjn DU SIN Z@ * SW W @ * - COS ;
12 : Jn W ! Z! C0 PI EPS ' fjn ffse PI / ;
13 : fin DU COS Z@ * EXP SW W @ * COS * ;
14 : In W ! Z! C0 PI EPS ' fin ffse PI / ;
15

```

Значения x_p и y_p для слов, представленных выше, приведены в табл. 5.1.

Ниже приведены распечатки контрольных примеров для вычисления алгебраических функций комплексного аргумента:

```

3 2 Z2^ Z. ;
5+j*(12)
3 2 Z-1^ Z. ;
0.23076923+j*(-0.15384615)
3 2 Z-2^ Z. ;
.029585799+j*(-.071005917)
3 2 ZEXP Z. ;
-8.3585326+j*(18.263727)
3 2 ZLN Z. ;
1.2824747+j*(0.5880026)
3 2 ZSQR Z. ;
1.817354+j*(0.55025052)

```

тригонометрических и обратных тригонометрических функций:

```

3 2 ZSIN Z. ;
0.53092109+j*(-3.5905646)
3 2 ZCOS Z. ;
-3.7245455+j*(-0.51182257)
3 2 ZTAN Z. ;
-.009884375+j*(0.96538588)
3 2 ZASN Z. ;
0.9646585+j*(1.9686379)
3 2 ZACS Z. ;
0.60613782+j*(-1.9686379)
0.3 0.2 ZATN Z. ;
0.30187467+j*(0.18499462)

```

а также гиперболических и обратных гиперболических:

```
3 2 ZHSN Z. ;  
-4.168907+j*(9.1544991)  
3 2 ZHCS Z. ;  
-4.1896257+j*(9.1092279)  
3 2 ZHTN Z. ;  
1.0032386+j*(-.0037640256)  
3 2 ZABS Z. ;  
1.983387+j*(0.57065278)  
3 2 ZANC Z. ;  
1.9686379+j*(0.60613782)  
0.3 0.2 ZANT Z. ;  
0.29574992+j*(0.21547449)
```

Отметим, что для функций \sqrt{Z} и $\operatorname{arsh} Z$ значения x_p и y_p могут иметь любой знак, но выдаются они только одного знака.

Следующий пример иллюстрирует комбинированные действия с функциями комплексного аргумента (вычисляется $F(Z) = \ln((1+j2) \times (3+j4)^2)$):

```
1 2 3 4 Z2^ Z* ZLN Z. ;  
4.0235948+j*(-0.1798535)
```

§ 5.7. Линейная и квадратичная интерполяция

В практических расчетах часто приходится пользоваться различными таблицами. Обширный набор функций (в том числе специальных — см. § 5.8), включенных в состав расширенной версии системы FSP88, резко уменьшает потребность в применении табличных данных. Тем не менее в состав FSP88 включены слова, обеспечивающие проведение (наиболее распространенных) линейной и квадратичной интерполяции при равномерном расположении узлов (см. лист 5.13 строки 5—9).

Линейная интерполяция выполняется при задании функции $y(x)$ двумя узлами: (x_0, y_0) и (x_1, y_1) и соответствует подмене $y(x)$ интерполяционным полиномом первой степени

$$y(x) = y_0 + (x - x_0)(y_1 - y_0)/h,$$

где h — шаг интерполяции, т. е. $h = x_1 - x_0$.

Линейная интерполяция реализуется двумя словами: П1 и Y1. Слово П1 (*input* — ввод при степени полинома $n=1$) служит для ввода исходных данных:

```
Y0 Y1 X0 h _ _ _
```

Слово

```
Y1 x _ _ _ y(x)
```

заменяет x на вершине стека значением $y(x)$.

Значения x_p и y_p для $F(Z) = x_p + jy_p$

Функция $F(Z)$	Слово	x_p (кроме слова Zab)	y_p (кроме слова Zab)
Z^2	Z2↑	$x^2 - y^2$	$2xy$
Z^{-1}	Z-1↑	$x/(x^2 + y^2)$	$-y/(x^2 + y^2)$
Z^{-2}	Z-2↑	$(x^2 - y^2)/(x^2 + y^2)^2$	$-2xy/(x^2 + y^2)^2$
\sqrt{Z}	ZSQR	$\sqrt{\frac{x + \sqrt{x^2 + y^2}}{2}}$	$\sqrt{\frac{-x + \sqrt{x^2 + y^2}}{2}}$
EXP(Z)	ZEXP	$e^x \cos y$	$e^x \sin y$
ln Z	ZLN	$0,5 \ln(x^2 + y^2)$	$\text{arctg}(y/x)$
sin Z	ZSIN	$\sin x \cdot \text{ch} y$	$\cos x \cdot \text{sh} y$
cos Z	ZCOS	$\cos x \cdot \text{ch} y$	$-\sin x \cdot \text{sh} y$
tg Z	ZTAN	$\sin(2x) / (\cos(2x) + \text{ch}(2y))$	$\text{sh}(2y) / (\cos(2x) + \text{ch}(2y))$
—	Zab	$A = (\sqrt{(x+1)^2 + y^2} + \sqrt{(x-1)^2 + y^2}) / 2$	$B = (\sqrt{(x+1)^2 + y^2} - \sqrt{(x-1)^2 + y^2}) / 2$
arcsin Z	ZASN	$\arcsin B$	$\ln(A + \sqrt{A^2 - 1})$
arccos Z	ZACS	$\arccos B$	$\ln(A - \sqrt{A^2 - 1})$
arctg Z	ZATN	$0,5 \text{arctg} \frac{2x}{1 - x^2 - y^2}$	$\frac{1}{4} \ln \frac{x^2 + (y+1)^2}{x^2 + (y-1)^2}$
sh Z	ZHSN	$\text{sh} x \cdot \cos y$	$\text{ch} x \cdot \sin y$
ch Z	ZHCS	$\text{ch} x \cdot \cos y$	$\text{sh} x \cdot \sin y$
th Z	ZHTN	$\frac{\text{sh}(2x)}{\text{ch}(2x) + \cos(2y)}$	$\frac{\sin(2y)}{\text{ch}(2x) + \cos(2y)}$
arsh Z	ZAHS		Соответствует $-j \arcsin(jZ)$
arch Z	ZAHC		Соответствует $\pm j \arccos Z$
arth Z	ZAHT		Соответствует $-j \text{arctg}(jZ)$

Квадратичная интерполяция ($n=2$) реализуется тремя словами: I2, PX и Y2. Слово I2 служит для ввода исходных данных

$$y_{-1} \ y_0 \ y_{+1} \ x_0 \ h \ _ \ _ \ _$$

где y_{-1} , y_0 и y_{+1} — ординаты левого, центрального и правого узлов интерполяции.

Слово PX вычисляет $p = (x - x_0) / h$, а слово

$$Y2 \ x \ _ \ _ \ _ \ y(x)$$

заменяет x на вершине стека значением $y(x)$, вычисляя его по прямой интерполяционной формуле Лагранжа:

$$y(x) = y_{-1} \frac{p(p-1)}{2} + y_0(1-p^2) + y_{+1} \frac{p(p+1)}{2}.$$

Примеры проведения интерполяции представлены ниже:

I2 20 1 1 I1 ;

1.5 Y1 . ;

15

1 4 9 2 1 I2 ;

1.5 Y2 . ;

2.25

2.5 Y2 . ;

6.25

Отметим, что с помощью описанного комплекта слов можно выполнять линейную и квадратичную экстраполяцию. Однако погрешность экстраполяции обычно существенно выше, чем у интерполяции. Для интерполяции $y(x)$, заданной рядом произвольно расположенных узлов, можно использовать слова, описанные в § 5.9.

§ 5.8. Вычисление специальных функций

Специальные математические функции широко используются в научно-технических расчетах в самых различных отраслях науки и техники. Большинство специальных функций табулировано [15]. Однако применение таблиц крайне неудобно из-за долгого поиска узловых точек нужной функции и необходимости в общем случае пользования аппаратом интерполяции. Поэтому целесообразно иметь набор слов, реализующих вычисления наиболее распространенных специальных функций. В FSP88 для вычисления специальных функций используется численное интегрирование методом Гаусса с заданной погрешностью $E = EPS = 1 \cdot 10^{-8}$.

В строках 11—14 листа 5.13 задано вычисление функций Бесселя:

$$J_n(x) = \frac{1}{\pi} \int_0^\pi \cos(x \sin \theta - n\theta) d\theta, \quad I_n(x) = \frac{1}{\pi} \int_0^\pi e^{x \cos \theta} \cos(n\theta) d\theta.$$

Слова fjp и fip вычисляют значения подынтегральных функций.
Основными являются слова

Jп x n — — — $J_n(x)$

Iп x n — — — $I_n(x)$

Они по выведенным x и n оставляют на вершине стека значения $J_n(x)$ и $I_n(x)$.

На следующих двух листах задан ряд слов для вычисления наиболее распространенных специальных функций (напоминаем, что функции erf(x) и si(x) были реализованы ранее).

Лист 5.14

Вычисление специальных функций (пакет слов 1)

```
0 : sss DU C0 SW EPS ;
1 : g1 DU NEG EXP C1 SW - SW / ;
2 : E1 sss ' g1 ffse CE -SW LN - ;
3 : g2 DU EXP 1- SW / ;
4 : E1 sss ' g2 ffse CE + SW LN + ;
5 : g3 DU COS 1- SW / ;
6 : C1 sss ' g3 ffse CE + SW LN + ;
7 : FCOS 2^ PI/2 * COS ;
8 : C(X) C0 SW EPS ' FCOS ffse ;
9 : FSIN 2^ PI/2 * SIN ;
10 : S(X) C0 SW EPS ' FSIN ffse ;
11 : g4 DU LN SW 1- / ;
12 : DLN C1 SW EPS ' g4 ffse NEG ;
13 : xg DU X! ABS B ! C1 D ! B @ INT C0 B @ C1 < ;
14 : xc xg IF DR DR ELSE %[ D @ B @ * D ! C1 NEG B +
! %] THEN ;
15
```

Лист 5.15

Вычисление специальных функций (пакет слов 2)

```
0 : fg1 B @ .035868343 * 0.1935278 - B @ * 0.4821993
9 + B @ * ;
1 : fg2 fg1 0.75670408 - B @ * 0.91820606 + B @ * 0.
8970569 - ;
2 : fg fg2 B @ * 0.98820589 + B @ * 0.57719165 - B @
* 1+ ;
3 : g0 fg D @ * X@ / ;
4 : G(X) xc X@ 0 > IF g0 ELSE PI DU X@ * SIN / D @ /
fg / THEN ;
5 : fm SIN 2^ W @ * C1 SW - ;
6 : fmf fm SQR C1 SW / ;
7 : fme fm SQR ;
8 : FQM W ! C0 SW EPS ' fmf ffse ;
9 : EQM W ! C0 SW EPS ' fme ffse ;
10 : vxi cn1 %1+ %[ R @ X@ * DU R ! L %@ %I M! %] ;
11 : vx DU L %! %N@ %1+ SW %- ROLL X! C1 R ! ;
12 : VXI VN VA0 cn1 %[ C1 %I C1 M! %I vx vxi %] ;
13 : VA0 %N@ C0 %[ ."A" %I . ."=" a0V %I %5* %+ ? CR
%] ;
14 : VVI VA0 MI VBI MV* A %@ c38498 %! VA0 ;
15
```

Ограничимся рассмотрением основных слов этих листов (вспомогательные слова, начинающиеся со строчных букв, в конечной реализации FSP88 скрываются при чистке словаря).

Слово

$$E1 \ x \ _ \ _ \ _ \ E_1(x) = \int_0^x \frac{1-e^{-t}}{t} dt - \ln x - \gamma$$

заменяет аргумент x на вершине стека значением интегральной показательной функции $E_1(x)$. Подынтегральная функция

$$(1-e^{-t})/t$$

вычисляется словом gl . Заметим, что обычно

$$E_1(x) = \int_x^{\infty} \frac{e^{-t}}{t} dt.$$

Однако вычисление интеграла с бесконечным верхним пределом сложнее, чем интеграла, входящего в выражение, использованное при реализации слова $E1$.

Слово

$$Ei \ x \ _ \ _ \ _ \ Ei(x) = \int_0^x \frac{e^t-1}{t} dt + \ln x + \gamma$$

заменяет значение x на вершине стека значением интегральной показательной функции $Ei(x)$. Здесь также использовано выражение для интеграла с конечными верхними пределами.

В приведенных формулах γ — константа Эйлера (в FSP88 она обозначена cE).

Слово

$$FCOS \ t \ _ \ _ \ _ \ \cos(\pi t^2/2)$$

заменяет t на вершине стека значением функции $\cos(\pi t^2/2)$.

Слово

$$C(X) \ x \ _ \ _ \ _ \ C(x) = \int_0^x \cos(\pi t^2/2)$$

заменяет x на вершине стека значением интеграла Френеля $C(x)$

Слово

$$FSIN \ t \ _ \ _ \ _ \ \sin(\pi t^2/2)$$

заменяет t на вершине стека значением функции $\sin(\pi t^2/2)$.

Слово

$$S(X) \quad x \quad \dots \quad S(x) = \int_0^x \sin(\pi t^2/2) dt$$

заменяет x на вершине стека значением интеграла Френеля $S(x)$.

Слово

$$DLN \quad x \quad \dots \quad - \int_0^x \frac{\ln t}{t-1} dt$$

заменяет x на вершине стека значением функции дилогарифма.

Слово

$$G(X) \quad x \quad \dots \quad \Gamma(x)$$

заменяет x на вершине стека значением гамма-функции $\Gamma(x)$. При этом для вычисления $\Gamma(x)$ используется следующий алгоритм.

1. Если $0 \leq x \leq 1$, то для $B = x + 1$ вычисляем $F = \Gamma(B) = (\dots (0,035868343B - 0,1935278B + 0,48219939)B - 0,75670408B + 0,91820606B - 0,8970569)B + 0,98820589B + 1$.

2. Если $x > 1$, то из $B = |x|$ вычитаем 1 до тех пор, пока $|x|$ не попадает в интервал $0 \leq x < 1$. При этом подсчитываем произведение $D = B(B-1)(B-2) \dots (B-K)$, где K — число вычитаний 1 из B . Тогда $\Gamma(x) = DF(Y)/x$, где $F(Y)$ (при $Y = B - K$) — функция, вычисляемая в п. 1.

3. Если $x < 0$, берем $x = |x|$ и вычисляем Y и $F(Y)$ по пп. 1 и 2, затем находим

$$\Gamma(x) = \frac{\pi}{\sin \pi x} \frac{1}{DF}$$

Слово

$$FQM \quad Q \quad m \quad \dots \quad F(Q|m) = \int_0^Q (1 - m \sin^2 Q)^{-1/2} dQ$$

заменяет Q и m на вершине стека значением неполного эллиптического интеграла $F(Q|m)$.

Слово

$$EQM \quad Q \quad m \quad \dots \quad E(Q|m) = \int_0^Q (1 - m \sin^2 Q)^{1/2} dQ$$

заменяет Q и m на вершине стека значением неполного эллиптического интеграла $E(Q|m)$.

Контрольные примеры для вычисления описанных в этом параграфе специальных функций даны на листе 5.16.

Лист 5.16

Примеры вычислений специальных функций

3 2 Jn . ;
0.48609125
2 1 In . ;
1.5906369
0.5 E1 . ;
0.55977359
0.5 Ei . ;
0.45421991
0.2 C(X) . ;
0.19992106
2 S(X) . ;
0.34341568
0.1 DLN . ;
1.2997147
-3.2 G(X) . ;
0.68905644
1.2 0.2 FQM . ;
1.2473374
1.2 0.2 EQM . ;
1.1555454

* При необходимости в FSP88 легко ввести и другие слова, реализующие вычисления других специальных функций или более быстрое вычисление описанных функций по другим алгоритмам (разложением в ряд или применением аппроксимаций).

§ 5.9. Полиномиальная аппроксимация

В ряде случаев возникает необходимость в аналитическом представлении таблично заданной функции $y_i(x_i)$ или сложной аналитической функции, вычисляемой для ряда значений x_i . Эта задача — аппроксимация функций — часто решается путем замены функции степенным полиномом $y(x)$:

$$y(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n.$$

Если $y_i(x_i)$ задана $n+1$ узлами, то можно записать следующую систему уравнений:

$$a_0 + a_1x_0 + a_2x_0^2 + \dots + a_nx_0^n = y_0,$$

$$a_0 + a_1x_1 + a_2x_1^2 + \dots + a_nx_1^n = y_1,$$

$$\dots$$

$$a_0 + a_1x_n + a_2x_n^2 + \dots + a_nx_n^n = y_n.$$

Если коэффициенты a_0, a_1, \dots, a_n полинома рассматривать как неизвестные, то данная система будет системой линейных уравнений. Для ее решения можно использовать реализованный в FSP88 метод

обращения матрицы коэффициентов системы и умножение ее на вектор свободных членов.

Этот метод (называющийся методом выбранных точек) реализуется следующими словами: VXI , VX , VXI , VAO и VYI (см. лист 5.5). Отметим функции основных слов. Слово VXI используется для ввода абсцисс функции:

$$x_0 \ x_1 \ x_2 \ \dots \ x_n \ \text{VXI}$$

Слово VAO выводит вектор коэффициентов a_i . Слово VYI используется для ввода ординат функции:

$$y_0 \ y_1 \ y_2 \ \dots \ y_n \ \text{VYI}$$

и обеспечивает автоматический переход к вычислению коэффициентов a_i аппроксимирующего полинома. Таким образом, слово VAO исполняется внутри словарной статьи слова VYI .

Аппроксимация заключается в вводе абсцисс (фиксируется словом VXI) и ординат y_i (фиксируется словом VYI). После этого вычисляются и индицируются значения a_i , причем они становятся хранимым в ОЗУ вектором. С помощью слова $VPOL$

$$x \ \text{---} \ \text{---} \ \text{---} \ y(x)$$

для заданного x можно вычислить $y(x)$. Таким образом, метод обеспечивает наряду с аппроксимацией полиномиальную интерполяцию функций при произвольном и любом числе узлов.

Для иллюстрации практического применения аппроксимации рассмотрим аппроксимацию экспериментальной зависимости относительной видности K_λ дневного зрения от длины волны света $\lambda(\text{А})$, показанной на рис. 5.4 и таблице.

$\lambda, \text{А} \cdot 10^3$	K_λ
4,4	0,023
4,8	0,139
5,2	0,710
5,6	0,995
6,6	0,631
6,4	0,175
6,8	0,017

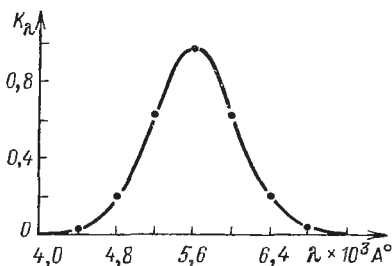


Рис. 5.4. Зависимость K_λ от длины волны λ

Порядок проведения аппроксимации и вычисления K_λ в промежутках между узлами представлен ниже:

4.4 4.8 5.2 5.6 6 6.4 6.8 VXI .0
23 0.139 0.71 0.995 0.631 0.175
.017 VYI ;
A0=-14055.247
A1=15867.763
A2=-7404.3745
A3=1827.7199
A4=-251.7088
A5=18.339417
A6=-0.55241377

5 VPOL . ;
0.40685272
4.75 VPOL . ;
.092971802

Хотя теоретически метод выбранных точек применим при любом числе узлов $n+1$, на практике оно ограничено значениями не более 8—10. С ростом степени полинома n быстро возрастает погрешность вычисления $y(x)$. Это, в частности, связано с погрешностями усечения результатов при решении системы линейных уравнений.

ЦЕЛОЧИСЛЕННЫЕ ВЕРСИИ ФОРТ-СИСТЕМ ПРОГРАММИРОВАНИЯ (FORTH-79, fig-FORTH и FORTH-83)

§ 6.1. Числа и арифметические операции с ними

Наиболее распространенные версии языка Форт, такие как FORTH-79, fig-FORTH и FORTH-83, в стандартном варианте являются целочисленными. Это означает, что все арифметические операции у них выполняются только над целыми числами. На первый взгляд последнее может показаться существенным недостатком. И это так, если рассматривать Форт как язык, готовый к немедленному применению в математических и научно-технических расчетах.

Однако форт-системы программирования в первую очередь созданы для разработки эффективного программного обеспечения самого разнообразного характера. Хотя это и покажется неудобным для научного работника и инженера, но упрямая статистика показывает, что доля научных и инженерных расчетов в совокупном программном продукте составляет не более 10 %. Большая часть создаваемых программ относится к операциям с текстами и графикой, которые оперируют с целыми числами. Поэтому исключение операций над числами с плавающей запятой из ряда версий Форты вполне оправданно. Там, где такие операции часто необходимы, нужно просто использовать специальные версии языка Форт, например описанную выше систему программирования FSP88 или другие расширенные версии Форты с такими операциями.

Между различными целочисленными версиями языка Форт есть ряд небольших отличий. Они могут породить определенные трудности при переводе программ из одной версии в другую. К тому же на разных ЭВМ нередко преимущественно реализованы разные версии языка Форт. Так, в отечественных ПЭВМ Искра-226 распространена операционная система ФОС на базе языка FORTH-79, в машинах с идеологией серии СМ (например, ДВК-2М) — версия fig-FORTH, а в ЕС ЭВМ внедрена версия FORTH-83. В то же время между этими версиями все же больше общего, чем различий. Исходя из этого, последующее описание дается параллельно для трех наиболее важных и признанных на мировом уровне версий языка Форт: FORTH-79, fig-FORTH и FORTH-83 [20, 21, 23, 25—28, 30, 32—37, 39, 40]. В необходимых случаях отмечается, как перейти от одной версии языка к другой.

Целочисленные версии языка Форт обычно оперируют с пятью типами десятичных чисел.

1. Целые 8-битовые числа (байты) без знака (*b*).

2. Целые 16-битовые числа со знаком + или -, принимающие значения от -32,768 до 32,767. Здесь запятая является формальным признаком этих чисел, не имеющим никакого отношения к обычному десятичному представлению дробных чисел. Такие числа обозначаются как *n* и занимают в ОЗУ 2 байта.

3. Целые 32-битовые числа двойной разрядности со знаком + или -, принимающие значения от -2,147,483,648 до 2,147,483,647. Эти числа обозначаются как *d* и занимают в ОЗУ 4 байта.

4. Целые 16-битовые числа без знака (т. е. всегда положительные). Эти числа лежат в пределах от 0 до 65,535 и обозначаются как *u* (от слова *unsigned* — беззнаковый).

5. Целые 32-битовые числа двойной разрядности без знака (т. е. всегда положительные). Их значения лежат в пределах от 0 до 4,294,967,295 и обозначаются как *ud*.

В арифметических операциях действия над числами одинарной разрядности *n* особо не выделяются. При операциях над числами двойной разрядности со знаком в слова вводится буква *D*, а при операциях над такими числами без знака — буква *U*. Поскольку адреса, используемые в форт-системах программирования, лежат в пределах от 0 до 65535, для их вызова следует использовать слово *U*. (вывод чисел *u*).

В табл. 6.1 приведены базовые слова для указанных выше трех версий языка Форт, используемые для проведения арифметических операций. С учетом знакомства с форт-системой FSP88 (а для лиц, освоивших основы программирования, то и без этого знакомства) характер выполняемых операций достаточно очевиден. Для иллюстрации большинства из них ниже дан листинг с рядом демонстрационных примеров по основным из таких операций.

Лист 6.1

Арифметические операции

```
6 LIST
```

```
SCR # 6
```

```
0 ( INTEGER ARITHMETICAL OPERATIONS )
```

```
1 : DEMO ." 2 3 + -> " 2 3 + . CR
```

```
2 ." 2 3 - -> " 2 3 - . CR
```

```
3 ." 2 3 * -> " 2 3 * . CR
```

```
4 ." 7 2 / -> " 7 2 / . CR
```

```
5 ." 4 2+ -> " 4 2+ . CR
```

```
6 ." 4 1+ -> " 4 1+ . CR
```

```
7 ." -5 ABS -> " -5 ABS . CR
```

```
8 ." 2 5 MAX -> " 2 5 MAX . CR
```

```
9 ." 2 5 MIN -> " 2 5 MIN . CR
```

```
10 ." 24 MINUS -> " 24 MINUS . CR
```

```
11 ." 6 2 MOD -> " 6 2 MOD . CR
```

```
12 ." 9 6 2 */ -> " 9 6 2 */ . CR
```

```

13 ." 9 6 2 */MOD -> " 9 6 2 */MOD . SPACE . CR
14 ." 10 6 4 +- -> " 9 6 4 +- . CR ;
15
ok
DEMO 2 3 + -> 5
2 3 - -> -1
2 3 * -> 6
7 2 / -> 3
4 2+ -> 6
4 1+ -> 5
-5 ABS -> 5
2 5 MAX -> 5
2 5 MIN -> 2
24 MINUS -> -24
6 2 MOD -> 0
9 6 2 */ -> 27
9 6 2 */MOD -> 27 0
10 6 4 +- -> 6
ok

```

Отметим, что при начальном пуске форт-систем по умолчанию задаются операции с целыми десятичными числами, имеющими основание (*base*) $b=10$. Контроль на потерю значимости чисел не производится. Знаковым является старший разряд (0 соответствует знаку +, 1 — знаку —).

Если результат вычислений выходит за допустимые пределы задания чисел, могут наблюдаться некорректные результаты. Чтобы

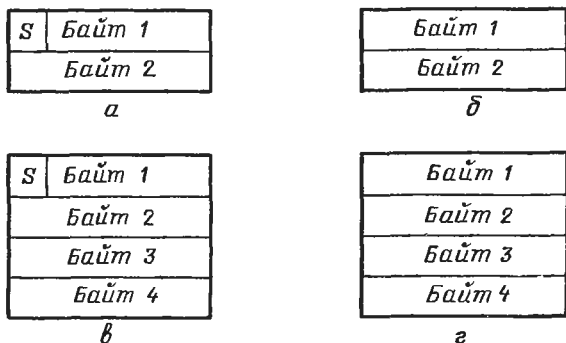


Рис. 6.1. Структура ячеек ОЗУ для хранения чисел различного типа: одинарной разрядности со знаком (а), одинарной разрядности без знака (б), двойной разрядности со знаком (в) и двойной разрядности без знака (г). Буквой S обозначен бит знака

понять суть искажений, следует помнить, как задаются числа в ОЗУ ПЭВМ. Многие ПЭВМ имеют элементарные ячейки ОЗУ, хранящие по 8 бит, т. е. содержат 1 байт (число от 0 до 255). На рис. 6.1 представлена структура ячеек ОЗУ для чисел различного типа. Буквой S (от слова *signum* — знак) обозначен бит, несущий информацию о знаке. Таким образом, числа одинарной разрядности без знака

Арифметические операции

FORTH			Стек	Действие слова
79	fig	83		
D.	D.	D.	$n \quad _ _ _$	Выводит на печать числа одинарной разрядности из вершины стека
U.	Нет	U.	$d \quad _ _ _$	Выводит на печать числа двойной разрядности из вершины стека
U.	Нет	U.	$u \quad _ _ _$	Выводит на печать числа u из вершины стека
+	+	+	$n1 \ n2 \ _ _ _ \ n1+n2$	Складывает $n1$ и $n2$ одинарной разрядности
1+	1+	1+	$n \quad _ _ _ \ n+1$	Прибавляет 1 к n
2+	2+	2+	$n \quad _ _ _ \ n+2$	Прибавляет 2 к n
D+	+	+	$d1 \ d2 \ _ _ _ \ d1+d2$	Складывает $d1$ и $d2$ при удвоенной разрядности чисел
-	-	-	$n1 \ n2 \ _ _ _ \ n1-n2$	Вычитает $n2$ из $n1$
1-	Нет	1-	$n \quad _ _ _ \ n-1$	Вычитает 1 из n
2-	Нет	2-	$n \quad _ _ _ \ n-2$	Вычитает 2 из n
D-	Нет	D-	$d1 \ d2 \ _ _ _ \ d1-d2$	Вычитает $d2$ из $d1$ при удвоенной разрядности чисел
*	*	*	$n1 \ n2 \ _ _ _ \ n1 \cdot n2$	Умножает $n1$ на $n2$
U*	U*	U*	$un1 \ un2 \ _ _ _ \ ud$	Умножает $un1$ на $un2$ с результатом ud двойной разрядности
Нет	Нет	2*	$n \quad _ _ _ \ 2n$	Умножает n на 2
Нет	M*	Нет	$n1 \ n2 \ _ _ _ \ d$	Умножает $n1$ на $n2$ с результатом $d=n1 \cdot n2$ двойной разрядности
Нет	Нет	M*	$u1 \ u2 \ _ _ _ \ ud$	Умножает $u1$ на $u2$ с результатом ud двойной разрядности

Таблица 6.1 (продолжение)

FORTH			Стек	Действие слова
79	fig	83		
/ MOD /MOD	/ MOD /MOD	/ MOD /MOD	$n1\ n2\ ____\ n$ $n1\ n2\ ____\ r$ $n1\ n2\ ____\ r\ n$	Делит $n1$ на $n2$ Получает остаток r от деления $n1$ на $n2$ Делит $n1$ на $n2$ с получением целой части результата n и остатка r
Нет	U/	Нет	$ud\ un\ ____\ r\ n$	Делит ud на un с получением целой части результата n и остатка r
Нет	M/MOD	Нет	$ud\ un\ ____\ r\ d$	Делит ud на un с получением целой части результата d и остатка r
Нет	Нет	2/	$n\ ____\ n/2$	Делит n на 2
*/ *MOD Нет	*/ */MOD Нет	*/ */MOD UM/MOD	$n1\ n2\ n3\ ____\ n$ $n1\ n2\ n3\ ____\ r\ n$ $ud\ u1\ ____\ u2\ u3$	Вычисляет $n1 \cdot n2 / n3$ Получает целую часть n и остаток r от $n1 \cdot n2 / n3$ Делит ud на $u1$, результат: $u2$ — остаток, $u3$ — нижняя граница частного
NEGATE DNEGATE	MINUS DMINUS	NEGATE DMINUS	$n\ ____\ -n$ $d\ ____\ -d$	Меняет знак n Меняет знак d
Нет Нет	+ - D + -	Нет Нет	$n1\ n2\ ____\ n$ $d1\ n\ ____\ d$	Оставляет первое число $n1$ со знаком второго $n2$ Оставляет первое число $d1$ со знаком второго n
Нет	S -> D	S > d	$n\ ____\ d$	Преобразует число n одинарной разрядности в число d двойной разрядности
ABS DABS	ABS DABS	ABS DABS	$n\ ____\ n $ $d\ ____\ d $	Заменяет n на модуль $ n $ Заменяет d на модуль $ d $

FORTH			Стек	Действие слова
79	fig	83		
MAX MIN DMAX DMIN	MAX MIN Her Her	MAX MIN DMAX DMIN	$n1\ n2\ \dots\ n_{\max}$ $n1\ n2\ \dots\ n_{\min}$ $d1\ d2\ \dots\ d_{\max}$ $d1\ d2\ \dots\ d_{\min}$	Оставляет максимальное число из двух чисел $n1$ и $n2$ Оставляет минимальное число из двух чисел $n1$ и $n2$ Оставляет максимальное число из двух чисел $d1$ и $d2$ Оставляет минимальное число из двух чисел $d1$ и $d2$
Her	Her	D2*	$d\ \dots\ 2d$	Умножение на 2 числа d двойной разрядности
BASE DECIMAL HEX	BASE DECIMAL HEX	BASE DECIMAL HEX	$\dots\ A$ \dots \dots	Перемещает на вершину стека адрес системной переменной, задающей основание чисел Задаёт основание чисел 10 Задаёт основание чисел 16
Her Her	Her Her	BINARY OCTAL	\dots \dots	Задаёт основание чисел 2 Задаёт основание чисел 8
TOGGLE	TOGGLE	Her	$A\ b\ \dots$	Складывает по модулю 2 содержимое ячейки ОЗУ, имеющей адрес A со значением b ; результат оставляет в этой ячейке

занимают в ОЗУ 16 бит, и их предельное значение $2^{16} - 1 = 65535$; числа со знаком занимают 15 бит, и их предельное значение $2^{15} - 1 = 32767$. Здесь от результата отнимается 1, так как первым числом принято считать не 1, а 0. Аналогичные рассуждения легко провести в отношении чисел двойной разрядности (рис. 6.1, *в* и *г*), занимающих в ОЗУ четыре ячейки.

Если задано значение, превышающее физические возможности его представления в ячейках ОЗУ, результат будет неверным. Это иллюстрируют следующие примеры:

50000 U. 5000 OK — результат верен, так как задан вывод числа *n* без знака, у которого предельное значение равно 65535;

50000 . — 15536 OK — результат неверен, так как задан вывод числа *n*, у которого предельное значение равно 32767;

—50 . —50 OK — результат верен, так как задан вывод числа *n*, у которого предельное значение равно —32768 (для отрицательных чисел).

В версии fig-FORTH отсутствуют отдельные слова, имеющиеся в версии FORTH-79. Однако их легко ввести дополнительно — см. § 7.1.

Аналогично этому, отсутствующие у FORTH-79 и FORTH-83 слова 2* и 2/ (имеющиеся у fig-FORTH) легко ввести дополнительно как:

: 2* 2 * ;

: 2/ 2 / ;

Таким образом, любая версия языка Форт может быть легко превращена в другую заданием или переименованием недостающих слов. Это относится не только к арифметическим операциям, но и к другим возможностям языка.

Теперь отметим специфику операций с числами двойной разрядности (*d*). Такое десятичное число должно в конце заканчиваться точкой, иначе применение слова D. будет некорректным. Ниже даны примеры, поясняющие это правило:

25 . 25 OK (результат верен)

25 D. 1638425 OK (результат неверен)

25. D. 25 OK (результат верен)

25 S—> D D. 25 OK (результат верен)

В последнем примере используется специальное слово S—> D, обеспечивающее преобразование числа одинарной разрядности *n* в число двойной разрядности *d*. Поэтому после числа $n = 25$ точка уже не нужна.

Результат цепочных вычислений будет неверным, если в ходе промежуточных операций превышены допустимые пределы задания чисел.

Например,

20000 10 * 100 / . 33 OK (результат неверен)

Результат должен быть 20000 и находится в пределах, допустимых для чисел одинарной разрядности. Однако промежуточный результат $20000 \cdot 10 = 200000$ выходит за эти пределы, что и порождает ошибку. В то же время двойное слово */ в этой ситуации даст верный результат:

```
20000 10 100 */ . 2000 ОК
```

Объясняется это тем, что слово */ вводит в действие аппарат вычислений для чисел двойной разрядности.

До сих пор мы рассматривали арифметические операции только с десятичными числами. Однако целочисленные версии языка Форт могут работать с числами, имеющими различное основание (обычно от 2 до 70). Часто применяются следующие основания:

Основание	Наименование
2	BINARY (двоичное)
8	OCTAL (восьмеричное)
10	DECIMAL (десятичное)
16	HEX (шестнадцатеричное)

Изменение основания производится словом

b BASE !

При этом значение *b* основания приписывается специальной фор-переменной BASE. Само слово BASE выработывает начальный адрес ячейки ОЗУ, в которой хранится *b*.

С помощью слова

```
BASE @ DECIMAL
```

можно вывести на печать значение *b*. Например:

```
16 BASE ! ОК (задано  $b=16$ )
```

```
BASE @ DECIMAL . 16 ОК (получено  $b=16$ )
```

Однако следует помнить, что при этом слово DECIMAL возвращает основанию счета значение, равное 10.

С помощью слова BASE можно задавать слова для установки нужного основания, например:

```
: BINARY 2 BASE ! ;
```

```
: OCTAL 8 BASE ! ;
```

и т. д. (для других оснований).

Шестнадцатеричная (HEX) система счета получила широкое распространение в вычислительной технике. Разряд шестнадцатеричного числа может иметь значения:

```
(HEX) 0 1 2 3 4 5 6 7 8 9 A B C D E F
```

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

На языке Форт с шестнадцатеричными числами могут выполняться операции с заданием их в естественном виде, например

HEX ОК (задание основания $b = 16$)
 A 1 + . В ОК (результат $B = A + 1$)
 ABC 2 + . ABE ОК (результат $ABE = ABC + 2$)

Ряд примеров операций над числами с различными основаниями описан в гл. 8 для версий MVP-FORTH и PC/FORTH ПЭВМ класса IBM PC.

§ 6.2. Операции со стеком

В табл. 6.2 приведены слова, обеспечивающие операции с арифметическим стеком для версий языка Форт: FORTH-79, fig-FORTH и FORTH-83. Большинство из них совершенно аналогично описанным выше (§ 2.5) для версии FSP88. Все примеры, приведенные в § 2.5, могут быть использованы для ознакомления с действием основных слов: DUP, ?DUP, OVER, PICK, DROP, SWAP, ROT, ROLL и DEPTH. Следует помнить, однако, что в описываемых здесь версиях языка Форт они относятся только к целым числам одинарной разрядности.

В версиях FORTH-79 и FORTH-83 имеется ряд слов аналогичного указанному в табл. 6.2 назначения, но рассчитанных на применение с числами двойной разрядности. Первым знаком таких слов является цифра 2. Действие этих слов в остальном аналогично действию слов, ориентированных на операции с числами одинарной разрядности. Многие из этих слов имеются и в расширенных версиях fig-FORTH (они в табл. 6.2 отмечены звездочкой). В минимальной типовой версии данные слова отсутствуют.

В § 7.1 приводится полный набор слов версии fig-FORTH, делающий ее полностью совместимой с версией FORTH-79 (и в значительной степени с FORTH-83). Следует обратить особое внимание на различные толкования слова S0 в версиях FORTH-79 и fig-FORTH. В версии FORTH-79 слова SP@ и S0 дают адреса указателей вершины первой ячейки стека и его дна (ячейки, следующей за последней ячейкой стека, занятой числом) — рис. 6.2, а. Слово SP! можно использовать для очистки арифметического стека.

В версии fig-FORTH слово S0 дает адрес особой двойной ячейки ОЗУ, в которую вписывается адрес дна стека (рис. 6.2, б). Эта ячейка и называется указателем стека. Таким образом, команда S0 в версии FORTH-79 на языке fig-FORTH заменяется эквивалентной последовательностью слов S0 @.

Как уже отмечалось, форт-системы программирования наряду с арифметическим стеком имеют еще и стек возврата. В табл. 6.3 приводятся слова, обеспечивающие операции со стеком возврата. Они сводятся к определению адресных координат указателя стека возврата и обмену содержимым между вершинами арифметического стека и стека возврата.

Операции с арифметическим стеком (звездочкой помечены слова расширенных версий)

FORTH			Стек	Действие слова
79	fig	83		
DUP ?DUP	DUP — DUP	DUP ?DUP	$n_ _ _ n \ n$ $n_ _ _ n \ n$ при $n \neq 0$	Дублирует n на вершине стека Дублирует $n \neq 0$ на вершине стека (при $n = 0$ дублирования нет)
2DUP	2DUP *	2DUP	$d_ _ _ d \ d$	Дублирует число d двойной разрядности на вершине стека
OVER	OVER	OVER	$n1 \ n2_ _ _ n1 \ n2 \ n3$	Помещает на вершину стека копию второго от вершины числа $n1$
2OVER	2OVER *	2OVER	$d1 \ d2_ _ _ d1 \ d2 \ d3$	Помещает на вершину стека копию второго от вершины числа $d1$
PICK	PICK	PICK [*]	$n_ _ _ _ n_n$	Помещает на вершину стека копию n_n — n -го от вершины числа (без учета заданного n)
DROP 2DROP	DROP 2DROP *	DROP 2DROP	$n_ _ _ _$ $d_ _ _ _$	Уничтожает число n на вершине стека Уничтожает число d на вершине стека

Таблица 6.2 (продолжение)

FORTH			Стек	Действие слова
79	fig	83		
Her	Her	CLEAR	$n_ _ _ _$	Очищает стек
SWAP	SWAP	SWAP	$n1\ n2_ _ _ n2\ n1$	Меняет два числа $n1$ и $n2$ местами в ячейках стека у вершины
2SWAP	2SWAP*	2SWAP	$d1\ d2_ _ _ d2\ d1$	Меняет два числа $d1$ и $d2$ местами в ячейках стека у вершины
ROT	ROT	ROT	$n1\ n2\ n3_ _ _ n2\ n3\ n1$	Вращает три числа у вершины стека
2ROT	Her	2ROT	$d1\ d2\ d3_ _ _ d2\ d3\ d1$	Вращает три числа двойной разрядности у вершины стека
ROLL	Her	ROLL	$n_ _ _ (n)$	Вращает n чисел у вершины стека
DEPTH	Her	DEPTH	$_ _ _ n$	Помещает на вершину стека количество чисел в стеке
S0	S0	Her	$_ _ _ A$	Помещает на вершину стека адрес указателя дна стека
SP@	SP@	Her	$_ _ _ A$	Помещает на вершину стека адрес указателя вершины стека. Запуск подпрограммы
SP!	SP!	Her	$_ _ _$	Иницирует стек с использованием адреса S0

Операция со стеком возврата

FORTH			Стек	Действие слова
79	fig	83		
R0	R0	Нет	_____A	Помещает на вершину стека адрес переменной — указателя стека возврата
RP!	RP!	Нет	n_____	Запускает подпрограммы инициализации стека возврата значением из R0
RP@	RP@	Нет	_____A	Вызывает адрес подпрограммы инициализации стека возврата
> R	> R	> R	n_____	Записывает число с вершины арифметического стека на вершину стека возврата
R>	R>	R>	_____n	Записывает число с вершины стека возврата на вершину арифметического стека
R@	R	R@	_____n	Копирует число с вершины стека возврата (с сохранением его на ней) на вершину арифметического стека

Хорошим примером на применение слов > R, R@ (R в версии fig-FORTH) и R> является слово:

```
3DTIMES x y z n_____nx ny nz
```

Оно обеспечивает умножение на число n сразу трех чисел: x , y и z .

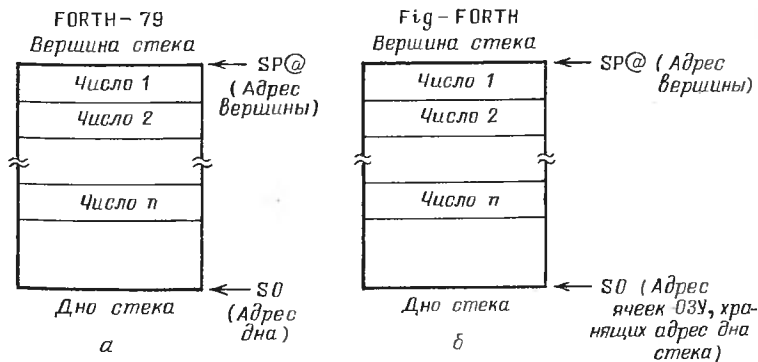


Рис. 6.2. Структура стека в версии FORTH-79 (а) и fig-FORTH (б)

Результат — числа nx , ny и nz — получается в виде чисел двойной разрядности. Словарная статья для этого слова имеет вид:

```
: 3DTIMES DUP >R * ROT R@ * ROT R> * ROT ;
```

Рекомендуем читателю внимательно разобраться в этой простой словарной статье, что поможет понять действие слов >R, R@ и R>.

§ 6.3. Операции с памятью, организация констант, переменных и массивов

В табл. 6.4 дается сводка слов, обеспечивающих операции с памятью (ОЗУ ПЭВМ). Приняты следующие обозначения: A — адрес (начальный) ячейки ОЗУ, n — целое число одинарной разрядности (занимает две ячейки ОЗУ), b — байт (целое число от 0 до 255, занимающее одну ячейку ОЗУ), d — целое число двойной разрядности (занимает четыре ячейки ОЗУ).

Основной набор слов (@, !, ?, +!, MOVE, FILL, ERASE и BLANKS) относится к операциям, обслуживающим действия над числами n одинарной разрядности. Приставка C (в словах C@, C! и CMOVE) означает, что слова оперируют с байтами-кодами, а приставка 2 (в словах 2@ и 2!) относится к операциям над числами двойной точности. Действие этих слов (табл. 6.4), за исключением поправки на тип чисел, аналогично описанному в § 2.6. Исключением является новое слово BLANKS, заполняющее n ячеек ОЗУ с адреса A кодом пробела (32 по ASCII).

Ряд слов позволяет судить об использовании ОЗУ. Основные из них указаны ниже.

Слово

HERE_ _ _ A

помещает на вершину стека адрес первой свободной ячейки словаря. Очередное вводимое слово будет начинаться с этого адреса.

Слово

FREE_ _ _ n

помещает на вершину стека число свободных ячеек (байтов) ОЗУ. Служит для оценки возможности дальнейшего наращивания словаря.

Слово

SIZE_ _ _ n

помещает на вершину стека число занятых ячеек ОЗУ (байтов).

Описываемые целочисленные версии языка Форт имеют довольно развитый и более удобный (в сравнении с FSP88) механизм задания констант, переменных, массивов и таблиц. Основные слова для этого сведены в табл. 6.5. Как и в версии FSP88, организация этих данных тесно связана с организацией памяти. В частности, каждая константа, переменная или компонента массива локализуется во вполне определенной области ОЗУ. Однако в версиях FORTH-79, fig-FORTH и FORTH-83 имеются два принципиальных отличия: предусмотрено прямое задание имени констант и переменных, а занимаемая ими область ОЗУ расположена непосредственно в области, занятой под словарную статью в машинных кодах. Таким образом, константы и переменные могут задаваться в режиме непосредственного исполнения (вне рамок :-определения слова).

Слова для операций с памятью (в стеке b — значение байта)

FORTH			Стек	Действие слова
79	fig	83		
@	@	@	$A_ _ _ n$	Заменяет адрес A числом n
!	!	!	$n A_ _ _$	Заносит число n по адресу A
C@	C@	C@	$A_ _ _ b$	Заменяет адрес A байтом b
C!	C!	C!	$b A_ _ _$	Заносит байт b по адресу A
2@	Нет	2@	$A_ _ _ d$	Заменяет адрес A числом d
2!	Нет	2!	$d A_ _ _$	Заносит число d по адресу A
? DUMP	? DUMP	? DUMP	$A n_ _ _$ $A n_ _ _$	Вызывает на индикацию число n с адресом A Вызывает на индикацию n чисел с адресами, начиная с A
+!	+!	+!	$n A_ _ _$	К числу по адресу A прибавляет n
MOVE	MOVE	Нет	$A1 A2 n_ _ _$	Копирует n чисел с адреса $A1$ в область ОЗУ с адресом, начиная с $A2$
<CMOVE	<CMOVE	<CMOVE	$A1 A2 n_ _ _$	Копирует n байтов с адреса $A1$ в порядке уменьшения адресов в область ОЗУ с адресом, начиная с $A2$
CMOVE'	CMOVE	CMOVE	$A1 A2 n_ _ _$	Копирует n байтов с адреса $A1$ в порядке роста адресов в область ОЗУ с адресом, начиная с $A2$
Нет	Нет	CMOVE>	$A1 A2 n_ _ _$	Копирует n байтов с адреса $A1$ в обратном порядке в область ОЗУ с адресом, начиная с $A2$
FILL	FILL	FILL	$A n b_ _ _$	Заполняет байтами b область из n ячеек ОЗУ с начальным адресом A
ERASE BLANKS	ERASE BLANKS	ERASE BLANKS	$A n_ _ _$ $A n_ _ _$	Заполняет кодом нуля n ячеек ОЗУ с начальным адресом A Заполняет кодом пробела (32 для ASCII) n ячеек ОЗУ с начальным адресом A
HERE	HERE	HERE	$_ _ _ A$	Помещает на вершину стека адрес первой свободной ячейки ОЗУ
FREE SIZE	FREE SIZE	FREE Нет	$_ _ _ A$ $_ _ _ n$	Помещает на вершину стека число n свободных байтов ОЗУ Помещает на вершину стека число n занятых байтов ОЗУ

Слова для организации констант, переменных, массивов и таблиц

FORTH			Стек	Действие слова
79	fig	83		
CONSTANT 2CONSTANT	CONSTANT Нет	CONSTANT 2CONSTANT	n ___ d ___	Задаёт константу n в форме n CONSTANT Имя Задаёт константу d в форме d 2CONSTANT Имя
USER	USER	Нет		Задаёт переменную в виде A USER Имя
VARIABLE 2VARIABLE	VARIABLE 2VARIABLE	VARIABLE 2VARIABLE	n^* ___ d^* ___	Задаёт переменную в виде n^* VARIABLE Имя. Задаёт переменную в виде d^* 2VARIABLE Имя (в fig-FORTH n^* и d^* задают начальное значение переменной)
ALLOT , C,	ALLOT , C,	ALLOT , C,	n ___ n ___ n ___	Резервирует в ОЗУ n байтов (используется после объявления слова) Резервирует в ОЗУ двойную ячейку (2 байта), заносит в нее значение числа n одинарной разрядности Резервирует в ОЗУ 1 байт, задает ему значение n (от 0 до 255)
CREATE DOES> <BUILDS	CREATE DOES> <BUILDS	CREATE Нет Нет	___ ___ ___	Задаёт слово с указанным именем в конструкции CREATE Имя Завершает задание слова. Используется в конструкции (FORTH-79) : Имя CREATE ... DOES> ... ; Аналогично слову CREATE. Используется в конструкции : Имя <BUILDS ... DOES> ... ;

Для задания константы одинарной разрядности используется выражение

`n CONSTANT Имя`

где *n* — число (значение константы), *Имя* — имя константы. Например,

`31416 CONSTANT PI`

задает константу *PI* со значением 31416 (число $\pi \cdot 10000$).

Аналогично выражение

`d 2CONSTANT Имя`

задает константу со значением *d* двойной разрядности. Так,

`3141592654 CONSTANT DPI`

задает константу *DPI* со значением 3141592654 (число $\pi \cdot 1000000000$ двойной разрядности).

По существу, слова `CONSTANT` и `2CONSTANT` резервируют в ОЗУ соответственно 2 и 4 ячейки и указывают на их содержимое. Поэтому выражение

`Имя_ _ _n`

заносят на вершину стека значение *n* константы, а

`Имя . _ _ _` (индикация *n*)

выводит *n* на индикацию. Так, для нашего примера имеем

`PI . 31416 ОК`

Адрес начальной ячейки ОЗУ константы (или просто адрес константы) можно получить, используя специальное слово ' (апостроф):

`' Имя_ _ _А`

В частности, это позволяет изменить значение константы

Новое значение ' Имя !

Например,

`314 ' PI !`

изменит значение константы *PI* на новое значение 314 (вместо прежнего 31416).

Переменной обычно называют группу символов, которой предписывается определенное числовое значение, способное изменяться по ходу выполнения программы. Эта группа знаков является именем переменной. Переменная задается в виде (для FORTH-79 и FORTH-83)

`VARIABLE Имя`

если ее значение — число n одинарной разрядности, и в виде

2VARIABLE Имя

если ее значение — число d двойной разрядности.

Например, выражение

VARIABLE ALPFA

задает слово ALPFA как имя переменной. Выражение

n Имя ! n _ _ _

присваивает переменной числовое значение n , выражение

Имя ? _ _ _

позволяет вывести n на индикацию, а выражение

Имя @ _ _ _ n

выводит значение переменной n на вершину стека.

Пример.

VARIABLE ALPFA OK	(задана переменная ALPFA)
100 ALPFA ! OK	(ALPFA←100)
ALPFA ? 100 OK	(вызов значения ALPFA на индикацию)
ALPFA @ OK	(вызов значения ALPFA на вершину стека)
. 100 OK	(индикация значения ALPFA)

Несколько иначе задаются переменные в версии fig-FORTH:

n VARIABLE Имя

Здесь n — начальное значение объявленной переменной в виде числа одинарной разрядности.

Соответственно выражение

d 2VARIABLE Имя

объявляет переменную с указанным именем и с начальным значением d в виде числа двойной разрядности.

Пример.

0 VARIABLE ALPFA OK	(задает ALPFA←0)
100 ALPFA OK	(задает ALPFA←100)
ALPFA @ . 100 OK	(выводит $n=100$ на индикацию)

Принятый в fig-FORTH способ задания переменных более строгий с позиций теоретического программирования, так как он исключает задание переменной с произвольным начальным значением. Однако, если оно не задано, переменная все же будет считаться объявленной, но ПЭВМ выведет сообщение об ошибке. Объявленная константа или переменная входит в словарь под своим именем.

При задании констант и переменных в ОЗУ формируются блоки, содержащие заголовок словарной статьи (коды имени и начальный адрес тела статьи) и тела статьи. Тело статьи представляет собой сдвоенную ячейку при задании констант и переменных со значениями одинарной разрядности (рис. 6.3, а) и две сдвоенные ячейки (рис. 6.3, б) для констант и переменных двойной разрядности.

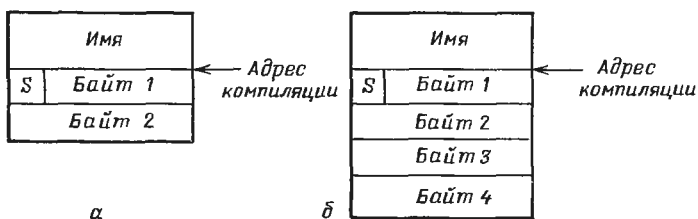


Рис. 6.3. Структура тела переменной одинарной разрядности (а) и двойной (б)

Еще один вид переменных — USER-переменные или пользовательские переменные задаются в виде

n USER Имя

Здесь n указывает на смещение в ОЗУ адреса ячейки, хранящей числовое значение переменной с объявленным именем. Исполнение слова-имени USER-переменной оставляет на вершине стека абсолютный адрес ячейки, хранящей числовое значение переменной.

Пример.

0 USER A ОК (задана переменная A со смещением $n=0$)
 1 USER B ОК (задана переменная B со смещением $n=1$)
 A U . 24064 ОК (абсолютный адрес переменной A)
 B U . 24065 ОК (абсолютный адрес переменной B — он смещен на 1 относительно адреса переменной A)
 123 A ! ОК (переменной A присвоено значение 123)
 A ? 123 ОК (индикация значения $A=123$)
 456 B ! ОК (переменной B присвоено значение 456)
 B ? 456 ОК (индикация значения $B=456$)
 A ? —14213 (иллюстрация перекрытия)

При объявлении USER-переменных пользователь должен следить за тем, чтобы их адресные пространства не перекрылись. Пример перекрытия дан выше. Переменные A и B, объявленные как USER-переменные, занимают две однобайтовые ячейки ОЗУ: Однако при задании переменной B это не было учтено и смещение n было задано равным 1 (надо было задать 2 для устранения перекрытия). В результате присвоение переменной B числового значения нарушает содержимое ячеек ОЗУ, в которых ранее хранилось числовое значение переменной A (и оно становится иным).

Если стереть переменную В, исполнив

FORGET В ОК

и вновь определить как

2 USER В

то получим:

123 А ! 456 В ! ОК

А ? 123 ОК (результат верен)

В ? 456 ОК (результат верен)

Как видно, в этом случае перекрытия адресного пространства USER-переменных А и В уже нет.

Для создания данных более сложного типа — суперпеременных (таблиц, векторов, матриц и т. д.) используется расширение адресного пространства, отведенного под тело переменной. Расширение задается в байтах с помощью слова

n ALLOT

где *n* — число байтов в расширенной области ОЗУ. Слово ALLOT должно использоваться после указания (определения) слов, к которым оно относится.

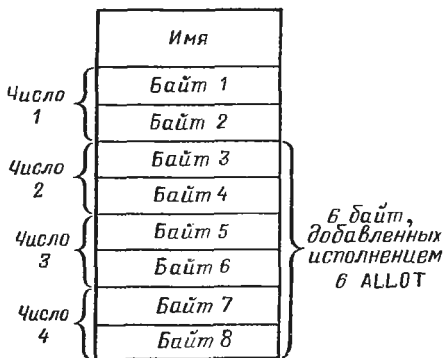


Рис. 6.4. Структура тела переменной с расширением словом ALLOT

Например, в версии fig-FORTH задана переменная

0 VARIABLE VECTOR ОК

При этом она вначале имеет единственное значение 0. Зададим

6 ALLOT ОК

Это значит, что к одной вдвоенной ячейке переменной VECTOR добавлены еще три вдвоенных ячейки (6 байт). Теперь переменная VECTOR имеет структуру рис. 6.4 и представляет собой вектор или одномерный

массив, способный хранить четыре числа одинарной разрядности. Если A — начальный адрес нулевой (по индексу) компоненты вектора, то $A + 2$, $A + 4$ и $A + 6$ есть адреса трех последующих компонент.

Пример. Записать в ячейки вектора VECTOR числа 100, 101, 102 и 103. Для этого нужно выполнить операции:

```
100 VECTOR !      (запись числа 100 по адресу A)
101 VECTOR 2 + !  (запись числа 101 по адресу A + 2)
102 VECTOR 4 + !  (запись числа 102 по адресу A + 4)
103 VECTOR 6 + !  (запись числа 103 по адресу A + 6)
```

Следующим образом можно проверить содержимое ячеек вектора:

```
VECTOR ?      100 (содержимое ячейки с адресом A)
VECTOR 6 + ?  103 (содержимое ячейки с адресом A + 6)
```

Расширение ОЗУ на 2 байта по мере ввода обеспечивает слово

```
, n _ _ _
```

Другое слово

```
- C, b _ _ _
```

расширяет ОЗУ на 1 байт. Например

```
0 VARIABLE 1 DATA
1 C, 2 C, 3 C, 4 C, 5 C, OK
```

Такая переменная будет содержать одну сдвоенную ячейку (в ней число 0) и пять одинарных ячеек с числами (в них 1, 2, 3, 4 и 5). Другие примеры задания векторов, таблиц и матриц описаны в гл. 8.

Форт обладает дополнительными возможностями по заданию данных различного специального типа. Для этого используются слова CREATE, DOES > и < BUILDS. Слово CREATE используется в виде

```
CREATE Имя
```

для задания новых слов. Для этого применяется конструкция

```
: Имя CREATE ... DOES> ... ;
```

где слово DOES> закрывает предложение, объявленное словом CREATE.

В версии fig-FORTH слово CREATE нельзя использовать совместно с DOES>. В этой версии используется другое слово < BUILDS. Оно используется и в версии FORTH-79. Таким образом, в этом случае применяется конструкция

```
: Имя < BUILDS ... DOES> ... ;
```

Пример. Слово ARRAY служит для задания массива чисел в виде n ARRAY Имя

где n — количество чисел в массиве. Листинг слова ARRAY, выполняющего такую функцию, следующий:

```
: ARRAY <BUILDS DUP + ALLOT DOES>  
SWAP DUP + + ;
```

Заметим, что примененные фрагменты DUP + аналогичны несколько более медленным 2*. Здесь фрагмент

```
<BUILDS DUP + ALLOT DOES>
```

обеспечивает задание массива объемом $2n$ байтов (т. е. n чисел одинарной разрядности).

Важно отметить, что в данном случае имя массива задается при выполнении слова ARRAY, т. е. вне определения словами : и ;. Таким образом, создается новый тип определения данных.

Примеры.

```
3 ARRAY DEMO OK (создан массив на три числа с именем  
DEMO)
```

```
10 0 DEMO ! (задана компонента  $n_0=10$ )
```

```
20 1 DEMO ! (задана компонента  $n_1=20$ )
```

```
30 2 DEMO ! (задана компонента  $n_2=30$ )
```

```
2 DEMO ? 30 (вызвана компонента  $n_2=30$ )
```

```
1 DEMO . 20 (вызвана компонента  $n_1=20$ )
```

Как видно из этих примеров, ввод числа n_n в массив выполняется следующим образом:

```
Число  $n$  DEMO !
```

Вызов числа n из массива выполняется так:

```
 $n$  DEMO ? (Индикация  $n_n$ )
```

```
 $n$  DEMO @ (Помещает  $n_n$  на вершину стека)
```

Подобным образом могут создаваться массивы и векторы с другими именами. Например, слово

```
4 ARRAY VEC OK
```

создаст еще один массив с именем VEC, содержащий четыре компоненты. Возможно создание и других видов данных: таблиц, двумерных массивов, массивов констант и т. д.

§ 6.4. Операции ввода-вывода и преобразования чисел

Операции ввода-вывода чисел целочисленных версий языка Форт представлены в табл. 6.6. Большая группа слов этой таблицы (до слова ·R) аналогична используемой в версии FSP88 и подробно описанной

в § 2.4. Исключением являются слова

BELL — — — (создает звуковой сигнал)

BL — — — (создает пробел при печати и помещает на вершину стека его код *c*)

Звуковой сигнал в зависимости от конструкции ПЭВМ может создаваться электрическим звонком, зуммером или громкоговорителем (в виде посылки с частотой 800—1000 Гц и длительностью 0,5—1 с). Слово BL создает пробел при печати и одновременно оставляет на вершине стека код пробела — как знака ASCII (обычно это число 32).

Для печати таблиц весьма удобен ряд слов (.R, D.R и U.R), осуществляющих печать (или вывод на экран дисплея) чисел *n*, *d* и *u* с отступом последней цифры вправо, дающим ширину пробела *w* (от *width* — ширина). Они позволяют создавать колонки чисел, выровненные по левому краю.

В ряде случаев необходим вывод данных в специальной форме. Например, часы, минуты и секунды часто задаются в виде HH:MM:SS, т. е. двоеточием в качестве разделительного знака. Номера телефонов часто указываются через тире, скажем 123—45—67. Для задания вывода данных в такой форме используется специальный шаблон <# ... #>, где слово <# открывает преобразование, слово #> закрывает его, а многоточие (...) означает тело преобразования (см. ниже).

Преобразование чисел всегда идет справа налево, т. е. от последней цифры к первой. Так, слово # обеспечивает преобразование одной цифры числа *ud* в соответствующее символьное представление. Введем слово C со словарной статьей

: C. <# # # # #> TYPE ;

Поскольку в теле конструкции <# ... #> здесь помещены три слова #, это означает, что лишь три знака числа *ud* будут преобразованы в символьное выражение. После знака #>, завершающего преобразование, в вершине стека остаются адрес *A* и число преобразованных элементов *n*, т. е. данные, необходимые для печати выражения словом TYPE.

Поэтому слово TYPE после слова #> даст распечатку числа в преобразованном виде. Действительно, следующие примеры убеждают в этом:

1.	. C	001 OK
12.	C	012 OK
123.	C	123 OK
1234.	C	234 OK
12345.	. C	345 OK

Слова для выполнения операций ввода-вывода

(с — код символа по ASCII, w — число отступов вправо при печати)

FORTH			Стек	Действие слова
79	fig	83		
KEY	KEY	KEY	— — — c	Помещает на вершину стека код <i>c</i> (по ASCII) нажимаемой клавиши
EMIT	EMIT	EMIT	c — — —	Выводит на печать (индикацию) символ с кодом <i>c</i> (по ASCII), увеличивает на 1 значение переменной OUT
BELL BL	Нет BL	Нет Нет	— — — — — — c	Создает звуковой сигнал (например, звонок) Помещает на вершину стека код <i>c</i> пробела (десятичный код 32 по ASCII)
SPACE SPACES	SPACE SPACES	SPACE SPACES	n — — —	Создает пробел при печати или индикации Создает <i>n</i> пробелов при печати или индикации
EXPECT	EXPECT	EXPECT	A n — — —	Вводит с клавишного пульта <i>n</i> знаков, размещая их коды с адреса <i>A</i>
TYPE	TYPE	TYPE	A n — — —	Выводит на печать <i>n</i> знаков, коды которых расположены в ячейках ОЗУ с адреса <i>A</i>
COUNT	COUNT	COUNT	A — — — A+1 b	Заменяет адрес <i>A</i> на вершине стека адресом <i>A+1</i> и значением байта <i>b</i>
PAD	PAD	PAD	— — — A	Помещает на вершину стека адрес первой ячейки текстового буфера

	FORTH		Стек	Действие слова
79	fig	83		
—TRAILING	—TRAILING	—TRAILING	$A \ n1 \ _ _ _ _ \ A \ n2$	Из строки с адресом A и длиной $n1$ создает строку с исключением конечных пробелов, оставляет ее адрес A и длину $n2$
.R D.R U.R	.R D.R U.R	.R D.R U.R	$n \ w \ _ _ _ _$ $d \ w \ _ _ _ _$ $u \ w \ _ _ _ _$	Печатает n с w отступами вправо Печатает d с w отступами вправо Печатает u с w отступами вправо
< # # #S	< # # #S	< # # #S	$d \ _ _ _ _ \ d$ $ud1 \ _ _ _ _ \ ud2$ $ud \ _ _ _ _ \ 00$	Используется в конструкции < #... # > Открывает процесс преобразования числа в знаки Преобразует одну цифру ud в знак ASCII Преобразует все оставшиеся цифры числа u в знаки ASCII
HOLD SIGN	HOLD SIGN — SIGN	HOLD SIGN	$c \ _ _ _ _$ $n \ _ _ _ _$ $nd \ _ _ _ _ \ d$	Присоединяет знак с кодом c к преобразуемой строчной переменной Присоединяет знак «—» к преобразуемой строчной переменной, если $n < 0$

Таблица 6.6 (Продолжение)

FORTH			Стек	Действие слова
79	fig	83		
#>	#>	#>	$d \text{ --- } A n$	В конструкции <# ... #> завершает преобразование числа в строчную переменную
CONVERT	(NUMBER)	CONVERT	$d1 A1 \text{ --- } d2 A2$	Преобразует строчную переменную с начальным адресом $A1+1$ с символами числа $d1$ в новое значение с основанием, заданным BASE
NUMBER	NUMBER	NUMBER	$A \text{ --- } d$	Преобразует строчную переменную с адресом $A+1$ в число d с основанием, заданным BASE
Her	Her	KEY?	$\text{--- } f$	Задаёт флаг $f=1$, если клавиша нажата (иначе $f=0$)
Her	Her	SPAN	$\text{--- } A$	Помещает на вершину стека адрес переменной, значение которой — номер последней позиции ввода словом EXPECT
Her	TIB	TIB	$\text{--- } A$	Помещает на вершину стека адрес входного буфера терминала
Her	Her	# TIB	$\text{--- } A$	Помещает на вершину стека адрес переменной, значение которой даёт число знаков во входном буфере

Обратите внимание, что недостающие слева знаки числа заменяются нулями и всегда выводятся три знака числа справа налево.

С помощью слова HOLD в виде

$\langle \# \dots c \text{ HOLD } \dots \# \rangle$

можно между знаками числа вставить любой знак с кодом c по ASCII. Так, если $c=58$, будет вставлено двоеточие, если $c=45$, то тире. Примеры применения шаблонов для задания вывода времени и телефонных номеров описаны в гл. 8.

Как отмечалось, преобразования вида $\langle \# \dots \# \rangle$ выполняются только с числами *ud* двойной разрядности без знака. В более общем случае можно воспользоваться преобразованиями, приведенными в табл. 6.7 [35].

Таблица 6.7

Конструкция $\langle \# \dots \# \rangle$ для чисел различного типа (версии FORTH-79 и FORTH-83)

Вид выводимого на печать числа	Конструкция преобразования
<i>ud</i> — двойной разрядности без знака	$\langle \# \dots \# \rangle$
<i>d</i> — двойной разрядности со знаком	SWAP OVER DABS $\langle \# \dots \text{ROT SIGN } \# \rangle$
<i>un</i> — одинарной разрядности без знака	0 $\langle \# \dots \# \rangle$
<i>n</i> — одинарной разрядности со знаком	DUP ABS 0 $\langle \# \dots \text{ROT SIGN } \# \rangle$

Знак — (минус) в составе предложения $\langle \# \dots \# \rangle$ может вводиться специальным символом SIGN, если преобразуемое число отрицательно. В противном случае он не вводится. Следует обратить внимание на некоторые отличия в использовании слова SIGN в версии fig-FORTH, отмеченные в табл. 6.6.

Для обратного преобразования символьного представления числа в обычное служат слова NUMBER и CONVERT (в версии fig-FORTH вместо этого применяется слово (NUMBER) с аналогичными функциями). Слова CONVERT и (NUMBER) используются в виде

CONVERT $d1 d2 \dots d1 A2$
 (NUMBER) $d1 d2 \dots d1 A2$

Число $d1$ можно рассматривать как фиктивный аргумент. Его задача заключается в инициации стека. Затем эти слова воспринимают знаки символьного представления числа, начиная с адреса $A1+1$, и превра-

щают символическое значение в обычное число $d2$, которое вместе с адресом выводится на вершину стека.

С помощью этих слов и имеющегося в форт-системах текстового буфера удобно создавать специальные слова для диалогового ввода, аналогичные применяемым на Бейсике. При этом необходимы две возможности — вывод комментария и ввод в ответ на него данных. Вывод комментария на Форте, как отмечалось, обеспечивается применением конструкции ". Текст ". Для ввода данных (чисел, кодов, знаков) используется слово EXPECT, а для печати текста — слово TYPE:

```
60000 10 EXPECT ABCD123456 OK
60000 10 TYPE ABCD123456 OK
```

В данном случае текст ABCD123456 из десяти знаков введен в ОЗУ с адреса 60000, затем он словом TYPE выведен из ОЗУ.

Текст можно вводить и в специальный текстовый буфер (до 80 знаков). Его начальный адрес в ОЗУ указывается словом PAD:

```
PAD 10 EXPECT .ABCD123456 OK
PAD 8 TYPE ABCD1234 OK
```

Обратите внимание, что в последнем примере текст оказался урезанным, так как был задан ввод десятью знаками, а вывод только восемью.

Для организации интерактивного обмена с ПЭВМ полезно слово WAIT (пауза). Если его нет, можно задать слово WAIT следующим образом:

```
: WAIT KEY DROP ;
```

Здесь слово KEY останавливает работу до нажатия любой клавиши и затем заносит ее код на вершину стека (он уничтожается словом DROP). Следующая программа иллюстрирует приложение слова WAIT:

```
: DEMO 12 34 + WAIT . ;
```

Если исполнить слово DEMO, обнаружится, что ПЭВМ остановилась без выдачи ожидаемого результата — числа 46. Стоит нажать любую клавишу, как результат тут же появится.

В некоторых версиях языка Форт встречается также слово

```
INKEY_ _ _c
```

Оно, как и слово KEY, составляет код нажатой клавиши, но без остановки ПЭВМ. Например:

```
INKEY 13 OK
```

В данном случае вырабатывается код 13 клавиши перевода строки. Еще один пример

```
PAD 1 EXPECT INKEY 65 OK
```

Здесь останов происходит при исполнении слова EXPECT (ввод одного знака в текстовый буфер). Если введен знак A, то дальнейшее исполнение слов INKEY и . вызовет печать кода 65 этого знака.

§ 6.5. Организация условных выражений, логических операций и циклов

Организация условных выражений, логических операций и циклов в целочисленных версиях языка Форт ничем не отличается от описанной для версии FSP88. Разумеется, за исключением того, что все они относятся только к целым числам. С учетом этого приведенные в § 2.10 примеры могут использоваться для разбора этих операций в версиях FORTH-79, fig-FORTH и FORTH-83. Кроме того, условные выражения и циклы можно использовать только в рамках :-определения; в непосредственном режиме они (в отличие от версий FSP88) не исполняются.

Напомним, что результат условий сравнения и логических операций (табл. 6.8) задается состоянием флага *f*. Значение *f*=TRUE (ИСТИНА) соответствует поднятому флагу, а *f*=FALSE (ЛОЖЬ) — опущенному.

Специфика версии FORTH-83 заключается в том, что для операций сравнения и логических операций FALSE есть число 0, а TRUE — число —1. Например:

2	3	=	0	OK
4	4	=	-1	OK
8	6	>	-1	OK
5	0	>	-1	OK
9	8	=	NOT	-1 OK
9	8	=	NOT	0 OK и т. д.

В других версиях условные выражения при *f*=TRUE выдают значение 1, например

Лист 6.2

Основные операции отношения и логики (версии fig-FORTH и FORTH-79)

```
7 LIST
SCR # 7
0 ( LOGIC OPERATIONS )
1 : DEMOL CR
2 ." 2 4 < -> " 2 4 < . CR
3 ." 4 2 < -> " 4 2 < . CR
4 ." 2 4 > -> " 2 4 > . CR
5 ." 2 0= -> " 2 0= . CR
6 ." 0 0= -> " 0 0= . CR
7 ." 1 0< -> " 1 0< . CR
8 ." 0 0 AND -> " 0 0 AND . CR
9 ." 1 1 AND -> " 1 1 AND . CR
10 ." 1 0 AND -> " 1 1 AND . CR
11 ." 1 0 OR -> " 1 0 OR . CR
```

```

12 ." 1 1 OR -> " 1 1 OR . CR
13 ." 0 0 OR -> " 0 0 OR . CR
14 ." 1 0 XOR -> " 1 0 XOR . CR
15 ." 1 1 XOR -> " 1 1 XOR . CR ;

```

ok

DEMOL

```

2 4 < -> 1
4 2 < -> 0
2 4 > -> 0
2_0= -> 0
0_0= -> 1
1 0< -> 0
0 0 AND -> 0
1 1 AND -> 1
1 0 AND -> 1
1 0 OR -> 1
1 1 OR -> 1
0 0 OR -> 0
1 0 XOR -> 1
1 1 XOR -> 0

```

ok

Условные выражения

f IF ... ELSE ... THEN

обеспечивают выполнение предложения ... после слова IF, если $f = \text{TRUE}$, и предложения ... после слова ELSE, если $f = \text{FALSE}$.

Может использоваться и следующая форма записи условных выражений:

f IF ... THEN

Тогда при $f = \text{TRUE}$ выражение ... выполняется, а при $f = \text{FALSE}$ не выполняется.

Условные выражения могут вкладываться друг в друга, например

IF ... IF ... ELSE ... THEN ELSE ... THEN

Однако пересечение их недопустимо.

Слова для задания циклов представлены в табл. 6.9. Кроме целочисленности управляющих переменных циклов других в сравнении с описанными в § 2.11 отличий в форме конструкций эти слова не имеют. Следует лишь отметить, что в целочисленных версиях языка Форт встречаются служебные слова в скобках: (DO), (LOOP), (+LOOP) и др., которые используются как вспомогательные директивы при применении основных слов DO, LOOP, +LOOP и др. Число *n* в выражении

*n*1 *n*0 DO ... *n* +LOOP

может быть отрицательным. В этом случае *n*1 должно быть меньше *n*0, а управляющая переменная цикла меняется от *n*0 до (*n*1 - 1), уменьшаясь каждый раз на величину *n* (т. е. имеем цикл вида DO ...

Слова для операций сравнения и организации условных выражений (*f* — флаг)

FORTH			Стек	Действие слова
79	fig	83		
0<	0<	0<	$n \text{ --- } f$	$f = \text{TRUE}$, если $n < 0$
0=	0=	0=	$n \text{ --- } f$	$f = \text{TRUE}$, если $n = 0$
0>	Нет	0>	$n \text{ --- } f$	$f = \text{TRUE}$, если $n > 0$
Нет	Нет	0<>	$n \text{ --- } f$	$f = \text{TRUE}$, если $n \neq 0$
<	<	<	$n1 \ n2 \text{ --- } f$	$f = \text{TRUE}$, если $n1 < n2$
=	=	=	$n1 \ n2 \text{ --- } f$	$f = \text{TRUE}$, если $n1 = n2$
>	>	>	$n1 \ n2 \text{ --- } f$	$f = \text{TRUE}$, если $n1 > n2$
<>	Нет	<>	$n1 \ n2 \text{ --- } f$	$f = \text{TRUE}$, если $n1 \neq n2$
D0=	Нет	D0=	$d \text{ --- } f$	$f = \text{TRUE}$, если $d = 0$
D<	Нет	D<	$d1 \ d2 \text{ --- } f$	$f = \text{TRUE}$, если $d1 < d2$
D=	Нет	D=	$d1 \ d2 \text{ --- } f$	$f = \text{TRUE}$, если $d1 = d2$
DU<	Нет	DU<	$ud1 \ ud2 \text{ --- } f$	$f = \text{TRUE}$, если $ud1 < ud2$
NOT	Нет	NOT	$f1 \text{ --- } f2$	Меняет на противоположное состояние флага $f1$ (если $f1 = \text{TRUE}$, то $f2 = \text{FALSE}$)
AND	AND	AND	$n1 \ n2 \text{ --- } n3$	Обеспечивает операцию AND (логическое умножение)
OR	OR	OR	$n1 \ n2 \text{ --- } n3$	Обеспечивает операцию OR (логическое сложение)
XOR	XOR	Нет	$n1 \ n2 \text{ --- } n3$	Обеспечивает операцию XOR (логическое сложение по модулю)
IF ELSE THEN	IF ELSE THEN	IF ELSE THER	$f \text{ --- } \text{---}$ --- ---	Создают конструкцию $f \text{ IF } \dots \text{ ELSE } \dots \text{ THEN}$ причем, если $f = \text{TRUE}$, то выполняется выражение после слова IF если $f = \text{FALSE}$, то выполняется выражение после слова ELSE

Слова для организации циклов

FORTH			Стек	Действие слова
79	fig	83		
BEGIN UNTIL REPEAT WHILE	BEGIN UNTIL REPEAT WHILE	BEGIN UNTIL REPEAT WHILE	---- ---- ---- ----	Задают циклы вида: BEGIN ... <i>f</i> UNTIL или BEGIN ... <i>f</i> WHILE...REPEAT
AGAIN END	AGAIN	AGAIN	----	Используется в виде BEGIN ... AGAIN Аналогично UNTIL
DO LOOP	DO LOOP	DO LOOP	<i>n1 n0</i> ----	Задает цикл <i>n1 n0</i> DO ... LOOP с управляющей переменной, меняющейся от <i>n0</i> до <i>n1</i> — 1 с шагом 1
+LOOP	+LOOP	+LOOP	<i>n</i> ----	Задает цикл <i>n1 n0</i> DO ... <i>n</i> +LOOP с управляющей переменной, меняющейся от <i>n0</i> до <i>n1</i> — 1 с шагом <i>n</i>
I J K I'	I J K I'	I J K I'	---- <i>i</i> ---- <i>j</i> ---- <i>k</i> ---- <i>n1</i>	Помещает на вершину стека значение управляющей переменной внутреннего цикла Помещает на вершину стека значение управляющей переменной внешнего цикла Помещает на вершину стека значение <i>n1</i> (см. DO)
EXIT LEAVE	EXIT LEAVE	EXIT LEAVE	---- ----	Обеспечивает выход из цикла с возвратом к управлению от терминала Обеспечивает выход из цикла заданием управляющей переменной конечного значения

— LOOP). Слова I, J выводят на вершину стека значение управляющих переменных циклов, а I' — значение n1 в заголовке DO.

В качестве полезного примера применения циклов DO ...LOOP рассмотрим реализацию слова PAUSE :

```
PAUSE n_ _ _ (пауза n·0,01 с)
```

Это слово создает останов вычислений (паузу) на время n·0,01 с. Оно весьма полезно, если нужно кратковременно выдать какое-то сообщение или замедлить ход вычислений.

Слово PAUSE можно реализовать, обращаясь к специальным системным переменным ПЭВМ, вырабатывающим временные интервалы, например в 0,01 с. Однако в этом случае словарная статья слова PAUSE окажется машинозависимой. Другой путь заключается в полезном использовании конечного времени выполнения циклов DO . LOOP. Так, если выполнить слово

```
: TCICLE 10000 0 DO LOOP ;
```

то можно выяснить, сколько времени займет выполнение 10000 циклов. Обычно это время составляет около 1 с. Тогда для реализации слова PAUSE можно воспользоваться следующей словарной статьей:

```
: PAUSE 0 DO 100 0 DO LOOP LOOP ;
```

Здесь внутренний цикл, повторяемый 100 раз, задает интервал около 0,01 с, а внешний — интервал n·0,01 с. Если точность задания времени паузы неудовлетворительна, нужно скорректировать время исполнения внутренних циклов, заменив число 100 другим числом, дающим время 0,01 с.

Еще один практический пример использования циклов DO ...LOOP представлен листингом 6.3.

Лист 6.3

Программа выдачи кодов ASCII, подготовленная в редакторе

```
1 LIST
SCR # 1
  0 ( CODES AND SIMBOLS ASCII )
  1 : .ASCII
  2   CR
  3   DO
  4     I 5 .R ( CODES PRT )
  5     SPACE
  6     I EMIT ( SIMBOLS PRT )
  7     I 6 MOD 0= IF
  8     CR THEN
  9   LOOP
10  CR
11 ;
12
13
14
15
```

ok


```
128 32 .ASCII
```

32	33 !	34 "	35 #	36 \$		
37 %	38 &	39 '	40 (41)	42 *	
43 +	44 ,	45 -	46 .	47 /	48 0	
49 1	50 2	51 3	52 4	53 5	54 6	
55 7	56 8	57 9	58 :	59 ;	60 <	
61 =	62 >	63 ?	64 @	65 A	66 B	
67 C	68 D	69 E	70 F	71 G	72 H	
73 I	74 J	75 K	76 L	77 M	78 N	
79 O	80 P	81 Q	82 R	83 S	84 T	
85 U	86 V	87 W	88 X	89 Y	90 Z	
91 [92 \	93]	94 ^	95 _	96 `	
97 a	98 b	99 c	100 d	101 e	102 f	
103 g	104 h	105 i	106 j	107 k	108 l	109 m
110 n	111 o	112 p	113 q	114 r	115 s	116 t
117 u	118 v	119 w	120 x	121 y	122 z	123 {
124 ;	125 }	126 ~				
127 ?						

ok

Эта программа подготовлена в редакторе и записана в структурированном виде. Обращение к программе задается в виде

$(c_k + 1) c_n$. ASCII

где c_n и c_k — начальное и конечное значения кодов ASCII, которые нужно вывести. Подробный комментарий облегчает разбор этой программы, иллюстрирующей выдачу результатов в табличной форме в виде восьми колонок кодов и знаков.

Текст программы можно значительно сократить, отказавшись от структурированной записи (она несет в себе чисто внешние признаки структурированности) и даже от подготовки словарной статьи в редакторе. Тогда та же самая программа примет следующий вид.

Лист 6.4

Программа выдачи кодов ASCII, введенная без применения редактора

```
: .ASCII CR DO I 5 .R SPACE I EMIT I 6 MOD 0= IF C  
R THEN LOOP CR ; ok
```

```
73 32 .ASCII
```

32	33 !	34 "	35 #	36 \$		
37 %	38 &	39 '	40 (41)	42 *	
43 +	44 ,	45 -	46 .	47 /	48 0	
49 1	50 2	51 3	52 4	53 5	54 6	
55 7	56 8	57 9	58 :	59 ;	60 <	
61 =	62 >	63 ?	64 @	65 A	66 B	
67 C	68 D	69 E	70 F	71 G	72 H	

ok

В последнем случае текст программы менее нагляден, но значительно короче.

§ 6.6. Операции редактирования

Как было отмечено, словарные статьи могут вводиться в рамках :-определения. Однако при этом их мнемонические листинги словарных статей после компиляции устраняются. Кроме того, ввод даже одной длинной словарной статьи оказывается не очень удобным. Например, если в конце ее была случайно сделана ошибка, то ввод статьи игнорируется, ПЭВМ сообщает об ошибке и вся словарная статья (включая весь ее правильный текст до ошибки) оказывается утраченной. Еще хуже обстоит дело в тех случаях, когда нужно уничтожить ранее введенное слово, так как при этом потребуются заново вводить все последующие слова путем повторного набора их словарных статей.

В связи с этим в состав версий FORTH-79, FORTH-83 и fig-FORTH включен экранный редактор для подготовки словарных статей. Общее представление о нем дается в § 1.3. Теперь наступила пора познакомиться с ним более подробно.

У многих версий языка Форт экранный редактор является составной частью форт-системы программирования. В этом случае он вводится словом (именем подсловаря)

EDITOR

Ввод редактора сопровождается занесением в словарь ряда его слов, большая часть которых перечислена в табл. 6.10.

Таблица 6.10

Слова редактора

Слово	Полное наименование	Действие слова
EDITOR s CLEAR L s LIST s1 s2 COPY FLUSH	EDITOR CLEAR LIST LIST COPY FLUSH	Редактирование листа: Вводит редактор Очищает лист с номером s Вызывает листинг листа с текущим номером Вызывает листинг листа с номером s Копирует лист s1 на лист s2 Завершает редактирование с закрытием файлов листов
TOP n M	TOP MOVE	Управление курсором: Устанавливает курсор в начало строки Перемещает курсор на n шагов (вправо при n > 0 и влево при n < 0)
n P Текст n D	PUT DELETE LINE	Редактирование линий листа: Вводит текст на линию n Уничтожает линию с номером n

Таблица 6.10 (продолжение)

Слово	Полное наименование	Действие слова
<i>n</i> H	HOLD	Размещает линию <i>n</i> в текстовом буфере
<i>n</i> T	TYPE	Печатает линию <i>n</i> и заносит ее в текстовый буфер
<i>n</i> R	REPLASE	Возвращает текст линии из текстового буфера и размещает его в линии <i>n</i>
<i>n</i> I	INSERT	Вставляет строку <i>n</i> с текстом из текстового буфера
<i>n</i> E	ERASE	Стирает текст линии <i>n</i> (пустая линия остается)
<i>n</i> S	SPREAD	Перемещает все линии, начиная с <i>n</i> , вниз (вставляется пустая линия; номера линий, начиная с <i>n</i> , увеличиваются на 1; линия 15 теряется)
F Текст	FIND	Редактирование строк: Ищет указанный текст и выставляет курсор в его конец. Если текста нет, сообщает об ошибке
B	BACK	Перемещает курсор обратно в начало текстовой строки
N	—	Возвращает курсор в конец выделенной словом F строки
X Текст	—	Находит и стирает указанный текст, оставляя на его месте курсор
C Текст	—	Вставляет на место курсора указанный текст
TILL Текст	TILL	Стирает указанный текст и всю строку до конца
Текст DELETE	DELETE	Стирает указанный текст и всю строку от начала до него
<i>s</i> 1 <i>s</i> 2 INDEX	INDEX	Выводит на экран текст нулевых строк листов с номерами от <i>s</i> 1 до <i>s</i> 2
SCR__ __A _s	SCREEN	Дает адрес системной переменной, т. е. ячейки ОЗУ, хранящей номер листа <i>s</i>
;S	—	Останавливает интерпретацию листа
— —>	—	Обеспечивает переход на интерпретацию следующего листа
<i>s</i> TRIAD	TRIAD	Выводит тексты трех листов, включая лист с номером <i>s</i>
<i>n</i> S . LINE	—	Выводит тексты линии <i>n</i> листа <i>s</i>
<i>n</i> LINE	—	Выдает на вершину стека адрес ячейки ОЗУ, с которого хранится текст линии <i>n</i>

Редактор — достаточно сложная и большая программа, занимающая значительный объем памяти в ОЗУ (порядка 10К байт и выше). Поэтому у ряда версий языка Форт редактор используется как отдельная программа, хранящаяся на магнитном диске или на магнитофонной

ленте, В этом случае она вводится словом

s LOAD

где s — номер листа (экрана), на котором записана программа редактора. Вместо принятого слова «экран» мы будем использовать слово «лист», так как текст листа не всегда вмещается в рабочее поле экрана.

Все слова редактора можно разбить на четыре группы:

- 1) для редактирования листа;
- 2) для управления курсором;
- 3) для редактирования линий;
- 4) для редактирования строк.

К словам для редактирования листа относятся следующие.

Слово

EDITOR

вводит в действие редактор и задает определение принадлежащих ему слов.

Слово

s CLEAR

очищает лист с номером s.

Линии листов редактора заполнены знаком ? (вопросительный знак). При этом каждая линия обычно имеет 64 знака (реже 80).

Слово

s CLEAR

стирает вопросительные знаки, и все линии листа с номером s оказываются пустыми, т. е. готовыми к примеру словарных статей и комментариев.

Слово

s LIST

вызывает на индикацию (экран дисплея) полный листинг листа с номером s, т. е. всех линий с их текстами. Если ранее вводилось слово s CLEAR, то появляется только столбец номеров линий, так как линии текста не содержат. Если текст есть, он появляется на экране.

Слово

L

используется для быстрого вызова на индикацию текущего листа из текстового буфера редактора.

Слова

s1 s2 COPY

позволяют скопировать целиком содержимое листа с номером *s1* на лист *s2*

Слово

FLUSH

завершает и фиксирует ввод листа, объявленного словами *s LIST* или *s CLEAR*. Помечает буферы ОЗУ, хранящие тексты листов, как измененные.

Количество листов, с которыми пользователь может одновременно работать в редакторе, лежит в пределах от 10 до 16, а иногда и больше (в зависимости от типа ПЭВМ и объема ОЗУ). Поэтому числа *s* лежат в пределах от 0 до 10—15 (и выше). Лист 0 обычно используется для записи комментариев. Тексты (листинги) всех листов хранятся в особом текстовом буфере редактора. Каждый лист (64 знака, 16 строк) требует объема ОЗУ 1024 байт. Следовательно, объем ОЗУ, занимаемого только под текстовый буфер редактора, лежит в пределах 10—16К байт и выше. Указание номера *s* выше максимального ведет к сообщению об ошибке.

До ввода очередного слова **FLUSH** любые изменения текста линий наблюдаются на экране дисплея. Но в текстовом буфере редактора сохраняется текст в исходном виде. Поэтому вызов листа словами *s LIST* или *L* приводит к выводу на экран дисплея исходного текста с потерей введенных изменений текста. Если после редактирования текста листа ввести слово **FLUSH**, в буфере фиксируется измененный текст листа. После этого слова *s LIST* и *L* будут вызывать на индикацию измененный текст листа с номером *s*.

Слово для управления курсором.

Слово

TOP

помещает курсор в исходную позицию в начало редактируемой линии.

Слово

n M

перемещает курсор на *n* позиций вправо, если *n* — положительное целое число, и на *n* позиций влево, если *n* — целое отрицательное число.

Слова для редактирования линий листа.

Слово

n P <текст>

для листа с номером *s*, введенного с помощью команд *s LIST* или *s CLEAR*, обеспечивает задание текста <текст> линии с номером *n*.

Например:

2 CLEAR

10 P : DEMO . "ПРИВЕТ!" ;

проведены очистка листа 2 и заполнение его строки 10 словарной статьей — слова DEMO Теперь команда 2 LIST выведет лист 2 с заполненной строкой 10 на печать.

В табл. 6.10 представлен ряд слов для редактирования текстов листов и их отдельных строк. Конкретные реализации редакторов могут иметь большие отклонения в составе этих слов (см., например, описание версий MVP-FORTH и FORTH/PC в гл. 8).

После редактирования слово FLUSH фиксирует изменения текста, а слово

s LOAD

ведет к интерпретации и компиляции текста словарных статей, записанных в листе s. При отсутствии ошибок входящие в текст слова пополняют словарь.

Слово

s1 s2 INDEX

позволяет просматривать текст нулевых строк листов с s1 по s2. Обычно в них помещают названия листов в круглых скобках. Таким образом, можно организовать просмотр листов.

Номер s текущего листа хранит системная переменная SCR. Так, слова

s SCR !

меняют s.

Для вывода номера текущего листа на вершину арифметического стека используются следующие слова:

SCR @

Если необходимо прекратить компиляцию листа в каком-либо месте, используется слово

;S

помещаемое в этом месте.

Напротив, слово

— — >

означает перевод компиляции с листа s на следующий лист $s + 1$. Это слово обычно применяют при составлении сложных программ или пакетов из них, когда они содержат несколько листов, компилируемых последовательно.

Слово

s TRIAD

выводит текст трех листов: обычно $s - 2$, $s - 1$ и s , если $s \geq 2$. Если, к примеру, задать $s=1$ или $s=0$, будут выведены листы 0, 1 и 2.

Слово

n s LINE

выводит на индикацию текст линии n листа s .

Слово

LINE n _ _ _ A

оставляет на вершине стека начальный адрес A , начиная с которого хранится в ОЗУ текст линии n .

§ 6.7. Задание и сохранение слов, работа с электронным квазидиском

Задание словарных статей в целочисленных версиях языка Форт аналогично описанному для версии FSP88. Используемые для этого слова приведены в табл. 6.11.

Следует отметить некоторую специфику задания комментариев и выводимых на печать текстов. Используемые для этого знаки () и ." должны отделяться пробелом от комментариев или текстов (см. примеры ниже):

(ИНТЕГРИРОВАНИЕ) — задано неверно,

(ИНТЕГРИРОВАНИЕ) — задано верно,

."ВВЕДИТЕ ТЕКСТ" — задано неверно,

." ВВЕДИТЕ ТЕКСТ " — задано верно.

Слова в общем списке, выводимые с помощью слова

VLIST

обычно печатаются подряд с разделением пробелами. В некоторых версиях языка Форт они выдаются в виде:

Слово Адрес компиляции

Слово Адрес компиляции

... ..

Слово Адрес компиляции

Под адресом компиляции обычно подразумевают адрес, при обращении по которому происходит исполнение скомпилированной словарной статьи данного слова. С этим адресом связаны наиболее существенные отличия у версии FORTH-79, с одной стороны, и fig-FORTH и FORTH-83 — с другой. Адресом компиляции у версии FORTH-79 явля-

ется адрес поля параметров словарной статьи (см. более подробно ниже), который оставляет на вершине стека выражение

' Имя

Выражение

' Имя @ U.

выводит этот адрес на печать (индикацию).

В версиях fig-FORTH и FORTH-83 адресом компиляции является адрес поля кодов словарной статьи. Поэтому для преобразования адреса поля параметров (см. выше) в адрес поля кодов используется слово CFA.

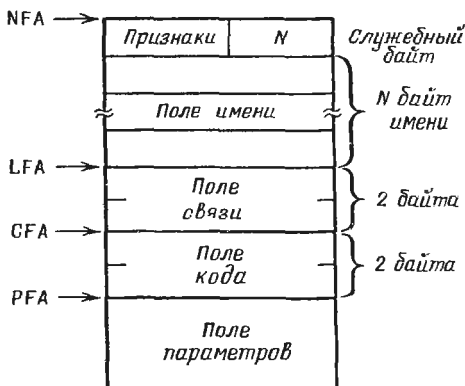


Рис. 6.5. Структура словарной статьи

Словарная статья в ОЗУ образует блок данных (рис. 6.5), содержащих четыре поля с начальными адресами NFA (*name field address*) — адрес поля имени, LFA (*link field address*) — адрес поля связи, CFA (*code field address*) — адрес поля кодов и PFA (*parameter field address*) — адрес поля параметров.

Поле имени начинается со служебного байта. Последние 5 бит несут информацию о длине имени (до 31 знака). Остальные хранят специальные признаки, например признак немедленного исполнения (он обычно занимает старший разряд). Далее идут байты с кодами знаков имени (значение последнего байта увеличено на 128 включением 1 в старший разряд).

Поле связи (2 байта) содержит адрес NFA, т. е. начала предшествующей словарной статьи. Через него словарные статьи связаны друг с другом в цепные списки.

Поле кода (также 2 байта) содержит адрес машинной подпрограммы, которая может быть необходима для выполнения действий,

Операции задания и хранения слов

FORTH			Стек	Действие слова
79	fig	83		
:	:	:	---	Открывает словарную статью в виде : Имя Листинг ; Завершает словарную статью, создает слово с указанным после : именем
;	;	;	---	
)	((---	Открывает поле неисполняемых комментариев Закрывает поле неисполняемых комментариев (Комментарий)
)))	---	
."	."	."	---	Открывает поле печатаемого текста: ." Текст " Закрывает поле печатаемого текста (см. выше)
"	"	"	---	
VLIST	VLIST	WORDS	---	Выводит на печать список всех слов, начиная с текущего по поиску подсловаря
FORGET	FORGET	FORGET	---	Стирает слово с указанным далее именем и все последующие слова
' Имя	' Имя	' Имя	--- A	Помещает на вершину стека адрес начала словарной статьи с указанным именем A
	FENCE	FENCE	--- A	Помещает на вершину стека адрес границы области ОЗУ, недоступной для стирания словом FORGET
EMPTY — BUFFERS UPDATE	EMPTY — BUFFERS UPDATE	EMPTY — BUFFERS UPDATE	---	Очищает буфер
INIT — DISC SAVE — BUFFERS	INIT — DISC SAVE — BUFFERS	— SAVE — BUFFERS	---	Помечает как записанные блоки, которые должны быть сохранены на диске
			---	Инициализирует и очищает диск
			---	Записывает все помеченные словом UPDATE блоки на диск
FLUSH	FLUSH	FLUSH	---	Действует аналогично слову SAVE — BUFFERS

связанных с этим словом. Если таких особых действий нет, то это поле содержит адрес PFA.

Поле параметров содержит обращение (в виде 2-байтных адресов) к словам тела словарной статьи и команды возврата RETURN. Длина поля параметров зависит от числа слов в теле словарной статьи.

В процессе создания словаря словарные статьи укладываются в ОЗУ друг за другом, образуя монолитную последовательность машинных кодов — так называемый сшитый код. При этом нет необходимости в создании промежуточных (резервных) областей ОЗУ (как при реализации версий FP50) и ОЗУ используется более экономно. Так, название словарной статьи требует лишь на 1 байт (счетчика) больше, чем число знаков в нем. Однако поиск слов из-за нерегулярности расположения названий более сложен и требует большего времени, чем у версий с отдельным расположением заголовков и полей параметров словарных статей.

С помощью слова

FORGET Имя

стираются указанное слово и все последующие слова словаря.

Однако в ряде расширений языка FORTH (в частности, fig-FORTH) имеется специальная переменная

FENCE _ _ _ A

помещающая на вершину стека адрес указателя границы областей ОЗУ, недоступной для стирания словом FORGET. Например, если нужно обеспечить возможность стирания начиная с данного слова, то следует исполнить выражение

' Имя FENCE !

Пр и м е р. Введем четыре слова

```
: DEMO1 ." DEMO1 " ;  
: DEMO2 ." DEMO2 " ;  
: DEMO3 ." DEMO3 " ;  
: DEMO4 ." DEMO4 " ;
```

Далее исполним выражение

' DEMO2 FENCE !

и сделаем следующие операции:

```
FORGET DEMO1 DEMO1 ? MSG # 21  
FORGET DEMO2 DEMO2 ? MSG # 21  
FORGET DEMO3 OK  
FORGET DEMO4 DEMO4 ? MSG # 0
```

Итак, попытки стереть слова DEMO1 и DEMO2 оказались безуспешными (ошибка MSG # 21 указывает на недопустимость применения стирания к этим словам), а стирание слова DEMO3 (и последующего DEMO4) прошло успешно. Отметим, что повторное стирание DEMO4 также ведет к сообщению об ошибке, но теперь иной — слова нет в словаре (MSG # 0).

Сохранение защищенных от стирания слов DEMO1 и DEMO2 легко проверить. Например, исполнив слово DEMO1, получаем

```
DEMO1 DEMO1 OK
```

Защищенные от стирания словом FORGET слова сохраняются при исполнении слова WARM («горячий» старт), но они полностью уничтожаются при исполнении слова COLD («холодный» старт). Защищены от любого стирания все слова, входящие в базовый набор словаря (FORTH).

Слова в словаре сохраняются в скомпилированном виде. Однако имеется возможность хранения их в трех мнемонических видах: в редакторе (раздел выше), кассетном и дисковом накопителях. При отсутствии дискового накопителя может использоваться электронный квазидиск (RAMD).

Слова для управления кассетным накопителем (бытовым магнитофоном) обычно не стандартизированы. Применяются слова (и им подобные):

SAVET или SAVETAPE — запись на магнитофон (tape),
LOADT или LOADTAPE — считывание с магнитофона,
VERIFY — контроль правильности записи после исполнения слова SAVET (или SAVETAPE).

Слова для управления записью слов на магнитные диски стандартизированы. Ниже описано их значение. Отметим, что деление ОЗУ на листы (экраны) и линии соответствует делению диска на магнитные дорожки и сектора. Таким образом, на языке Форт реализована идея создания электронного квазидиска (RAMD), вся идеология применения которого соответствует используемой в дисковых накопителях. Поэтому описанные далее команды управления электронным квазидиском относятся и к дисковым накопителям (см. табл. 6.11). Электронный квазидиск широко используется в версиях, ориентированных на простые ПЭВМ с кассетным накопителем.

Процесс программирования при использовании форт-операционной системы с электронным квазидиском (или с дисковым накопителем) можно представить в виде ряда этапов.

1. Инициация диска, реализуемая словом

```
INIT-DISC _ _ _
```

Это слово есть не во всех версиях Форты, так как инициация и раз-

метка диска могут выполняться функциями операционной системы ПЭВМ.

2. Очистка всех блоков буфера электронного квазидиска (т. е. сегментированной области ОЗУ, выделенной под квазидиск) с помощью слова

EMPTY—BUFFERS _ _ _

Эти блоки не переписываются во внешнюю память и помечаются как свободные.

3. Выбор листа (экрана) для подготовки в нем словарных статей. Номер листа хранится в ячейке ОЗУ, адрес которой указывает специальная переменная

SCR _ _ _ A

Задать нужный номер листа *s* можно следующим образом:

s SCR !
s LIST
s CLEAR

В двух последних случаях встроенное в них предложение

s SCR !

выполняется автоматически.

4. Подготовка программ в редакторе (этот процесс подробно описывается в § 6.6).

5. Отметка об изменении буферов с текстом словарных статей обеспечивается словом

UPDATE _ _ _ _

6. Запись содержимого всех блочных буферов, помеченных словом UPDATE, на диск выполняется словом

SAVE—BUFFERS _ _ _ _

Слова UPDATE и SAVE—BUFFERS выполняются автоматически, если после завершения редактирования программы в редакторе задано слово

FLUSH _ _ _ _

Это слово завершает процесс редактирования. В некоторых версиях языка Форт, например fig-FORTH, слова SAVE—BUFFERS нет (но есть FLUSH).

7. Просмотр листингов программ или их печать принтером (при вводе команды его включения) выполняется вводом слова

s LIST

где *s* — номер листа вводимого листинга.

8. Компиляция словарных статей, подготовленных в редакторе, выполняется вводом слова

s LOAD

где s — номер листа, начиная с которого идет компиляция. Знак ;S в тексте редактора останавливает компиляцию, а знак —> переводит компиляцию на следующий лист. Таким образом, можно компилировать большие программы, занимающие ряд листов.

9. Исполнение программы заключается в указании слова — имени программы (или ряда слов, если программа разбита на части, имеющие самостоятельное значение).

10. Завершение работы, если программа выполняется успешно, или переход к пп. 3, 4, если необходима коррекция программ.

Таким образом, подготовка программ носит итерационный и интерактивный характер. В процессе ее проведения может потребоваться неоднократная коррекция листингов, перезапись их на диск и компиляция фрагментов программ. Перед ней не следует забывать о необходимости стирания ранее введенных слов с помощью слова FORGET (общее стирание удобно выполнять вводом слова COLD).

В отдельных довольно редких случаях эффективность форт-программ может быть повышена включением в них фрагментов в машинных кодах. С этой целью в состав расширенных версий языка Форт вводятся слова, указанные в табл. 6.12. Там же приводится форма задания таких фрагментов программ. Следует отметить, что с помощью переменной BASE можно назначить задание кодов в виде чисел с любым основанием (обычно используются десятичные, шестнадцатеричные и реже двоичные коды).

У некоторых версий языка Форт слова, указанные в табл. 6.12, задаются в составе специального подсловаря, который вводится указанием его имени

ASSEMBLER

Организация словаря с подсловарями описана в § 6.8.

После задания и компиляции новых слов обычно возникает необходимость в записи на носители всей форт-системы с новыми словами. Эта процедура часто не стандартизирована и решается с помощью машинозависимых действий. В основном при этом приходится выполнять два этапа таких действий: 1) выяснение областей памяти, занятых форт-системой, и их длины и 2) перемещение указателей словаря, защищающих базовые слова от любого стирания, на новое место. Запись форт-системы после этого производится в системном мониторе (или системными командами встроенного в ПЭВМ языка Бейсик). У многих расширенных версий языка Форт имеется слово SIZE, оставляющее на вершине стека длину (в байтах) области ОЗУ, занятой словарем. В § 7.11 описано слово SFORTH, обеспечивающее автома-

Слова для организации словарных статей с командами в машинных кодах

FORTH			Стек	Действие слова
79	fig	83		
;CODE	;CODE	;CODE	— — —	В конструкции : Имя ... ;CODE задает словарную статью в машинных кодах, помещаемых на место многоточия
CODE	CODE	CODE	— — —	Задаёт заголовок словарной статьи в форме CODE Имя ...
END — CODE	END — CODE	END — CODE	— — —	Фиксирует конец словарной статьи в форме CODE Имя ... END — CODE

тическую реализацию второго этапа сохранения всей системы — перемещение указателей словаря. С помощью этого слова выполняются все подготовительные действия для записи измененной форт-системы на магнитный диск или магнитофонную кассету.

§ 6.8. Организация словарей и подсловарей

По мере включения в форт-систему программирования новых слов их количество быстро растет, и пользователю порою становится трудно ориентироваться в их общем потоке, выводимом словом VLIST (или WORDS в версии FORTH-83). В связи с этим предусмотрено создание ряда словарей с приоритетом их по записи или по поиску. Такой приоритетный словарь называют *текущим* по записи или по поиску.

У большинства форт-систем имеется по крайней мере два словаря: словарь редактора, вводимый словом EDITOR, и словарь базовых слов языка Форт, вводимый словом FORTH. При начальной загрузке FORTH-словарь становится автоматически текущим по поиску и по записи.

Исполнив, к примеру (версия fig-FORTH), слово VLIST, получим
UDG INIT—DISC INKEY ... (и т. д.)

У разных версий могут в начале FORTH-словаря стоять и другие слова. Однако запомним эти для иллюстрации операций со словарями.

Теперь введем слова

EDITOR VLIST

Получим на экране дисплея следующий перечень слов:

```
C TILL X B F N DELETE FIND
1 LINE MATH—TEXT
. . . . .
UDG INIT—DISC INKEY
```

Итак, теперь в начало словаря попала довольно большая группа слов редактора (начиная со слова C). FORTH-словарь базовых слов сместился вниз (у некоторых реализаций языка этот словарь может вообще не выводиться). Новая группа слов и есть подсловарь с именем EDITOR. Как мы видим, чтобы вывести его в приоритетное положение, т. е. сделать текущим по поиску, достаточно указать его имя во входном потоке (ввести слово EDITOR). Указание слова FORTH возвращает приоритет ввода к основному словарю.

В табл. 6.13 приведены слова для организации словаря. Так, слово

VOCABULARY Имя

обеспечивает создание словарной статьи с заданным именем, которая создает новый подсловарь. Упоминание этого имени во входном потоке делает этот подсловарь текущим по поиску.

Чтобы задать слова этого подсловаря, нужно исполнить предложение

Имя DEFINITIONS

повторив данное имя. Теперь все слова, определяемые в рамках : - определения, стаиут словами подсловаря с выбранным именем. В гл. 8 описано применение этих слов для организации словаря с древовидной структурой.

Системная переменная

CONTEXT _ _ _ A

помещает на вершину стека адрес двухбайтной ячейки ОЗУ, которая хранит адрес текущего по поиску словаря.

Другая системная переменная

CURRENT _ _ _ A

помещает на вершину стека адрес двухбайтной ячейки ОЗУ, которая хранит адрес текущего по записи словаря (т. е. того, который может пополняться новыми словами).

Еще одна системная переменная WIDTH задает предельную длину имени слов. Обычно она по умолчанию равна 31 и может изменяться заданием значения переменной WIDTH.

§ 6.9. Управление компиляцией и интерпретацией

Возможности в управлении компиляцией и интерпретацией слов у целочисленных версий языка Форт (FORTH-79, fig-FORTH и FORTH-83) намного обширнее, чем у версий FSP88 и его модификации (табл. 6.14). Это позволяет строить довольно изощренные программные системы, в полной мере использующие возможности как компиляции, так и интерпретации, присущие этим версиям.

До сих пор отмечалось, что директива : (двоеточие) автоматически переводит форт-систему в режим компиляции словарной статьи, следующей за этой директивой. При этом слова этой статьи, как правило, не выполняются, а лишь используются для преобразования словарной статьи в машинные коды (т. е. задания нового слова). Выполнение слова происходит, если оно задано вие :-определения, т. е. не заключено между знаками : и ; (конец компиляции).

Однако у целочисленных версий языка Форт могут встретиться слова, которые исполняются даже в режиме компиляции, т. е. когда они встречаются между знаками : и ; определения новых слов. Более того, исполнение или неисполнение слов в режиме компиляции пользователь форт-системой может устанавливать по своему усмотрению. Для этого слова снабжаются специальными системными признаками *P* и *R*, которые хранятся вместе с ними в ОЗУ ПЭВМ.

Признак *P* разрешает исполнение слов в режиме компиляции, а признак *R* запрещает это. Когда во входном потоке встречается директива : , то она обычно устанавливает признак *R*. Однако слова с ранее установленным признаком *P* будут исполняться, даже если они идут после конструкции

: Имя ...

Слово

IMMEDIATE

(в переводе непосредственный), помещаемое после директивы ; , устанавливает признак *P* непосредственного исполнения для последнего введенного слова.

Пример. Введем следующее предложение:

: DEMO 2 3 + . ; IMMEDIATE

Теперь слово DEMO приобрело признак *P* немедленного исполнения. Вначале попробуем его выполнить обычным способом:

DEMO 5 OK

В данном случае ничего особого пока не видно. Но теперь попробуем определить новое слово DEMO1, в состав которого входит DEMO:

: DEMO1 DEMO ; 5 OK

Слова для организации словарей

	FORTH		Стек	Действие слова
79	fig	83		
FORTH	FORTH	FORTH	— — —	Делает словарь базовых слов FORTH текущим по поиску
EDITOR	EDITOR	EDITOR	— — —	Вводит словарь редактора и делает его текущим по поиску
ASSEMBLER	ASSEMBLER	ACSEMBLER	— — —	Вводит словарь машинных кодов и делает его текущим по поиску
VLIST	VLIST	WORDS	— — —	Выводит список всех слов и названий подсловарей, начиная с текущего по поиску
VOCABULARY	VOCABULARY	VOCABULARY	— — —	В конструкции VOCABULARY Имя задает создание словарной статьи с указанным именем, порождающей новый подсловарь
DEFINITIONS	DEFINITIONS	DEFINITIONS	— — —	В конструкции Имя DEFINITIONS задает всем вводимым далее словам принадлежность к подсловарю с указанным именем
CONTEXT	CONTEXT	CONTEXT	— — — A	Помещает на вершину стека адрес ячейки, хранящей адрес текущего по поиску словаря
CURRENT	CURRENT	CURRENT	— — — A	Помещает на вершину стека адрес ячейки, хранящей адрес текущего по записи словаря
WIDHT	WIDHT	WIDHT	— — — A	Задаёт длину n имени слова (по умолчанию $n=31$)

Слова для управления компиляцией и интерпретацией

	FORTH		Стек	Действие слова
79	fig	83		
IMMEDIATE	IMMEDIATE	IMMEDIATE	— — —	После директивы; устанавливает признак <i>P</i> непосредственного исполнения последнего слова
[COMPILE] COMPILE	[COMPILE] COMPILE	[COMPILE] COMPILE	— — —	Обеспечивает компиляцию слов с признаком <i>P</i>
			— — —	В выражении : AAA ... COMPILE BBB ... обеспечивает компиляцию <i>BBB</i> при исполнении <i>AAA</i>
SMUDGE	SMUDGE	SMUDGE	— — —	Инвертирует признак слова (<i>P</i> на <i>R</i> и <i>R</i> на <i>P</i>)
[[[— — —	Отменяет режим компиляции, задает режим непосредственного исполнения
]]]	— — —	Возвращает к режиму компиляции
STATE	STATE	STATE	— — — A	Помещает на вершину стека адрес ячейки, содержимое которой указывает на режим работы (0—непосредственного исполнения, ≠0 — компиляции)
?COMP	?COMP	Нет	— — —	Помечает слово, допустимое во входном потоке в режим компиляции
?EXEC	?EXEC	?EXEC	— —	Помечает слово, допустимое во входном потоке только в режиме непосредственного исполнения
EXECUTE	EXECUTE	EXECUTE	A — — —	Исполняет слово с адресом поля параметров (иногда кодов) <i>A</i>
INTERPRET	INTERPRET	INTERPRET		Вводит процедуру внешнего интерпретатора текста, который последовательно исполняет или компилирует текст входной строки (в зависимости от значения STATE)

Тут мы обнаружим, что, как только клавишей перевода строки завершён ввод директивы ; , тут же появляется результат 5 (исполнение слова DEMO).

Если внутри :-определения встречается слово с признаком P, которое нужно скомпилировать (а не немедленно исполнить), перед ним следует поставить слово

[COMPILE]

Пример. Введем в дополнение к словам DEMO и DEMO1 новое слово

: DEMO2 [COMPILE] DEMO ; OK

Мы обнаружим, что теперь после ввода директивы ; слово DEMO внутри определения слова DEMO2 уже не выполняется. Задав

DEMO2 5 OK

получим ожидаемый результат исполнения слова DEMO — число 5.

Ещё одно слово

COMPILE

обычно используется в предложениях вида

: AAA ... COMPILE BBB ...

Оно используется, если при исполнении директивы AAA нужно, чтобы она компилировала директиву BBB.

Инвертирование признака (т. е. замена признака P на R, а R на P) осуществляется с помощью слова

SMUDGE NFA _ _ _

Здесь NFA — адрес поля имени словарной статьи (см. § 6.11). Слово SMUDGE может использоваться как внутри, так и вне :-определения.

Ещё один простой способ управления компиляцией группы слов заключается в применении слов [и] (квадратных скобок):

[— отмена режима компиляции с переходом в режим непосредственного исполнения,

] — восстановление режима компиляции.

Пример. Зададим слово

: DEMO3 [2 3 + .] ; 5 OK

Как только будет завершён ввод директивы ; , тут же появится результат (число 5), так как выражение

[2 3 + .]

не компилируется, а непосредственно исполняется.

Режим работы интерпретатора выявляет специальная переменная
STATE _ _ _ A

Содержимое ячейки с адресом A равно 0, если происходит работа в режиме непосредственного исполнения, и не равно 0, если происходит компиляция. Так, в режиме непосредственного исполнения имеем

```
STATE ? 0 OK
```

Слово

```
?COMP
```

используется внутри :-определения для придания заданному слову особого свойства — оно может поступать на вход интерпретатора только в режиме компиляции. В противном случае происходит останов ПЭВМ с сообщением об ошибке.

Другое слово

```
?EXEC
```

также налагает на определяемое слово ограничение — оно должно поступать на вход интерпретатора только в режиме непосредственного исполнения. Иначе ПЭВМ останавливается с сообщением об ошибке.

Пример. Очистим словарь от ранее введенных слов (например, командой COLD) и введем слово

```
: DEMO 2 3 + . ?EXEC ; OK
```

Придадим ему признак P:

```
IMMEDIATE OK
```

Теперь зададим слово

```
: DEMO1 DEMO ; 5 DEMO ? MSG # 18
```

Если бы в конце словарной статьи слова DEMO не было слова ?EXEC, то слово DEMO должно было бы исполниться при определении слова DEMO1 без всяких затруднений. Однако слово ?EXEC разрешает выполнение слова DEMO только в режиме непосредственного исполнения. Поэтому после ввода директивы ; выдается сообщение об ошибке. Однако обычное исполнение слова DEMO в непосредственном режиме дает верный результат:

```
DEMO 5 OK
```

Слово

```
EXECUTE A _ _ _
```

обеспечивает выполнение слова, заданного в версии FORTH-79 адресом его поля параметров (см. подробнее § 6.12). Как отмечалось, этот

адрес можно найти, используя предложение

' Имя

где Имя — слово. В результате на вершине стека останется адрес указанного вслед за апострофом слова.

Пример.

```
123 ' DUP EXECUTE OK  
. 123 . 123 OK
```

В данном случае цепочка слов ' DUP EXECUTE ведет себя как слово DUP.

У некоторых реализаций языка Форт (fig-FORTH, FORTH-83) в качестве адреса *A* может использоваться не адрес поля параметров словарной статьи, а адрес ее поля кодов. В этом случае нужно преобразовать первый адрес во второй с помощью слова CFA. Приведенный выше пример в этом случае можно реализовать так (версия fig-FORTH):

```
123 ' DUP CFA EXECUTE OK  
. 123 . 123 OK
```

Словом EXECUTE следует пользоваться с известной осторожностью. Так, ошибочное указание адреса чревато непредвиденными последствиями, например сбросом форт-системы в режим монитора или отказом от управления. Применение слова EXECUTE обеспечивает реализацию структурного программирования по методу сверху вниз и создание рекурсивных процедур.

§ 6.10. Обработка входного потока

Совокупность слов (команд и данных), поступающих на интерпретатор форт-систем, принято называть *входным потоком*. Он может создаваться различными источниками: терминалом (с клавишным пультом), внешними накопителями (например, дисковыми) или электронным квазидиском. Всем им присваиваются определенные номера.

Системная переменная (табл. 6.15).

```
ВЛК _ _ _ A
```

оставляет на вершине стека адрес ячейки ОЗУ, хранящей номер источника входного потока. Терминалу обычно приписывается значение ВЛК, равное 0.

Информация от внешних устройств (включая организованный в ОЗУ или внешний электронный квазидиск) поступает в виде блоков, содержащих 1024 байт. Эти блоки также нумеруются, начиная с 0 до $N-1$, где N — максимальное число доступных блоков. К каждому блоку однозначно предписывается буфер — область ОЗУ, хранящая

также 1024 байт. Емкость блоков и буферов у отдельных реализаций языка Форт может быть и иной, например 128 байт.

Системная переменная

BLOCK n — — — A

заменяет номер блока n на вершине стека на адрес первого байта буфера A , предписанного к блоку n . К каждому блоку может быть предписано не более одного буфера. Если блока n нет в ОЗУ, то он переписывается из внешней памяти в предписанный ему буфер. Если n не является допустимым номером, то выводится сообщение об ошибке.

Пример, демонстрирующий действие переменной BLOCK (адреса могут быть другими в зависимости от конкретной применяемой ПЭВМ):

1 BLOCK U. 52590 OK

2 BLOCK U. 52722 OK

3 BLOCK U. 52854 OK

Нетрудно заметить, что здесь адреса идут с интервалом 132 байт. Это указывает на то, что буферы имеют объем 128 байт (4 байта используются на организацию буфера в ОЗУ). Объем буферов указывается значением специальной системной переменной B/BUF (см. § 6.12).

• Слово

BUFFER n — — — A

предписывает блочный буфер блоку с номером n и оставляет на вершине стека адрес A первого блока n в буфере. Это слово входит в определение слова BLOCK, за исключением того, что если блока еще нет в памяти, то его можно не пересылать из внешних устройств.

Используя слово CMOVE, можно осуществить загрузку одного блока в другой. Например, если блоки 1 и 3 имеют объем 128 байт, то слова

1 BLOCK 3 BLOCK 128 CMOVE

пересылают эти 128 байт из блока 1 в блок 3.

Кроме отмеченных буферов в ОЗУ имеются текстовый буфер (начальный адрес его дает системная переменная PAD), буфер диска (нижний и верхний адреса его дают переменные FIRST и LIMIT), а также динамический буфер входного потока от терминала. Нижний адрес последнего указывает переменная TIB (от слов *terminal input buffer* — входной буфер терминала), а число занятых ячеек ОЗУ (байт) указывает переменная > IN (или IN в версии fig-FORTH).

Действие слова IN поясняют следующие примеры:

1 DUP IN ? 10 OK

1 DUP DUP IN ? 14 OK

В первом случае число знаков во входном потоке (от знака I до зна-

Слова организации и обработки входного потока

FORTH			Стек	Действие слова
79	fig	83		
BLK	BLK	BLK	— — — A	Помещает на вершину стека адрес ячейки ОЗУ, хранящей указатель источника входного потока (0 — терминал; 1, 2, 3 и т. д. — блоки ОЗУ)
BLOCK	BLOCK	BLOCK	n — — — A	Помещает на вершину стека адрес блока ОЗУ с номером <i>n</i> , с которого считывается входной поток
BUFFER	BUFFER	BUFFER	n — — — A	Помещает на вершину стека адрес буфера <i>n</i>
TIB	TIB	TIB	— — — A	Помещает на вершину стека адрес буфера ввода терминала
> IN	IN	> IN	— — — A	Помещает на вершину стека адрес ячейки ОЗУ, в которой хранится длина текста входного потока
WORD			c — — — A	В конструкции c WORD Текст считывает текст, используя знак с кодом <i>c</i> в качестве ограничителя. Помещает на вершину стека адрес ячейки, хранящей длину текста
FIND	WORD (см. FIND в табл. 6.16)	WORD	c — — — — — — A	Аналогично WORD для FORTH-79, но без вывода адреса
		FIND	— — — 0 — — — A1	В конструкции FIND Текст ищет Текст в словаре и дает адрес слова, совпадающего с текстом, <i>A</i> (иначе <i>A</i> =0)
QUERY	QUERY	Нет	— — — —	В конструкции ищет Текст в словаре. Оставляет на вершине стека адрес слова, совпадающего с текстом, и <i>f</i> =1. В противном случае дает <i>f</i> =0 Вводит текст (до 80 знаков) во входной буфер с терминала

ка ?), включая пробелы между словами, равно 10. Именно это значение хранится по адресу, указанному переменной IN, и поступает на индикацию. Во втором случае добавлены еще одно слово DUP и один пробел. В результате число введенных знаков во входном потоке увеличилось на четыре и составило 14.

Специальное слово WORD служит для считывания текста, поступающего с терминала во входной поток в виде группы знаков, образующих слова. Оно задается в виде

c WORD Текст

Здесь *c* — код так называемого ограничителя. Например, если *c*=32 (или вместо *c* слово BL), то в качестве ограничителя будет использоваться пробел. Так, выражение

32 WORD ABCD OK

приведет к считыванию символов ABCD до тех пор, пока не будет обнаружен пробел, указывающий на конец считывания. Коды знаков размещаются в ОЗУ (в версии FORTH-79 слово WORD оставляет адрес ячейки ОЗУ, в которой хранится число, указывающее на длину считываемого текста). При исполнении слова WORD подсчитывается число введенных символов до ограничителя (счетчиком является первый байт строки). Если ограничитель не найден, > IN дает длину входного потока, иначе значение > IN указывает на литеру, следующую за ограничителем.

Примером применения слова WORD (версия fig-FORTH) является слово

: ASCII 32 WORD HERE I+ C@ . ;

Оно выдает код ASCII указанного вслед за этим словом знака. Здесь слова 32 WORD считывают любой знак (32 — код ограничителя в виде пробела).

Слова HERE I+ дают адрес ячейки ОЗУ, в которой хранится код знака, введенного после слова .ASCII.

Примеры.

.ASCII A 65 OK (код A *c*=65)

.ASCII D 68 OK (код D *c*=68)

Отметим, что слово .ASCII выдает код первого знака текста, например

.ASCII ABCD 65 OK

(выдаст код первого знака A текста ABCD).

Еще одно слово, используемое для анализа слов входного потока в версии FORTH-79:

FIND Текст _ _ _ A

Слово ищет указанный текст в словаре. Если поиск завершился успешно (т. е. текст — слово), на вершину стека помещается адрес этого слова. В противном случае $A=0$. Текст рассматривается как строка со счетчиком (см. далее слово WORD), начинающаяся с определенного адреса.

В версии FORTH-83:

FIND Текст — — — Текст 0 (текст не найден)

FIND Текст — — — A 1 (текст найден)

Здесь 0 и 1 — состояния флага.

Аналогичное по назначению слово — FIND в расширенной версии fig-FORTH приведено в табл. 6.16 (см. § 6.11).

Другое слово

QUERY Текст

при исполнении вызывает останов ПЭВМ с ожиданием вводимого с терминала текста. Текст считывается во входной буфер с терминала, причем длина текстовой строки может достигать до 80 символов. Введенный текст рассматривается как входной поток и исполняется при нажатии клавиши перевода строки.

Пример.

QUERY <ENTER> 2 3 + <ENTER> 5 OK

Здесь после нажатия клавиши перевода строки <ENTER> происходит останов и далее вводится текст 2 3 + Нажатие клавиши перевода строки еще раз ведет к исполнителю текста, т. е. последний рассматривается как включенный во входной поток.

§ 6.11. Дополнительные данные об организации памяти, системных переменных и слов управления

Описанный выше набор слов типичен для современных версий языка Форт среднего уровня сложности. В более сложных версиях могут встречаться различные дополнительные слова, необходимые для контроля над организацией памяти и системными функциями ПЭВМ, а также для управления ПЭВМ и ее периферийным оборудованием.

ОЗУ в форт-системах программирования делится на ряд характерных областей: буферы блоков и электронного квазидиска — зона утилит (программ), стеки (возврата и арифметический), буфер ввода, текстовый буфер, зона слов (словарей, констант и переменных). Границы некоторых из этих областей не фиксированы и могут физически изменяться по мере пополнения форт-системы новыми словами. В связи с этим такие границы (указатели) снабжаются именами и являются системными переменными.

В качестве примера на рис. 6.6 приведена карта ОЗУ, типичная для версии fig-FORTH [32]. Из этой карты можно получить наглядное представление о назначении ряда системных переменных (в том числе и некоторых новых, ранее не упомянутых). Отметим, что в этой версии все системные переменные вырабатывают начальный адрес ячеек ОЗУ, которые хранят числовое значение соответствующей переменной.

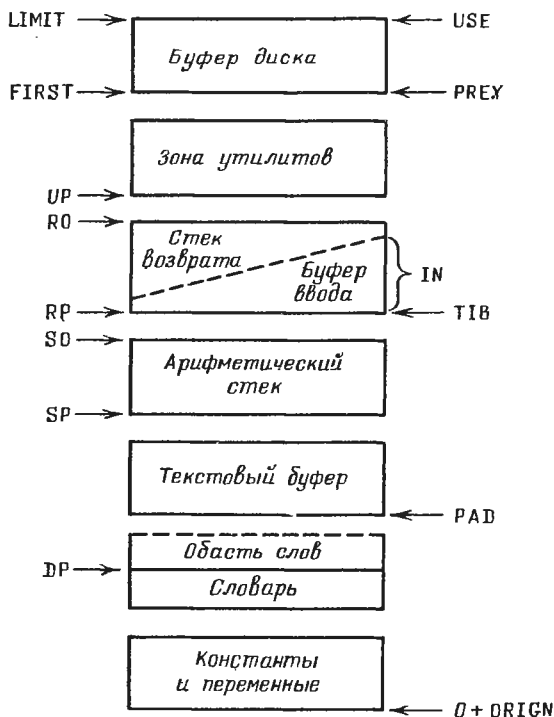


Рис. 6.6. Карта распределения памяти с обозначением системных переменных для версии fig-FORTH

К примеру, как видно из рис. 6.6, системная переменная

DP — — — A

указывает на адрес верхушки словаря (или, точнее, первой свободной ячейки поверх словаря). Таким образом, задав

DP @ .

мы получаем адрес первой свободной ячейки поверх словаря. Отметим, что тот же результат получим, используя специальное слово

HERE U.

: HERE DP @ ;

Ряд системных переменных предназначен для организации словаря и стеков. Для их описания воспользуемся следующими общепринятыми обозначениями:

- NF — поле (т. е. участок ОЗУ) имени словарной статьи (*name file*),
- LF — поле связи словарной статьи (*line file*),
- CF — поле кода словарной статьи (*code file*),
- PF — поле параметров словарной статьи (*parameters file*),
- EF — поле выхода словарной статьи (*exit file*),
- AS — арифметический стек (*arithmetical stack*),
- RS — стек возврата (*return stack*),
- NFA — адрес поля имени словарной статьи,
- LFA — адрес поля связи словарной статьи,
- CFA — адрес поля кода словарной статьи,
- PFA — адрес поля параметров словарной статьи,
- EFA — адрес поля выхода словарной статьи,
- SP — указатель арифметического стека,
- RP — указатель стека возврата.

В табл. 6.16 указаны некоторые системные переменные и функции целочисленных версий языка Форт, которые по тем или иным соображениям не описывались ранее или тесно связаны со спецификой организации памяти.

Отметим, что большинство слов для преобразования адресов словарных статей в версиях FORTH-79 и fig-FORTH не имеют прямых аналогов в версии FORTH-83. В ней приняты следующие слова [20]:

- > BODY CFA _ _ _ PFA (Заменяет CFA на PFA)
- > NAME CFA _ _ _ NFA (Заменяет CFA на NFA)
- > LINK CFA _ _ _ LFA (Заменяет CFA на LFA)
- BODY> PFA _ _ _ CFA (Заменяет PFA на CFA)
- NAME> NFA _ _ _ CFA (Заменяет NFA на CFA)
- LINK> LFA _ _ _ CFA (Заменяет LFA на CFA)
- N> LINK NFA _ _ _ LFA (Заменяет NFA на LFA)
- L> NAME LFA _ _ _ NFA (Заменяет LFA на NFA)

В табл. 6.16 приведена также большая группа дополнительных слов, в основном входящих в состав расширенных версий языка Форт. Часть из них подробно комментируется ниже.

Слова, начинающиеся со знака ?, предназначены для вывода сообщения об ошибке с остановом ПЭВМ. К этой группе относится и слово

n MESSAGE (печать сообщения MSG # n)

Некоторые системные переменные и функции форт-систем программирования

FORTH			Стек	Действие слова
79	fig	83		
			___ PFA	Помещает на вершину стека адрес поля параметров указанного вслед за точкой слова
NFA	NFA	Нет	PFA ___ NFA	Заменяет на вершине стека адрес поля параметров на адрес поля имени
PFA	PFA	Нет	NFA ___ PFA	Заменяет адрес поля имени на адрес поля параметров
CFA	CFA	BODY>	PFA ___ CFA	Заменяет адрес поля параметров на адрес поля кода
EFA	EFA	Нет	PFA ___ EFA	Заменяет адрес поля параметров на адрес поля выхода
LFA	LFA	Нет	PFA ___ LFA	Заменяет адрес поля параметров на адрес поля связи
LATEST	LATEST	Нет	___ NFA	Помещает на вершину стека адрес поля имени последней текущей по записи словарной статьи
VOC — LINK	VOC — LINK	Нет	___ A	Помещает на вершину стека адрес ячейки ОЗУ, содержимое которой указывает на адрес поля связи последнего входа в словарь
Нет	—FIND	Нет	___ PFA b1 или ___ 0	Выделяет указанное вслед за —FIND слово во входном потоке и ищет его в словарях, текущих по поиску. Если слово обнаружено, помещает в стек адрес поля параметров, байт длины его имени и логическое TRUE. В противном случае оставляет логическое FALSE. Длина слова равна b—128
(FIND)	(FIND)	Нет	A1 A2 ___ PFA b1 или A1A2 ___ 0	Ищет в словаре имя с NFA=A2, начиная с адреса A1. Результат аналогичен описанному для слова —FIND

Таблица 6.16 (продолжение)

FORTH			Стек	Действие слова
79	fig	83		
TRAVERSE	TRAVERSE	Нет	$A1 \ n \ _ _ _ \ A2$	По адресу границы текста $A1$ и указателю направления n ($n = +1$ — поиск вправо, $n = -1$ — поиск влево) ищет адрес $A2$ другой границы текста
LIT	LIT	LIT	$_ _ _ \ n$	Помещает на вершину стека число n , скомпилированное сразу после вызова LIT
LITERAL	LITERAL	LITERAL	$n \ _ _ _$	Компилирует число n внутри :-определения (предварительно компилирует вызов директивы LIT)
DLITERAL	DLITERAL	DLITERAL	$d \ _ _ _$	Компилирует число d двойной разрядности в составе :-определения (предварительно компилирует вызов директивы LIT перед каждым разрядом)
BRANCH	BRANCH	BRANCH	$_ _ _$	Обеспечивает выполнение безусловного перехода
0BRANCH	0BRANCH	Нет	$N \ _ _ _$	Обеспечивает выполнение условного перехода (если $N=0$, то управление передается по адресу скомпилированного вслед за 0BRANCH слова)
MESSAGE	MESSAGE	Нет	$n \ _ _ _$	Выводит на индикацию $MSG \ # \ n$ (ошибка с номером n)
?ERROR	?ERROR	Нет	$f \ n \ _ _ _$	Сообщает об ошибке с номером n , если $f=1$
?PAIRS	?PAIRS	Нет	$n1 \ n2 \ _ _ _$	Сообщает об ошибке, если $n1$ и $n2$ не эквивалентны
?LOADING	?LOADING	Нет	$_ _ _$	Сообщает об ошибке, если не исполняется слово LOAD

FORTH			Стек	Действие слова
79	fig	83		
?STACK	?STACK	Нет	— — —	Сообщает об ошибке, если адресное пространство стека вышло за допустимые пределы
?CSP	?CSP	Нет	— — —	Сообщает об ошибке, если позиция стека отличается от значения в CSP
?TERMINAL	?TERMINAL	Нет	— — — <i>f</i>	Помещает на вершину стека $f=1$, если нажата клавиша BREAK, иначе $f=0$
Нет	INP	Нет	<i>n1</i> — — — <i>n2</i>	Помещает на вершину стека значение <i>n2</i> из порта <i>n1</i> , число <i>n1</i> устраняется
Нет	OUTP	Нет	<i>n1</i> <i>n2</i> — — —	Помещает на вершину стека значение <i>n1</i> в порт <i>n2</i>
Нет	PUSHDE	Нет	— — — <i>n</i>	Помещает на вершину стека число <i>n</i> , имеющееся в регистрах DE микропроцессора
Нет	PUSHHL	Нет	— — — <i>n</i>	Помещает на вершину стека число <i>n</i> , имеющееся в регистрах HL микропроцессора
Нет	ID	Нет	NFA — — —	Выводит на индикацию (печать) имя слова с указанным адресом поля имени словарной статьи
Нет	CPU	Нет	— — —	Сообщает тип процессора или ПЭВМ, использующего данную версию FORTH
79-STANDART	Нет	Нет	— — —	Сообщает о принадлежности версии к FORTH-79
Нет	Нет	FORTH-83	— — —	Сообщает о принадлежности версии к FORTH-83
Нет	DPL	Нет	<i>d</i> — — — <i>A</i>	Заменяет на адрес ячейки, в которой хранится позиция разделительной точки (от конца числа), число <i>d</i> двойной разрядности
QUIT	QUIT	QUIT	<i>n</i> — — — <i>n</i>	Останавливает счет и передает контроль терминалу (числа в стеке, введенные до слова QUIT, сохраняются)
ABORT	ABORT	ABORT	<i>n</i> — — —	Останавливает счет, очищает стеки и передает контроль терминалу (у ряда версий ABORT очищает экран)

FORTH			Стек	Действие слова
79	fig	83		
Нет	WARM	Нет	<i>n</i> _ _ _	Останавливает счет, очищает стеки, передает управление терминалу, очищает экран, устраняет подсловарь редактора (введенные пользователем слова сохраняются)
Нет	COLD	Нет	<i>n</i> _ _ _	Действует как WARM, но уничтожает все слова словаря, кроме базового FORTH
Нет BYE	NOP MON	Нет BYE	_ _ _	Нет операции (резервируется только память в ОЗУ) Обеспечивает возврат в операционную систему (или Бейсик)
Нет	UDG	Нет	_ _ _ <i>A</i>	Помещает на вершину стека адрес ячейки ОЗУ, содержимое которой дает адрес ОЗУ графем пользователя
Нет	B/SCR	Нет	_ _ _ <i>n</i>	Помещает на вершину стека число блоков, отведенных под лист редактора
Нет	B/BUFF	Нет	_ _ _	Помещает на вершину стека число ячеек буферов
Нет	LIMIT	Нет	_ _ _ <i>A</i>	Помещает на вершину стека адрес ячейки ОЗУ, в которой содержится верхнее значение адреса буфера диска
Нет	FIRST	Нет	_ _ _ <i>A</i>	Помещает на вершину стека адрес ячейки ОЗУ, в которой хранится адрес дна буфера диска
Нет	C/L	Нет	_ _ _ <i>n</i>	Помещает на вершину стека длину линии листа (в байтах)
Нет	OUT	Нет	_ _ _ <i>A</i>	Помещает на вершину стека адрес переменной, управляющей форматом вывода (содержимое ячейки с адресом <i>A</i> увеличивается на 1 командой EMIT)
Нет	HLD	Нет	_ _ _ <i>A</i>	Помещает на вершину стека адрес ячейки, указывающей последнюю литеру в текстовом буфере

FORTH			Стек	Действие слова
79	fig	83		
Her	WARNING	Her	— — — —	Помещает на вершину стека адрес ячейки ОЗУ — указателя типа используемого накопителя (0 для электронного квазидиска)
Her	LINK	Her	<i>f</i> — — — —	Управляет выводом (при $f=0$ — вывод на дисплей, при $f=1$ — вывод на принтер)
(.) (;CODE)	(.) (;CODE)	(.) (;CODE)	— — — — — — — —	Совместно со словом . обеспечивает печать текста Совместно со словом ;CODE обеспечивает задание словарной статьи в машинных кодах
(+LOOP)	(+LOOP)	(+LOOP)	<i>n</i> — — — —	Обеспечивает организацию конца цикла, компилируется в составе слова +LOOP
(ABORT)	(ABORT)	(ABORT)	— — — —	Обеспечивает прерывание в составе слова ABORT
(DO)	(DO)	(DO)	— — — —	Обеспечивает в составе слова DO организацию начала цикла
(LINE)	(LINE)	(LINE)	<i>n1 n2</i> — — — — <i>Ac</i>	Обеспечивает исполнение слова LINE
(LOOP)	(LOOP)	(LOOP)	— — — —	Обеспечивает исполнение слова LOOP, служит для организации конца циклов
(NUMBER)	(NUMBER)	(NUMBER)	<i>d1 A1</i> — — — — <i>d2 A2</i>	Преобразует текст ASCII с адреса $A1+1$ в соответствии со значением переменной BASE. Помещает на вершину стека число <i>d2</i> и адрес <i>A2</i>
Her DIGIT	(TAPE) DIGIT	Her DIGIT	<i>n</i> — — — — <i>c n1</i> — — — — <i>n1 1</i>	Обеспечивает исполнение слов SAVET и LOADT Преобразует знак с кодом <i>c</i> по ASCII в число с основанием <i>n1</i> . Если результат положительный, помещает на вершину стека <i>n1</i> и значение $f=1$ (TRUE), иначе $f=0$ (FALSE)

Например,

```
25 MESSAGE MSG # 25 OK
```

Это слово рассчитано на включение в состав форт-системы программирования сообщений о специальных видах ошибок, дополнительных к тем, которые уже используются в системе.

Приведем также примеры генерации сообщений об ошибках, создаваемых другими словами:

```
1 7 ?ERROR ?ERROR ? MSG # 7 (генерируется сообщение об ошибке 7, так как флаг  $f=1$  поднят)
```

```
5 5 ?PAIRS OK (сообщения об ошибке нет, так как число  $n1=5$  эквивалентно числу  $n2=5$ )
```

```
2 5 ?PAIRS ?PAIRS ? MSG # 19 OK (появилось сообщение об ошибке, так как числа  $n1=2$  и  $n2=5$  не эквивалентны)
```

Данная группа слов обычно включается в состав словарных статей в рамках :-определения. Например, в словарной статье

```
: D ?STACK 10000 0 DO I ? STACK LOOP ;
```

слово ?STACK используется для контроля переполнения арифметического стека. В слове D ?STACK задан ввод в стек недопустимого количества чисел — 10000. Если задать исполнение этого слова, то получим

```
D ?STACK D ?STACK ? MSG # 7
```

Итак, произойдет останов ПЭВМ с выдачей сообщения об ошибке, как только стек переполнится. Если попробовать исполнить слово DSTACK со словарной статьей

```
: DSTACK 10000 0 DO I LOOP ;
```

не содержащей слова ?STACK, то это приведет к непредсказуемым результатам. Наиболее вероятны ситуации, когда ПЭВМ «зависает» (т. е. перестает реагировать на команды) или «сбрасывает» форт-систему. Причины в том, что бесконтрольное заполнение стека большим количеством чисел ведет к тому, что адресное пространство стека проникает в другие системные области, нарушая их нормальное функционирование.

Еще один пример иллюстрирует применение слова

```
?TERMINAL _____ f
```

которое дает на вершине стека $f=1$, если нажата клавиша BREAK (останов). Слово DEMO со словарной статьей

```
: DEMO 1000 0 DO ?TERMINAL NOT  
IF I.CR THEN LOOP ;
```

обеспечивает печать столбца чисел 0, 1, 2 и т. д. до 999, если не нажата клавиша BREAK. Нажатие клавиши BREAK останавливает печать (хотя цикл DO — LOOP выполняется и при нажатой клавише BREAK).

В расширении fig-FORTH (например, фирмы Abersoft [22]) группа слов (INP, OUTP, PUSHDE, PUSHHL) обеспечивает прямой доступ к портам и внутренним регистрам микропроцессора. Назначение этих слов указано в табл. 6.16. Слово ID . обеспечивает вывод имени слова с указанным адресом поля имени словарной статьи NFA. Это полезно, так как применение слова TYPE с той же целью не всегда удобно: заранее не известна длина имени и последняя буква слова обычно имеет особый код (к нормальному коду прибавляется число 128). Все эти обстоятельства учитываются при реализации слова ID .

Ряд слов позволяет уточнить принадлежность системы к используемой версии языка Форт. Так, слово .CPU выводит текстовый комментарий, указывающий на тип центрального процессора ПЭВМ, на который ориентирована форт-система. Слова 79-STANDART и FORTH-83 выводят текстовый комментарий, указывающий на принадлежность данной версии к стандартной версии FORTH-79 или FORTH-83.

Для облегчения составления программ, ориентированных на работу с числами с фиксированной запятой, служит слово

DPL *d* _ _ _ A

Оно помещает на вершину стека адрес (в версии fig-FORTH) ячейки ОЗУ, в которой размещена позиция разделительной точки числа *d*. Номер позиции отсчитывается с конца числа. Число *d* из стека удаляется. Например:

123.4756 DPL @ . 4 ОК

(точка находится на четвертой позиции с конца числа).

Весьма важен для практической работы ряд слов, реализующих общие директивы управления. Слово QUIT ведет к немедленному прекращению выполнения программы с возвратом управления к терминалу. Однако состояние стека до ввода этого слова сохраняется и после его исполнения.

Слово ABORT также останавливает счет, но очищает стеки. Управление передается терминалу. У ряда систем при этом происходит очистка экрана. Все словари и подсловари сохраняются.

Слово WARM («горячий» старт) действует как слово ABORT с удалением одновременно подсловаря редактора. Однако другие подсловари сохраняются.

Слово COLD («холодный» старт) действует как WARM, но с удалением всех подсловарей, кроме основного FORTH. Слово NOOP

(нет операции) полезно в отладочных ситуациях. Оно резервирует место в ОЗУ, но ничего не выполняет.

Слово `BYE` (иногда `MON`) служит для завершения работы с форт-системой и возврата в исходную операционную систему (монитор или язык Бейсик, встроенный в ПЗУ ПЭВМ). Применение этого слова расширяет возможности ПЭВМ; так, ряд функций (например, распечатку копии изображения с экрана) можно выполнить, пользуясь языком Бейсик или средствами монитора.

Системная переменная `UDG` указывает адрес ячейки, хранящей начальный адрес области ОЗУ, отведенной под хранение графических элементов (графем) пользователя. Применение слова `UDG` описано в § 6.12.

Ряд системных переменных `V/SCR`, `V/BUFF`, `LIMIT`, `FIRST`, `C/L` и др. (см. табл. 6.16) выявляют структуру форт-системы. Слово

`f LINK`

служит для управления устройствами вывода. Если $f=0$, вывод идет на дисплей, а если $f=1$, то вывод — на принтер. У многих версий языка Форт включение принтера производится нажатием специальной клавиши (или специальных клавиш).

Имеется также ряд слов, относящихся к исполняющей системе (*run-time*) языка Форт. Эти слова заключены в круглые скобки и обычно не используются (предназначены для системных программистов). Тем не менее об их функциях полезно иметь представление (см. также табл. 6.16).

Ряд слов-расширений включается в конкретные реализации стандарта FORTH-83. Наиболее важные из них приведены в табл. 6.16. Достаточно полный набор таких слов, а также соответствие их стандарту FORTH-79 даются в работе [2], содержащей перевод стандарта FORTH-83 с английского языка на русский.

Следует отметить, что язык Форт интенсивно развивается и дополняется новыми словами. Важно, чтобы это расширение непременно базировалось на стандартных версиях (FORTH-79 и FORTH-83). Отклонение от этого правила и небрежное отношение к описанию новых слов (например, пропуск хотя бы одного слова) ведет к тому, что обмен программным обеспечением становится практически невозможным и резко снижает ценность разрабатываемых программ.

§ 6.12. Графика и звук целочисленных версий языка Форт

В расширенные целочисленные версии языка Форт, ориентированные на применение в современных персональных компьютерах, включаются типовые возможности машинной графики: задание цветов, построение точек, отрезков прямой и (реже) окружностей.

Типовой набор графических средств соответствует описанному в § 2.14. Ниже отметим некоторые дополнительные возможности, встречаемые в отдельных реализациях языка Форт [22] с развитыми графическими средствами.

Слово

DRAW Δx Δy _ _ _

строит отрезок прямой от исходной точки (x, y) до точки $(x + \Delta x, y + \Delta y)$. Однако у некоторых версий Форта, например у fig-FORTH v. 1.1 для ПЭВМ ZX-Spectrum, применяется слово DRAW в иной интерпретации:

DRAW x y _ _ _

Он соединяет точку с заданными координатами (x, y) с последней, построенной ранее точкой (x_0, y_0) отрезком прямой.

Иногда необходимо слово, строящее отрезок прямой, проходящей через две точки: (x, y) и (x_0, y_0) . С помощью слов PLOT и DRAW можно создать слово

: DDRAW PLOT DRAW ;

выполняющее такие функции:

DDRAW x y x_0 y_0 _ _ _

Для операций с графикой часто применяются специальные системные переменные.

XI _ _ _ A

помещает на вершину стека адрес ячейки, в которой хранится координата x последней построенной точки.

YI _ _ _ A

помещает на вершину стека адрес ячейки, в которой хранится координата y последней построенной точки

INCX _ _ _ n

помещает на вершину стека приращение (*increment*) Δx координаты x при выполнении слова DRAW. Если $\Delta x > 0$, то $n = +1$; если $\Delta x = 0$, то $n = 0$; если $\Delta x < 0$, то $n = -1$.

INCY _ _ _ n

помещает на вершину стека приращение Δy координаты y при выполнении слова DRAW, если $\Delta y > 0$, то $n = +1$; если $\Delta y = 0$, то $n = 0$; если $\Delta y < 0$, то $n = -1$.

Цвет знаков и линий, страницы и бордюра обычно указывается специальными кодами. Однако у ряда версий языка Форт для этого

используются более наглядные системные переменные, имена которых прямо указывают на цвет, например:

BLUE _ _ _ _ *c*

помещает на вершину стека код *c* синего цвета;

RED _ _ _ _ *c*

помещает на вершину стека код *c* красного цвета;

YELLOW _ _ _ _ *c*

помещает на вершину стека код *c* желтого цвета и т. д.

Применение таких системных переменных делает графические программы более ясными и выразительными.

Если таких системных переменных нет, их легко задать. Например, если красный цвет имеет код $c=2$, то системная переменная для задания красного цвета задается словом

: RED 2 ;

Теперь выражение RED PAPER (красная страница) приведет к тому, что цвет страницы будет установлен красным, а RED INK устанавливает красный цвет для знаков. Аналогичным образом можно задать и системные переменные для других цветов. Отметим и альтернативный подход — можно ввести константы, например

2 CONSTANT RED

для задания красного цвета.

Существенное расширение графических возможностей у различных версий языка Форт достигается использованием особых средств Лого-графики. Они описываются в гл. 8 вместе с некоторыми дополнительными возможностями обычной графики.

Большинство расширенных версий для ПЭВМ с графикой имеет специальные средства для задания графических элементов (графем) пользователя. Обычно они создаются в матрице $m \times n$ светлых или темных квадратиков (точек) минимально возможного размера. Графемы могут кодироваться в двоичном или ином другом виде. Например, ниже представлено кодирование графемы с матрицей 8×8 элементов в двоичном, шестнадцатеричном и десятичном кодах:

Двоичный код	Шестнадцатеричный код	Десятичный код
00001000	08	8
00011000	18	24
00110001	31	49
11111110	FE	254
11111110	FE	254
00110001	31	49
00011000	18	24
00001000	08	8

В данном примере в совокупности нулей и единиц двоичного представления нетрудно разглядеть фигуру самолета (она составлена из цифр 1), летящего справа налево. Для представления графём, однако, удобнее более компактные коды — шестнадцатеричный или десятичный. При этом графема из 8×8 элементов может задаваться 8 байтами (числами от 0 до 255 в десятичном представлении).

Кроме кодирования самими кодами, графема должна быть закреплена за определенной клавишей (или за определенными клавишами, нажимаемыми в заданном порядке). В ОЗУ каждой клавише соответствует место в виде блока из 8 байт (считаем, что графема имеет матрицу 8×8). Начальный адрес области ОЗУ, отведенной под хранение графем, помещается на вершину стека специальной системной переменной:

UDG — — — A_0

Обычно адрес A_0 соответствует графеме, закрепленной за клавишей А, $A_0 + 8$ — за клавишей В и т. д. (могут быть и исключения из этого правила).

К примеру, задание графемы, описанной ранее и закрепленной за клавишей А, будет выглядеть следующим образом:

```

0 UDG C! OK
104 UDG 1 + C! OK
204 UDG 2 + C! OK
304 UDG 3 + C! OK
404 UDG 4 + C! OK
504 UDG 5 + C! OK
604 UDG 6 + C! OK
704 UDG 7 + C! OK
* <- UDG

```

Теперь нажатие в графическом режиме работы клавиши А, как видно из приведенного выше примера, выводит не букву А, а силуэт самолета. Он выводится и на печать, как любой другой символ ПЭВМ.

Для задания графем удобно ввести в словарь специальное слово DEFG со списком кодов $n_0 \div n_7$ и порядковым номером n (0 для клавиши А, 1 для клавиши В, 2 для клавиши С и т. д.):

n_0 n_1 n_2 n_3 n_4 n_5 n_6 n_7 n DEFG

Ниже дается словарная статья для задания этого слова и пример задания графемы (фигурки самолета), закрепленной за клавишей С:

```

OK
: DEFG 8 * UDG + DUP 8 + SWAP DO
  I C! LOOP ; OK
6 24 49 254 254 49 24 8 2 DEFG 0
* CR ." * <- UDG
* <- UDG

```

В версии Spectrum-FORTH для массовой и дешевой ПЭВМ ZX-Spectrum имеется аналогичное слово DEF. Оно отличается от описанной только тем, что вместо номера n задается код клавиши, закрепленной за графемой (144 для А, 145 для В и т. д. до 164 для U).

Графемы могут использоваться для создания различных спецзнаков (букв различных алфавитов, математических знаков, отсутствующих в алфавите данной ПЭВМ, сложных образов и т. д.). Графемы широко применяются при создании подвижных изображений, так как скорость перемещения графем на языке Форт велика (намного превосходит таковую для языка Бейсик). Это позволяет реализовать на языке Форт игровые и обучающие программы, машинные фильмы.

Синтез звука в целочисленных версиях языка Форт представлен обычно типовыми словами ВЕЕР или BLEEP. Однако аргументы их задаются в единицах, допускающих представление достаточно большими целыми числами. Например:

ВЕЕР t (мс) f (Гц) — — —

Таким образом, предложение

1000 400 ВЕЕР

создает звуковой сигнал с длительностью 1 с (1000 мс) и частотой f (400 Гц).

Как отмечалось, наряду со словами ВЕЕР и BLEEP часто встречается слово BELL — кратковременное (0,5 — 1 с) включение электрического звонка или зуммера. Такой сигнал обычно используется как предупреждающий об остановке вычислений, возникновении ошибки или неисправности и т. д. У некоторых версий, например у FORTH-79 операционной системы ФОС ПЭВМ «Искра-226», вместо слова BELL используется слово VOICE [4].

ПРОГРАММИРОВАНИЕ НА ЦЕЛОЧИСЛЕННЫХ ВЕРСИЯХ ЯЗЫКА ФОРТ

§ 7.1. Расширение версии fig-FORTH

Любая версия языка Форт обычно содержит 10—20 минимально необходимых слов, из которых затем создается набор остальных базовых слов и дополнительных слов, вводимых пользователем. При этом, как отмечалось, между различными версиями языка Форт могут быть некоторые расхождения по составу базовых слов.

Одной из наиболее распространенных конкретных реализаций языка Форт является версия fig-FORTH. Ниже описано дополнение этой версии рядом новых для нее слов, что делает ее полностью совместимой со стандартной версией FORTH-79. Расширения fig-FORTH представлены листами 7.1 и 7.2.

Лист 7.1

Слова, преобразующие версию fig-FORTH в FORTH-79

```
1 LIST
SCR # 1
0 ( FORTH-79 --> FIG-FORTH )
1 : 0> 0 > ;
2 : 1- 1 - ;
3 : 2- 2 - ;
4 : >IN IN ;
5 : ?DUP -DUP ;
6 : CONVERT (NUMBER) ;
7 : DNEGATE DMINUS ;
8 : NEGATE MINUS ;
9 : SAVE-BUFFERS FLUSH ;
10 : -ROT SWAP >R SWAP R> ;
11 : TUCK SWAP OVER ;
12 : PICK DUP + SP@ + @ ;
13 : 3DUP 3 PICK 3 PICK 3 PICK ;
14 : NDUP 1+ 1 DO DUP PICK SWAP LOOP DROP ;
15 -->
```

ok

Первые девять слов листа 7.1 (0>, 1—, 2—, >IN, ?DUP, CONVERT, DNEGATE, NEGATE и SAVE — BUFFERS) имеют в fig-FORTH полные функциональные аналоги. Поэтому производится только переименование слов (например, слово — DUP получает двойника в виде слова ?DUP). Такой подход позволяет в расширенной версии fig-FORTH выполнять программы, написанные как в версии fig-FORTH, так и FORTH-79. Все описанные далее программы реализованы таким образом.

Слово — ROT (строка 10 листа 7.1) является хорошей иллюстрацией к применению слов $>R$ и $R>$, обеспечивающих ввод и вывод чисел в стек возврата. Это новое слово (его нет в версии FORTH-79) обеспечивает следующую функцию:

```
--ROT n1 n2 n3 _____ n3 n1 n2
```

Таким образом, осуществляется вращение чисел в трех ячейках в обратном (в сравнении с ROT) направлении.

Еще одно новое слово TUCK имеет определение:

```
TUCK n1 n2 _____ n2 n1 n2
```

Таким образом, оно копирует на вершине стека второе от нее число (а не первое, как DUP).

Слово PICK, как и в версии FORTH-79, позволяет поместить на вершину стека n -е от вершины число. Для дублирования сразу трех чисел в стеке используется слово

```
3DUP n1 n2 n3 _____ n1 n2 n3 n1 n2 n3
```

Последнее слово листа 7.1 NDUP обеспечивает дублирование N чисел (N задается на вершине стека). Например, при $N=3$ NDUP эквивалентно 3DUP.

Последующее расширение версии fig-FORTH представлено словами листа 7.2.

Лист 7.2

Расширение версии fig-FORTH

```
2 LIST
SCR # 2
  0 ( NEW WORDS )
  1 : ROLL DUP 1 = IF DROP ELSE DUP 1 DO SWAP R> R
> ROT >R >R
  2 >R LOOP 1 DO R> R> R> ROT ROT >R >R SWAP LOOP
THEN ;
  3 : R@ R ;
  4 : 2ROT 6 ROLL 6 ROLL ;
  5 : D- DNEGATE D+ ;
  6 : D@= OR @= ;
  7 : D< D- SWAP DROP @< ;
  8 : D= D- D@= ;
  9 : DMAX 2OVER 2OVER D< IF 2SWAP THEN 2DROP ;
 10 : DMIN 2OVER 2OVER D< NOT IF 2SWAP THEN 2DROP
;
 11 : ** DUP @= IF DROP DROP 1 ELSE DUP 1 = IF DRO
P ELSE OVER
 12 SWAP 1- @ DO OVER * LOOP SWAP DROP THEN THEN ;
 13 : RLOAD BLK @ + CR 8 SPACES @ OVER .LINE CR 2
SPACES LOAD ;
 14 : ASCII BL WORD HERE 1+ C@ [COMPILE] LITERAL ;
IMMEDIATE
 15 : TRUE 1 ; : FALSE 0 ;      -->
ok
```

Слово ROLL (строки 1 и 2 листа 7.2) аналогично такому же слову в версиях FORTH-79 и FORTH-83. Слово R@ (строка 3) является переименованием слова R. Ряд последующих слов (2ROT, D—, D0=, D<, D=, DMAX, DMIN) обеспечивают совместимость версий fig-FORTH и FORTH-79 для операций над числами двойной разрядности.

Возведение целого числа в целую степень обеспечивает слово:

** $n1\ n2\ ______ n1^{n2}$ если $n2 > 0$
 $n1\ n2\ ______ 1$ если $n2 = 0$

Здесь следует обратить внимание на то, что $n2$ должно быть положительным числом. Если $n2 < 0$, то результат ($1/n1^{n2}$) окажется дробным числом. А это в целочисленных версиях недопустимо.

Два дополнительных слова RLOAD и ASCII (строки 13 и 14 листа 7.2) не относятся к обработке числовых данных. Слово RLOAD аналогично слову LOAD и используется в виде

s RLOAD

где s — номер листа. Однако в отличие от слова LOAD оно при компиляции листов выводит на экран дисплея заголовки листов, заключенных в круглые скобки. Например, если дать

1 RLOAD

получим (при введенных листах 7.1 и 7.2):

(FORTH-79→FIG-FORTH)
 (РАСШИРЕНИЕ FIG-FORTH)
 ... и т. д.

Слово ASCII служит для преобразования указанного вслед за ним символа в код ASCII. Например:

ASCII В . 66 ОК (66 — код ASCII буквы В)

В строке 15 листа 7.2 заданы слова TRUE и FALSE. Они выдают числа 1 и 0:

TRUE _ _ _ 1
 FALSE _ _ _ 0

Ряд других полезных расширений fig-FORTH будет рассмотрен ниже.

§ 7.2. Организация массивов, таблиц и матриц

Простейший одномерный массив с небольшой размерностью может размещаться непосредственно в стеке. Более сложные виды массивов, таблиц и матриц (двумерных массивов) создаются с помощью слова ALLOT, резервирующего в ОЗУ необходимое число ячеек памяти.

Для организации указанных видов данных используются слова, представленные в листе 7.3.

Лист 7.3

Создание массивов, таблиц и матриц

3 LIST

SCR # 3

0 (ARRAYS, TABLES, MATRIXS)

1 : DEPTH SP@ S@ @ SWAP - 2 / ;

2 : .S DEPTH 2 * ?DUP 0= IF CR ." STACK EMPTY "

ELSE

3 SP@ 2+ DUP ROT + SWAP DO CR I ? 2 +LOOP THEN ;

4 : C? C@ . ; 1 2* DUP + ;

5 : ARRAY (n ARRAY name)

6 <BUILDS 2* ALLOT DOES> SWAP 2* + ;

7 : CARRAY (n ARRAY name)

8 <BUILDS ALLOT DOES> + ;

9 : TABLE <BUILDS DOES> SWAP 2* + @ ;

10 : CTABLE <BUILDS DOES> + C@ ;

11 : DARRAY (n DARRAY name)

12 <BUILDS 4 * ALLOT DOES> SWAP 4 * + ;

13 : MATRIX (n m MATRIX name)

14 <BUILDS SWAP 1 + SWAP DUP , * 2* ALLOT DOES>

ROT OVER

15 @ * ROT + 2* + 2+ ; -->

ok

Простейший одномерный массив располагается в виде n чисел в стеке. Слово DEPTH (строка 1 листа 7.3) выводит n на вершину стека. Слово .S (строки 2 и 3) обеспечивают просмотр всех чисел массива, т. е. вывод на индикацию всего содержимого стека (при его сохранении).

В строке 4 определены два вспомогательных, но в ряде случаев полезных слова:

C? A _____ b (выводит содержимое байта b по адресу A)

2* n _____ 2n (обеспечивает умножение n на 2 как $2n = n + n$)

В строках 5 и 6 задано слово ARRAY, обеспечивающее создание одномерных массивов чисел одинарной разрядности с указанными именами по схеме

n ARRAY Имя

Работа с этим словом подробно описывалась в § 6.3. Напоминаем, что слово ARRAY и другие слова этого параграфа создают виды данных в режиме непосредственного исполнения.

В строках 7 и 8 описано слово CARRAY, обеспечивающее задание одномерных массивов чисел от 0 до 255 (байт). Массивы задаются аналогично описанному для слова ARRAY, но занимают вдвое меньший объем в ОЗУ.

Слово TABLE (строка 9) позволяет создавать таблицы с именем по схеме

TABLE Имя n_1, n_2, \dots, n_n

Для вывода i -го числа из таблицы задается выражение

i Имя

Прекрасный и практически полезный пример задания таблицы значений функции $\sin\varphi$ для φ от 0 до 90° описан в [35] и в § 7.6.

Слово DARRAY (строки 11 и 12) служит для создания одномерных массивов чисел двойной разрядности. Оно используется в виде

N ARRAY Имя (задание массива из N чисел d_i)

d_i i Имя 2! (запись d_i в ячейку i)

i Имя 2@ (вызов d_i на вершину стека)

Двумерные массивы (матрицы) размерностью $m \times n$ создаются с помощью слова MATRIX (строки 13—15 листа 7.3). Задание матриц выполняется по схеме

m n MATRIX Имя

Ввод и вывод элементов матриц производятся следующим образом:

m_{ij} j i Имя ! (ввод элемента m_{ij} матрицы)

j i Имя @ (вывод m_{ij} на вершину стека)

j i Имя ? (вывод m_{ij} на индикацию)

Следует еще раз обратить внимание на то, что компоненты данных в этих версиях Форта располагаются не в отдельных и произвольно выбираемых областях ОЗУ (как в FSP88), а в общем поле образующейся словарной статьи. Задание массивов и матриц больших размеров может привести к исчерпыванию памяти, отводимой под словарь.

§ 7.3. Специфика целочисленных вычислений

Для пользователя ПЭВМ, привыкшего к языку Бейсик, необходимость выполнения операций только с целыми числами весьма непривычна и способна создать серьезный психологический барьер в применении целочисленных версий языка Форт. Цель этого параграфа — хотя бы частично сломать этот барьер.

Целочисленные операции в отличие от операций над числами с плавающей запятой выполняются идеально точно со скоростью в 10—20 раз большей, чем у операций с плавающей точкой. Надо согласиться с тем, что это серьезный довод в пользу их применения. Как правило, для хранения целых чисел нужен меньший объем ОЗУ и структура ячеек ОЗУ, отведенных под числа, проще. Наконец, следует помнить, что часто итог вычислений нам нужен просто в целых числах.

Например, при решении задач машинной графики конечный результат (координаты точки на экране дисплея) задается в виде целых

чисел. Даже если получены координаты точки, скажем, (20, 125; 30, 248), то ПЭВМ автоматически представит их в виде (20; 30), отбросив дробные части координат x и y . В то же время скорость выполнения вычислений координат x и y в задачах машинной графики является решающим фактором, дающим целочисленным операциям приоритет.

При выполнении целочисленных операций мы нередко вынуждены ограничивать их точность. Например, числа одинарной разрядности n на языке Форт лежат в пределах от -32768 до $+32767$. Таким образом, минимальная приведенная относительная погрешность задания чисел составляет $1/32767 \approx 3 \cdot 10^{-5}$. Нетрудно заметить, что для многих задач практически такая погрешность вполне достаточна и даже избыточно мала.

Допустим, нам нужно вычислить площадь вырезанного из металлической пластинки, например ножницами, круга:

$$S = \pi D^2 / 4.$$

Ясно, что на практике мы не сможем измерить диаметр D с погрешностью менее 0,5 или 1%. И вряд ли нам нужно знать S с меньшей погрешностью. Однако нас может не удовлетворить необходимость задания D в целых числах, например сантиметрах.

Читатель, вероятно, догадался, что можно просто перейти к миллиметрам или даже к десятым их долям, умножив D (в сантиметрах) на 10 или соответственно на 100. Затем результат надо поделить на 100 или 10000 (так как он задается в квадрате). Но как быть с числом, например

$$\pi = 3,14159265...?$$

Можно ли взять $\pi \cdot 100 = 314$ и вычислить

$$100S = \frac{314}{100} \cdot \frac{D \text{ (см)} \cdot 100 \cdot D \text{ (см)} \cdot 100}{4 \cdot 100}?$$

Действительно, такой путь вполне реален. Допустим, измерено $D = 8,61$ см. Следовательно, $D \cdot 100 = 861$ и

$$100S = \frac{314 \cdot 861 \cdot 861}{40000}.$$

Вычислить такое выражение на языке Форт, используя целочисленную арифметику, можно, если оперировать числами двойной разрядности. Однако надо следить, чтобы промежуточные результаты не вышли за пределы, допустимые для таких чисел.

Прежде чем выполнить это вычисление, отметим еще один изящный путь представления многих констант — через отношение целых чисел. Как видно из табл. 7.1, в ряде случаев это дает поразительно малую погрешность! Так, число $\pi \approx 355/113$ с семью верными знаками. А коли так, то лучше вычислять $10000S$ следующим образом (d в де-

сятых долях миллиметра):

$$10000S = \frac{355 \cdot d \cdot d}{452} = \left(\frac{d \cdot d}{452} \right) \cdot 355.$$

Здесь учтено, что Форт имеет слово */, обеспечивающее вычисление члена в круглых скобках с автоматическим обращением к аппарату арифметики двойной точности.

Итак, окончательно вычисление $10000S$ при $d=8,61$ см = $=861$ (мм) $\cdot 10^{-1}$ можно выполнить следующим образом:

861 861 452 */ 355 U* D. 582200 ОК

Следовательно, $S=582200/10000=58,22$ см² (точное значение $S=58,2232$ см²).

Не очень удобно? Верно, если использовать целочисленный Форт для прямых вычислений, то это не слишком удобно. Более того, проделав эти вычисления для $d=100$ (мм) $\cdot 10^{-1}$, мы обнаружим, что погрешность заметно возросла ($S=7810$ мм² вместо $S=7853$ мм²). Причина этого явления — в погрешности вычисления члена $d \cdot d/452$ с отбрасыванием дробной части результата.

Таблица 7.1

Представление некоторых констант отношением целых чисел

Константа	Представление	Относительная погрешность
π	355/113	$8,5 \cdot 10^{-8}$
$180/\pi$	4068/71	$5 \cdot 10^{-6}$
e	28667/10546	$5,5 \cdot 10^{-9}$
$\sqrt{2}$	19601/13860	$1,5 \cdot 10^{-9}$
$\sqrt{3}$	18817/10864	$1,1 \cdot 10^{-9}$
$\sqrt{10}$	22963/7253	$5,7 \cdot 10^{-9}$
$\ln 2$	6931/9999	$3 \cdot 10^{-5}$
$\sin 45^\circ$	70/99	$5 \cdot 10^{-5}$
$\frac{1000}{16384} \ln 2$	846/19997	$1,2 \cdot 10^{-8}$

Наибольшую точность могут обеспечить целочисленные операции двойной разрядности. Например, если $d=100$ (мм) $\cdot 10^{-1}$, то S вычисляется следующим образом (по формуле $d \cdot d \cdot 355/452$):

100. 100. D* 355 D* 452 D/
D. 7853 ОК

Однако тут мы попадаем в затруднительное положение, если попробуем вычислить S для $d=861$ (мм) $\cdot 10^{-1}$. Результат будет совершенно неверным, так как вычисление $d \cdot d \cdot 355$ дает значение, выходящее за пределы допустимых значений для чисел двойной разрядности.

Эти трудности (они в какой-то мере искусственно подчеркивались) вполне устранимы, если воспользоваться специальным аппаратом цело-

численных вычислений, позволяющим получать любую точность арифметических операций [33]. Однако в этом случае требуется задание специальных слов. Ограничимся описанием часто применяемого слова

$$M*/ d1 n1 n2 _ _ _ d2=d1 \cdot n1/n2$$

которое умножает число двойной разрядности $d1$ на число одинарной разрядности $n1$, а результат делит на другое число одинарной разрядности $n2$ (получаем число двойной разрядности $d2=d1 \cdot n1/n2$). Если это слово отсутствует, его можно задать (см. лист 7.10 в § 7.9).

Так, введя специальное слово S со словарной статьей

: S DUP U* 355 452 M*/ D. ;

будем вычислять S по заданным d (мм) $\cdot 10^{-1}$:

100 S 7853 ОК (точное значение $S=0,785398$ см²)

861 S 582232 (точное значение $S=58,222$ см²)

Как видно, результаты теперь получаются вполне удовлетворительными.

Аналогичный подход используется при вычислении значений функций. Например, вычисляя $\sin x$ при x в радианах (допустим от $-\pi/2 = -1,5708$ до $+\pi/2 = 1,5708$), целесообразно аргумент задавать умноженным на 10000 (т. е. 0,5 задаем как 5000, $-1,5708$ как -15708). Так как значения $\sin x$ лежат в отрезке $[-1, 1]$, то результат также следует выдавать умноженным на 10000. К примеру, $\sin 0,5 = 0,4794$ на Форте вычисляется как 5000 SIN, что дает в результате 4794.

Этот подход широко используется в практике вычислений на целочисленных версиях языка Форт. Примеры его применения даны ниже (§ 7.8, 7.9).

§ 7.4 Целочисленные операции и функции

Над целыми числами могут производиться различные специальные операции. Одной из них является факторизация — разложение целого числа на простые множители. Она реализуется с помощью слова FACTORS, например:

$$5600 \text{ FACTORS} = 1 * 2 \uparrow 5 * 5 \uparrow 2 * 7$$

Это означает, что число 5600 можно разложить на множители в виде:

$$5600 = 1 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 5 \cdot 5 \cdot 7 = 1 \cdot 2^5 \cdot 5^2 \cdot 7$$

Соответственно

$$5601 = 1 * 3 * 1867$$

В листе 7.4 содержатся словарные статьи для слов PRINT, TOTNE, CVECTOR и FACTORS, реализующих факторизацию 33.

Некоторые целочисленные действия и функции

1 LIST

SCR # 1

```

0 ( FACTOR, HCF, LCM )
1 : PRINT -DUP IF ROT ." * " . DUP 1 > IF ." ^"
. ELSE
2 DROP THEN DROP ELSE DROP DROP THEN ;
3 : TOTHE 0 BEGIN >R OVER OVER /MOD R> ROT 0= WH
ILE
4 1+ >R >R SWAP DROP R> SWAP R> REPEAT PRINT
;
5 : CVECTOR <BUILDS 0 DO C, LOOP DOES> + C@ ;
6 31 29 23 19 17 13 11 7 8 CVECTOR TRAIL
7 : FACTORS ." =1 " 2 TOTHE 3 TOTHE 5 TOTHE
8 DUP 1 = IF CR DROP EXIT THEN DUP
9 0 DO 8 0 DO I TRAIL J + TOTHE LOOP
10 DUP I < IF LEAVE THEN 30 +LOOP DROP ;
11 : HCF BEGIN SWAP OVER MOD -DUP 0= UNTIL ;
12 : .HCF HCF CR ." HCF=" . ;
13 : LCM >R 0 OVER R SWAP R> HCF M*/ ;
14 : .LCM LCM CR ." LCM=" D. ;
15 -->

```

Слова PRINT, TOTHE и CVECTOR являются вспомогательными для реализации основного слова:

FACTORS n _ _ _

которое обеспечивает факторизацию целых чисел n одинарной разрядности (до 32767). Примеры факторизации были приведены выше.

Слово

HCF $n1$ $n2$ _ _ _ $n3$

вычисляет и оставляет на вершине стека число $n3$ — наименьший общий множитель двух целых чисел $n1$ и $n2$ (HCF — сокращение в виде первых букв слов *Highest Common Factor*).

Слово

.HCF $n1$ $n2$ _ _ _ (вывод HCF= $n3$)

выводит на индикацию значение HCF.

Пример.

```

125 15 .HCF
HCF=15 OK

```

Слово

LCM $n1$ $n2$ _ _ _ $d=n1 \cdot n2 / HCF$

вычисляет и оставляет на вершине стека наименьшее общее кратное (Lowest Common Multiple) двух целых чисел $n1$ и $n2$.

Слово

.LCM $n1$ $n2$ _ _ _ (вывод $d=n1 \cdot n2 / HCF$)

выводит значение наименьшего общего кратного двух целых чисел n_1 и n_2 на индикацию.

Пример.

```
125 15 .LCM
LCM=375 OK
```

К распространенным целочисленным функциям относится факториал:

```
0! = 1,
n! = 1 · 2 · ... · (n - 1) · n.
```

Обычно факториал вычисляется прямо по этим формулам с применением цикла DO—LOOP. В листе 7.5 содержатся слова N! и ND!, вычисляющие факториал при $n \geq 8$ (слово N!, результат — число одинарной разрядности) и при $d \geq 12$ (слово ND!, результат — число двойной разрядности).

Лист 7.5

Слова N! и ND! для вычисления факториала

```
2 LIST
SCR # 2
0 ( CALCULATION FACTORIAL )
1 : N! ( N---N! N<=8 )
2     1 SWAP 1+ 1
3     DO
4         I *
5     LOOP
6 ;
7 : ND! ( N---D=N! N<=12 )
8     .1 ROT 1+ 1
9     DO
10        I ROT ROT UM*
11    LOOP
12 ;
13 -->
14
15
ok
```

Примеры.

```
0 N! . 1 OK
4 N! . 24 OK
0 ND! D. 1 OK
10 ND! D. 3628800 OK
```

Факториал может вычисляться также с применением рекурсивной процедуры (см. § 7.10). Факториалы можно использовать для вычисления числа размещений из n элементов по m :

$$A_n^m = n! / (n - m)!$$

и числа сочетаний из n элементов по m :

$$C_n^m = n! / ((n-m)! m!).$$

Например, если $n=10$ и $m=5$, то $A_{10}^5 = 10!/5!$, то для вычисления A_{10}^5 достаточно ввести слова:

10 ND! 5 ND! D/ D. 30240

Аналогичным образом легко вычислить C_n^m .

§ 7.5. Сортировка данных

Одной из довольно утомительных операций является сортировка данных, например чисел. Попробуйте расположить в порядке возрастания 100 — 200 произвольных чисел и вы сразу убедитесь в этом. Поэтому сортировка данных — одна из наиболее часто встречающихся операций, реализуемых на ПЭВМ.

Разработано большое число различных алгоритмов сортировки данных. Ограничимся описанием двух из них, предназначенных для сортировки чисел в одномерном массиве, характеризующемся начальным адресом A (адрес первой ячейки) и числом ячеек n (каждая ячейка вмещает число одинарной разрядности).

Один из самых простых алгоритмов сортировки называется алгоритмом BUBBLE-SORT (или сокращенно BBL-SORT) [7]. Это название происходит от слова *bubble* — пузырек (как более легкое вещество пузырька всплывает на

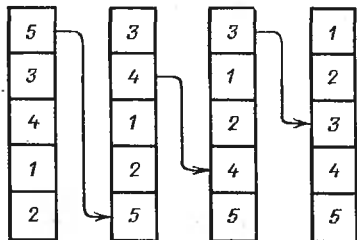


Рис. 7.1. Сортировка чисел по алгоритму BBL-SORT

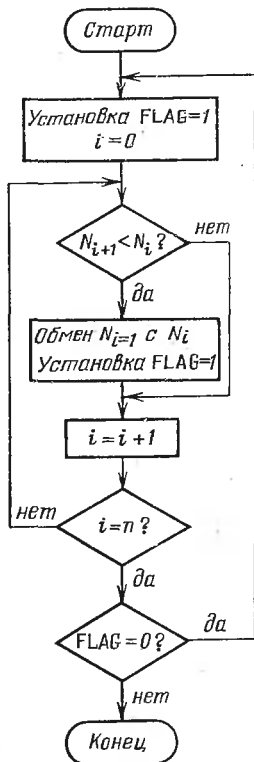


Рис. 7.2. Алгоритм сортировки BBL-SORT

поверхность жидкости). Рисунок 7.1 поясняет механизм сортировки, а рис. 7.2 иллюстрирует графически алгоритм сортировки [35].

В версии fig-FORTH сортировка по этому алгоритму реализуется словом

BBL—SORT A_n — — —

Словарная статья BBL—SORT представлена в листе 7.6.

Лист 7.6

Слова BBL—SORT, BSORT, <CMOVE и ISORT для сортировки данных в массиве

```
3 LIST
SCR # 3
  0 ( SORTING OPERATIONS )
  1 : BBL—SORT DUP + OVER + 2 -
  2 SWAP >R >R BEGIN 1 R> R OVER
  3 >R DO I 2 + @ I @ < IF I 2 +
  4 DUF @ I DUP @ 4 ROLL ! !
  5 DROP 0 THEN 2 +LOOP UNTIL R> R> DROP DROP ;
  6 : BSORT DUP + OVER + SWAP >R >R BEGIN 1 R> 2 -
  R OVER >R
  7 DO I 2 + @ I @ < IF I 2 + DUP @ I DUP @ 4 ROLL
  ! !
  8 DROP 0 THEN 2 +LOOP UNTIL R> R> DROP DROP ;
  9 : <CMOVE DUP ROT + SWAP ROT 1- DUP ROT + DO 1-
  I C@
  10 OVER C! -1 +LOOP DROP ;
  11 : ISORT DUP + OVER + OVER 2+ DO I @ DUP I 2 -
  @ < IF
  12 0 3 PICK 2 - I 2 - DO DROP DUP I @ < IF I ELSE
  I 2 +
  13 LEAVE THEN -2 +LOOP DUP DUP
  14 2 + I 3 PICK - <CMOVE !
  15 ELSE DROP THEN 2 +LOOP DROP ; -->
ok
```

В листе 7.6 дана также словарная статья для слова BSORT, реализующая несколько иную модификацию алгоритма BBL—SORT, имеющую на 20 — 25% меньшее время сортировки.

В алгоритме BBL—SORT на каждом шаге идет поиск наибольшего числа по всему массиву, после чего оно помещается в низ массива. Время сортировки можно значительно сократить, используя алгоритм INSERTION—SORT (или сокращенно ISORT). Схема поиска для этого алгоритма показана на рис. 7.3. В процессе поиска каждый раз уменьшается на 1 количество чисел в массиве поиска. Например, если $n=5$, то вначале поиск идет по массиву из четырех чисел. Наибольшее число помещается на вершину массива и исключается (*insertion*) из поиска. Затем поиск идет по массиву из трех чисел и т. д.

Реализация этого алгоритма также дана в листе 7.6 — слова <CMOVE и ISORT.

Слово

<CMOVE $A_1 A_2 n$ — — —

копирует n байтов с начальным адресом A_1 в область ОЗУ с начальным адресом A_2 . Направление копирования обратное, т. е. адреса байтов идут от больших значений к меньшим.

Слово

ISORT $A n$ _ _ _ _

сортирует данные в массиве, начинающемся с адреса A и имеющем n чисел одинарной разрядности.

Для проверки этих слов можно воспользоваться листом 7.7.

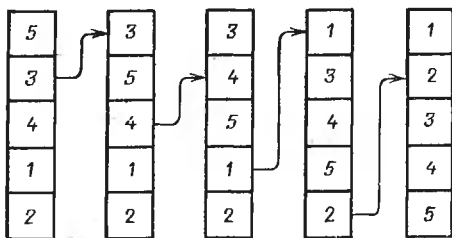


Рис. 7.3. Сортировка чисел по алгоритму ISORT

Лист 7.7

Задание массива VECTOR из 200 чисел, ввод чисел от 199 до 0 словом VINP и контроль 200 чисел в массиве V?

```

4 LIST
SCR # 4
0 ( SORTING CONTROL )
1 200 ARRAY VECTOR
2 : VINP ( INPUT 200 NUMBERS )
3   200 0 DO
4     200 I - I VECTOR !
5   LOOP
6 ;
7 : V? ( PRINT 200 NUMBERS )
8   200 0 DO
9     I VECTOR ?
10  LOOP
11 ; -->
12
13
14
15
ok

```

Вначале в листе 7.7 задан массив из 200 чисел с именем VECTOR. Далее слово VINP обеспечивает запись 200 чисел от 199 до 0 (т. е. в обратном порядке) в ячейки массива VECTOR. Слово V? выводит на индикацию все числа.

Так, выполнив компиляцию слов листов 7.6 и 7.7 и введя

VINP V?

получим на экране числа

199 198 197 196 ... 2 1 0

Теперь введем слова

BBL—SORT V?

Спустя некоторое время (обычно около 1—2 мин для ПЭВМ с 8-разрядным микропроцессором) на экран будут выведены числа

0 1 2 3 4 5 6 ... 197 198 199

Отсюда видно, что произошла сортировка чисел в массиве и они оказались расположенными в прямом порядке — от меньших чисел до больших. Повторив эти операции с применением слов BSORT и ISORT, можно убедиться в аналогичном действии и этих слов.

Время сортировки зависит от алгоритма и скорости вычислений ПЭВМ. Если принять за 1 время выполнения сортировки с помощью слова BBL—SORT, то время сортировки с помощью слова BSORT составит 0,88, а ISORT — 0,31. Таким образом, алгоритм INSERTION—SORT обеспечивает существенный выигрыш во времени сортировки.

Еще более эффективны специальные алгоритмы сортировки. Большое число форт-программ сортировки различных типов данных описано в книге [7]. Сортировка данных полезна и сама по себе, поскольку отсортированными данными удобно пользоваться (например, сразу выявляя минимальное и максимальное числа). Многие виды статистических вычислений и специальной обработки данных значительно облегчаются и выполняются с большей точностью, если данные предварительно подвергались сортировке.

§ 7.6. Операции над числами с фиксированной точкой

Остановимся на еще одном очень распространенном виде целочисленных расчетов — финансовых. Всем хорошо известно правило таких расчетов — доли копейки не учитываются. Следовательно, расчет денежных средств в копейках сводится к исключительно целочисленным операциям. А чтобы получить результат в рублях и копейках, достаточно отделить точкой или запятой цифры рублей от копеек. Фактически это означает переход к арифметике с фиксированной точкой (операции FIX-типа).

Форт обладает всеми возможностями для довольно простой организации таких операций. Необходимые для этого дополнительные слова представлены в листе 7.8.

Лист 7.8

Арифметика с фиксированной точкой, ввод чисел

```

4 LIST
SCR # 4
  0 ( FIX-POINT NUMBERS OPERATIONS )
  1 : FIX? CR SWAP OVER DABS <# # # 46 HOLD #S SIG
N #> TYPE CR ;
  2 0 VARIABLE AA 0 VARIABLE BB
  3 0 VARIABLE CC 0 VARIABLE DD
  4 : D* AA ! BB ! CC ! DD ! DD @ BB @ U* DD @ AA
@ U* DROP
  5 + CC @ BB @ U* DROP + ;
  6 : D/ DROP M/ SWAP DROP S->D ;
  7 : FIX+ D+ ; : FIX- D- ;
  8 : FIX* D* DUP >R DABS 100 M/MOD R> 0< IF DMINU
S THEN ROT DROP ;
  9 0 VARIABLE DIV
 10 : FIX/ DROP DUP DIV ! M/ 100 * SWAP DIV @ 100
/ / + S->D ;
 11 : INTEXT CR ." ВВЕДИТЕ " QUERY 1 TEXT ;
 12 : DINP INTEXT PAD NUMBER ;
 13 : SINP INTEXT PAD NUMBER DROP ;
 14 : FIXINP DINP ;
 15 : M+ S->D D+ ; : UM* >R OVER U* ROT R> * + ;
-->
ok

```

Рассмотрим эти слова более подробно.

Слово

FIX? *d* _ _ _

вставляет фиксированную десятичную точку в число *d*, отделяя две последние цифры справа. Число *d* может быть любого знака. К примеру, если ввести

3.14 FIX? 3.14 OK

.314 FIX? 3.14 OK

31.4 FIX? 3.14 OK

314. FIX? 3.14 OK

В последнем случае действует общее правило задания целых чисел двойной разрядности — они помечаются точкой в любом (!) месте. Слово FIX? преобразует такое число в число, у которого точка всегда отделяет только две последние цифры.

Полезно сразу приучить себя к тому, что в операциях с фиксированной точкой нужно вводить все цифры — как до разделительной точки, так и обе цифры после нее, даже если они нули. Так, верным будет ввод чисел 3 и 5,1 в виде

3.00 OK (ввод числа 3)

5.10 OK (ввод числа 5,1)

Отметим, что число цифр после фиксирующей точки в слове FIX? можно сделать и другим (не только 2). Оно определяется количеством знаков # перед словом HOLD. Наконец, можно вместо разделительной

точки (так принято в большинстве языков программирования) использовать запятую или любой иной спецзнак. Для этого вместо кода 46 точки надо указать код ASCII этого знака. Однако, поскольку ввод чисел двойной разрядности предусмотрен заведомо с указанием точки, этой возможностью лучше не пользоваться (применение разделительной точки вместо запятой принято и в ряде других языков программирования, включая Бейсик).

В строках 2 и 3 листа 7.8 определены четыре вспомогательные переменные AA, BB, CC и DD, которые используются в дальнейшем.

Слово

$D* d1 d2 _ _ _ d1 \cdot d2$

обеспечивает умножение двух чисел двойной разрядности. Результат — также число двойной разрядности.

Слово

$D/ d1 d2 _ _ _ d1/d2$

обеспечивает деление двух чисел двойной разрядности. Результат — число двойной разрядности.

Эти слова заданы в строках 4, 5 и 6. Они не имеют непосредственного отношения именно к операциям с фиксированной точкой, т. е. могут использоваться как элементы целочисленных операций.

Слова

$FIX+ d1 d2 _ _ _ d1 + d2$

$FIX- d1 d2 _ _ _ d1 - d2$

заданы в строке 7. Они, в сущности, являются просто переименованными словами $D+$ и $D-$. Такое переименование необходимо для унификации имен слов, предназначенных для операций над числами с фиксированной точкой. (в их состав введено обозначение FIX).

Пример.

2.51 3.42 FIX+ FIX? 5.93 OK

2.51 3.42 FIX- FIX? -0.91 OK

В строке 8 определена операция умножения

$FIX* d1 d2 _ _ _ d1 \cdot d2$ (с отбросом двух последних цифр)

умножает $d1$ на $d2$ по правилам арифметики с фиксированной точкой. Два последних знака числа отбрасываются.

Примеры.

2.20 3.00 FIX* FIX? 6.60 OK

-2.25 44.00 FIX* FIX? -99.00 OK

12.34 2.50 FIX* FIX? 30.85 OK

В строке 9 листа 7.8 определена вспомогательная переменная

DIV, а в строке 10 — операция деления.

Слово FIX/ d1 d2 — — — d1/d2

Деление также проводится по обычным правилам арифметики с фиксированной точкой.

Примеры.

1.00 3.00 FIX/ FIX? 0.33 ОК

—111.00 —2.12 FIX/ FIX? 52.38 ОК

При необходимости операции FIX* и FIX/ могут быть перестроены под иное количество цифр после фиксирующей точки. Для этого надо изменить константу 100 в этих словах (например, на 10, если отделяется одна цифра, или на 1000, если три).

§ 7.7. Интерактивный ввод данных

На языке Бейсик общепринято осуществлять ввод исходных данных по запросу ПЭВМ. Для этого используется оператор

INPUT " ВВЕДИТЕ " ; X

В кавычки введен текст комментария запроса. В нашем случае исполнение такого оператора приведет к появлению запроса

ВВЕДИТЕ

после чего нужно ввести число и нажать клавишу перевода строки. Переменной X будет присвоено введенное значение.

На языке Форт более привычным является ввод числа на вершину стека. Затем с ним можно делать что угодно, например присвоить в качестве значения переменной X или просто использовать для последующих вычислений.

До сих пор мы спокойно обходились без комментариев при вводе. Однако надо признать, что в сложных программах (особенно обучающих), диалог при вводе чисел или команд бывает принципиально необходим. На Форте он реализуется несколько более изощренными и сложными методами, чем на Бейсике. В строках 11—14 листа 7.8 дан набор слов для организации ввода с запросом.

Слово

INTEXT Текст — — —

(от *input text* — ввод текста), определенное в строке 11, обеспечивает вывод запроса

ВВЕДИТЕ

и затем ввод любого текста во входной буфер (см. слова QUERY I TEXT). Текст запроса можно изменить, например, на

ВВЕДИТЕ ЗНАЧЕНИЕ

и т. д.

После запроса вводится число, которое поступает во входной буфер в виде символического текста. Последующие слова с помощью слова NUMBER преобразуют этот текст в число, размещаемое на вершине стека.

Слово

DINP _ _ _ d

по запросу размещает введенное число как число двойной разрядности на вершине стека.

Слово

SINP _ _ _ n

по запросу размещает введенное число как число одинарной разрядности на вершине стека. В действительности введенное число имеет двойную разрядность, но старший разряд с нулевым значением удаляется с помощью слова DROP.

Слово

FIXINP _ _ _ d

обеспечивает ввод числа в формате FIX по запросу и размещение его на вершине стека. В сущности, FIXINP — просто переименование слова DINP.

Пример выполнения операций с вводом чисел по запросу:

DINP

ВВЕДИТЕ 12345. ОК

DINP

ВВЕДИТЕ 250. ОК

D + D. 12595. ОК

FIXINP

ВВЕДИТЕ 3.14 ОК

FIXINP

ВВЕДИТЕ 2.00 ОК

FIX* FIX? 6.28 ОК

В строке 15 листа 7.8 задано еще два дополнительных слова.

Слово

M + d n _ _ _ $d + n$

складывает число двойной разрядности с числом одинарной разрядности. Результат $d + n$ есть число двойной разрядности.

Слово

UM* d u _ _ _ $d \cdot u$

обеспечивает умножение числа двойной разрядности d на число без знака одинарной разрядности u . Результат $d \cdot u$ — число двойной разрядности.

Эти слова не имеют отношения к основному содержанию этого параграфа и введены в лист 7.8 с целью его полиого заполнения и применения в дальнейшем.

§ 7.8. Быстрое вычисление синуса и косинуса выбором из таблицы

Если угол φ тригонометрических функций задан в градусах (от 0 до 360°), то при целочисленных операциях целесообразно ограничить вычисление синуса и косинуса значениями $\varphi = 0, 1^\circ, 2^\circ, \dots, 359^\circ$, т. е. при дискретности φ в 1° . В частности, необходимость в этом возникает при реализации быстрой Лого-графики, где обычно вполне достаточно задания углов с подобной точностью.

Такое представление угла позволяет находить синус по заранее составленной таблице. Ниже дан фрагмент ее начала:

φ , град	0	1	2	3	4	...
$(\sin \varphi) \cdot 10000$	0	175	349	523	698	...

Это способ, по-видимому, является самым быстрым из всех возможных, так как вычисление $(\sin \varphi) \cdot 10000$ сводится просто к выбору заранее вычисленного значения из табличного набора. Например, если таблице присвоено имя SINTAB, то для получения $(\sin 30^\circ) \cdot 10000$ на вершине стека достаточно слова

```
30 SINTAB @
```

Однако в этом случае нам придется хранить 360 чисел двойной разрядности. С точки зрения необходимых затрат памяти это явно нерационально. Тем более что функция $\sin \varphi$ имеет одно и то же по модулю значение для четырех углов в пределах изменения φ от 0 до 360° . Это означает, что $\sin \varphi$ достаточно определить на отрезке $[0, 90^\circ]$. При этом имеем:

φ	Значение $\sin \varphi$
$0-90^\circ$	$\sin \varphi$
$90-180^\circ$	$\sin (180^\circ - \varphi)$
$180-270^\circ$	$-\sin (\varphi - 180^\circ)$
$270-360^\circ$	$-\sin (360^\circ - \varphi)$

Привести аргумент синуса в нужный диапазон значений $[0, 90^\circ]$ можно с помощью выражений IF—ELSE—THEN. Поскольку соответствующие команды выполняются быстро, это не приведет к большому увеличению времени счета, но позволит хранить только 90 значений $(\sin \varphi) \cdot 10000$. Таким образом, затраты памяти уменьшатся почти в

4 раза (почти, поскольку часть памяти займет реализация этих процедур).

Функция $\cos \varphi$ повторяет функцию $\sin \varphi$ со сдвигом φ на 90° . Таким образом:

$$\begin{aligned}\cos \varphi &= \sin (\varphi + 90^\circ) && \text{при } 0^\circ < \varphi < 270^\circ, \\ \cos \varphi &= \sin (\varphi - 270^\circ) && \text{при } 270^\circ < \varphi < 360^\circ.\end{aligned}$$

Следовательно, вычисление $\cos \varphi$ легко сводится к вычислениям $\sin \varphi$. Все эти операции реализованы в листе 7.9.

Лист 7.9

Быстрое вычисление $\sin \varphi$ и $\cos \varphi$ по таблице $\sin \varphi$ (φ в градусах)

5 LIST

SCR # 5

```
0 ( INTEGER SIN, COS )
1 TABLE SINTAB 0 , 175 , 349 , 523 , 698 , 872 ,
1045 , 1219 ,
2 1392 , 1564 , 1736 , 1908 , 2079 , 2250 , 2419
, 2588 , 2756 ,
3 2924 , 3090 , 3256 , 3420 , 3584 , 3746 , 3907
, 4067 , 4226 ,
4 4384 , 4540 , 4695 , 4848 , 5000 , 5150 , 5299
, 5446 , 5592 ,
5 5736 , 5878 , 6018 , 6157 , 6293 , 6428 , 6561
, 6691 , 6820 ,
6 6947 , 7071 , 7193 , 7314 , 7431 , 7547 , 766
0 , 7771 , 7880 ,
7 7986 , 8090 , 8191 , 8290 , 8387 , 8480 , 857
2 , 8660 , 8746 ,
8 8829 , 8910 , 8988 , 9063 , 9135 , 9205 , 927
2 , 9336 , 9397 ,
9 9455 , 9511 , 9563 , 9613 , 9659 , 9703 , 974
4 , 9781 , 9816 ,
10 9848 , 9877 , 9903 , 9926 , 9945 , 9962 , 997
6 , 9986 ,
11 9994 , 9998 , 10000 ,
12 : SIN ( SIN*10000 FOR 0<ANGLE<360 DEG ) DUP 27
0 >
13 IF 360 SWAP - SINTAB MINUS ELSE DUP 180 > IF 1
80 - SINTAB
14 MINUS ELSE DUP 90 > IF 180 SWAP - THEN SINTAB
THEN THEN ;
15 : COS ( COS*10000 ) DUP 270 > IF 270 - ELSE 90
+ THEN SIN ; -->
ok
```

В строках 1—11 задана таблица из 90 значений $(\sin \varphi) \cdot 10000$ для φ от 0 до 89° . В строках 12—14 выполняется вычисление $(\sin \varphi) \cdot 10000$ по описанному выше алгоритму, а в строке 15 — вычисления косинуса.

Примеры.

30 SIN . 5000 OK (точно 0,5)

45 COS . 7071 OK (точно 0,707107)

Отметим, что при необходимости можно вычислять $\sin \varphi$ и $\cos \varphi$ при φ , изменяющемся с меньшей дискретностью. Для этого следует воспользоваться интерполяцией (линейной или даже квадратичной). Другие способы вычисления элементарных функций рассматриваются в § 7.9.

§ 7.9. Вычисление других элементарных функций

Продолжим описание приемов вычисления элементарных функций с применением аппарата целочисленной арифметики. В некоторых случаях нужные функции вычисляются через другие, ранее определенные. Например, тангенс угла φ можно выразить как

$$\operatorname{tg} \varphi = \sin \varphi / \cos \varphi. \quad (7.1)$$

Однако простота такого вычисления оказывается обманчивой. Дело в том, что значения $\operatorname{tg} \varphi$ лежат в интервале $-\infty < \operatorname{tg} \varphi < \infty$.

Даже если ограничиться представлением тангенса числами двойной разрядности, вычисляя функцию DTAN как $(\operatorname{tg} \varphi) \cdot 10000$, практическая реализация вычисления по формуле (7.1) осложняется чрезмерными значениями промежуточных результатов. Поэтому для реализации вычислений $(\operatorname{tg} \varphi) \cdot 10000$ приходится вводить ряд новых слов (отметим, что они полезны и для многих других применений). Эти слова и слова для вычисления ряда других элементарных функций представлены в листе 7.10.

Лист 7.10

Вычисление некоторых элементарных функций

```
6 LIST
SCR # 6
0 ( INTEGER DTAN, ASINR, EXP, DSQR )
1 : UN* >R OVER U* ROT R> * + ; : U/ U/MOD ;
2 : UN/ SWAP OVER /MOD >R SWAP U/ SWAP DROP R> ;
3 : M*/ 2DUP XOR SWAP ABS >R SWAP ABS >R OVER XO
R -ROT DABS SWAP
4 R U* ROT R> U* ROT 0 D+ R U/ -ROT R> U/ SWAP D
ROP SWAP ROT D+- ;
5 0 VARIABLE FI : DTAN FI ! 10000. FI @ SIN FI
@ COS M*/ ;
6 : 3PICK 6 SP@ + @ ; 10000 CONSTANT 10K
7 : +DX M*/ 10K 0 D+ 3 PICK 10K M*/ ;
8 : +SX M*/ 10K 0 D+ DROP SWAP DROP ;
9 : ATNR DUP DUP M* 10K M/ SWAP DROP DUP S->D 3P
ICK 3PICK 3PICK
10 3 14 +DX 10 9 +S} >R 64 735 +DX 7 9 +SX M* R>
M/ SWAP DROP ;
11 : ASINR DUP DUP M* 10K M/ SWAP DROP DUP S->D B
757 17280 +DX
12 600 10000B +DX 18 40 +DX 1 6 +SX M* 10K M/ SWAP
DROP ;
13 : EXP DUP S->D 4 17 +DX 17 48 +DX 192 383 +DX
383 384 +SX ;
```

14 : DSQR 127 7 0 DO 3DUP M/MOD ROT DROP ROT 0 D+
 2 U/ SWAP
 15 DROP LOOP >R DROP DROP R> ; -->

ok

Слово

UN* *u d1* — — — *d2*

обеспечивает умножение числа *u* двойной разрядности без знака на число *d1* двойной разрядности. Результат — число *d2* двойной разрядности.

Слово

UN/ *u d1* — — — *d2*

обеспечивает деление числа *u* двойной разрядности без знака на число *d1* двойной разрядности. Результат — число *d2* двойной разрядности.

Слово

M*/ *d1 n1 n2* — — — *d2*

обеспечивает умножение числа *d1* двойной разрядности на число *n1* одианрной разрядности и деление результата на число *n2* одианрной разрядности. Результат — число *d2* двойной разрядности.

Слово

FI . (Задание переменной FI)

Слово

DTAN φ — — — $\text{tg } \varphi \cdot 10000$

вычисляет $\text{tg } \varphi \cdot 10000$ на основании соотношения (7.1). Результат — число двойной точности. Например:

30 DTAN D. 5773 ($\text{tg } 30^\circ = 0,57735$)

Для вычисления арктангенса могут использоваться разложения в ряд (результат в радианах):

$$\text{arctg } x = x - x^3/3 + x^5/5 - x^7/7 + \dots$$

или (для $x < 1$) рациональные полиномиальные приближения (аппроксимация Падэ):

$$\text{arctg } x = \frac{x + 7x^3/9 + 64x^5/945}{1 + 10x^2/9 + 5x^4/21} \quad (7.2)$$

В строках 6—10 листа 7.6 задано вычисление $\text{arctg } x$ по формуле (7.2). Оно реализовано следующими словами [33]:

3PICK (тройное PICK)

1 + DX *x d1 n1 n2* — — — *x d2*

1 + SX *x d1 n1 n2* — — — *n3*

которые являются вспомогательными и используются для вычисления значения $\text{arctg } x \cdot 10000$ с помощью слова

$$\text{ATNR } x \text{ --- } 10000 \cdot \text{arctg } x$$

Пример.

$$10000 \text{ ATNR } . 7855 \text{ (arctg } 1 = 0,785398).$$

Разложение в ряд для $\text{arcsin } x$ имеет вид: $x + \frac{1}{6}x^3 + \frac{3}{40}x^5$.

В строках 11 и 12 задано слово

$$\text{ASINR } 10000 \cdot x \text{ --- } 10000 \cdot \text{arcsin } x$$

реализующее вычисление арксинуса по этому разложению.

Пример.

$$5000 \text{ ASINR } . 5232 \text{ ОК (arcsin } 0,5 = 0,523599)$$

Для вычисления экспоненциальной функции e^x можно использовать следующее выражение:

$$e^x = 1 + \frac{383}{384}x + \frac{1}{2}x^2 + \frac{17}{96}x^3 + \frac{1}{24}x^4.$$

В строке 13 это выполняет слово

$$\text{EXP } 10000 \cdot x \text{ --- } 10000 \cdot e^x$$

Пример.

$$5000 \text{ EXP } 16484 \text{ (exp } 0,5 = 1,648721)$$

Таким образом, выше представлены примеры вычисления элементарных функций на языке Форт с применением аппроксимаций и разложений в ряд на базе аппарата только целочисленной арифметики. Наряду с этим для вычисления ряда функций могут использоваться итерационные методы. Например, вычисление функции \sqrt{x} может выполняться на основе итерационной формулы

$$x' = (N/x + x)/2, \quad (7.3)$$

где x' — новое приближение, x — старое приближение, N — число, из которого извлекается корень. Процесс вычисления $x(x \rightarrow \sqrt{N})$ продолжается до тех пор, пока не окажется $x = x'$. Для чисел двойной точности обычно достаточно провести до 7 итерационных приближений. Это делает слово

$$\text{DSQR } d \text{ --- } n \text{ (} n = \sqrt{d}\text{)}$$

дающее целочисленное значение квадратного корня n из числа d двойной точности. Слово DSQR определено в строке 15 листа 7.6.

Пример.

$$144 \text{ DSQR } . 12 \text{ ОК } (\sqrt{144} = 12)$$

Приведенные выше слова обеспечивают вычисление наиболее распространенных элементарных функций с помощью целочисленных версий языка Форт. Их можно использовать как при вычислениях, так и при построении графиков различных функций, полученных с помощью комбинаций элементарных функций.

§ 7.10. Рекурсивные процессы и программирование сверху вниз

Целочисленные версии языка Форт не поддерживают непосредственно аппарат программирования сверху вниз. Этот недостаток отчетливо заметен при попытках реализации рекурсивных процедур, т. е. процедур, внутри которых имеется обращение к самим себе.

Пусть мы решили создать рекурсивную процедуру, которая к некоторому числу n_0 каждый раз прибавляет 1 и выводит результат до тех пор, пока n не достигнет 10. Попытаемся создать словарную статью для такой процедуры в виде слова RECURS:

```
: RECURS 1 + DUP . DUP 10 = IF  
QUIT ELSE RECURS THEN ;
```

Здесь, если $n=10$, запланирован выход из процедуры (слово QUIT), а иначе — обращение к ней самой (слово RECURS после слова ELSE). К сожалению (в отличие от версии FSP88), как только мы попытаемся ввести эту словарную статью, будет получено сообщение об ошибке:

```
RECURS ? MSG # 0
```

(что за RECURS ?, слова нет в словаре)

Итак, ПЭВМ выразила недоумение по поводу наличия слова RECURS внутри словарной статьи. В результате ввод статьи игнорируется. Это объясняется тем, что в целочисленных версиях языка Форт определяемое слово входит в словарь только после проверки всех слов словарной статьи на наличие их в словаре и исполнении слова ; (точка с запятой).

В версии FSP88 в состав словаря сразу включается слово с именем, стоящим вслед за двоеточием. Поэтому на это имя можно ссылаться далее в тексте словарной статьи и тем самым естественным образом реализовать рекурсивные процедуры. Еще большими возможностями в этой части обладает система ДССП-80 [5, 9], в которой в словарную статью на стадии компиляции могут включаться любые слова, определяемые позже.

Однако и для обычных версий языка Форт ситуация с созданием рекурсивных процедур и программированием сверху вниз выглядит не столь мрачно, как это кажется на первый взгляд.

Как отмечалось, в этих версиях имеется слово EXECUTE, вы-

полняющее любое другое слово, заданное его адресом (поля параметров или кодов). Адрес можно задать значением переменной, например ADDR, которое будет уточнено в дальнейшем. Вернемся к нашему примеру. Зададим вначале нулевое значение переменной ADDR (версия fig-FORTH):

```
0 VARIABLE ADDR OK
```

Затем запишем определение слова RECURS:

```
: RECURS 1 + DUP . DUP 10 = IF  
QUIT ELSE ADDR @ EXECUTE THEN ;
```

Теперь задание слова RECURS прошло гладко. Обратите внимание на то, что вместо прямого обращения к слову RECURS после слова ELSE стоит косвенное обращение:

```
ADDR @ EXECUTE
```

Такое обращение позволяет выполнить в принципе любое слово, если оно задано адресом в виде значения переменной ADDR. Ни в коем случае не пытайтесь выполнить слово RECURS до задания переменной ADDR конкретного и реального значения. Если не следовать этому правилу, можно вызвать непредвиденные ситуации — вплоть до полного сброса системы (после этого все придется вводить заново).

Чтобы выполнить слово RECURS, надо задать значение ADDR, например так:

```
' RECURS CFA ADDR ! OK
```

если адрес для слова EXECUTE задается как адрес поля кодов, или

```
' RECURS ADDR ! OK
```

если указанный адрес задается как адрес поля параметров словарной статьи. Уточните это по конкретной реализации Форта, иначе результат будет тот же, что и при произвольном указании адреса.

Теперь можно пустить слово RECURS:

```
.0 RECURS 1 2 3 4 5 6 7 8 9 10 или  
5 RECURS 6 7 8 9 10
```

Останов вычислений словом QUIT происходит без выдачи сообщения OK (но его можно получить, нажав клавишу перевода строки).

Отметим, что адрес как значение переменной ADDR можно задать и другими способами. Например, для этого можно воспользоваться словом FIND (или —FIND в версии fig-FORTH). После этого слова задается слово, адрес которого ищется.

Примечательным свойством слова FIND (или —FIND) является то, что последующее слово может быть как входящим в словарь, так и отсутствующим в нем. В зависимости от этого результат выполнения

слова FIND будет различным. В версии FORTH-79 имеем

FIND Слово — — — 0 (если слова нет в словаре)

FIND Слово — — — PFA (если слово есть в словаре)

Таким образом, значение адреса 0 указывает на то, что слова вслед за FIND в словаре нет. Адрес, не равный нулю, указывает на наличие слова в словаре. Следовательно, в первом случае нельзя применять слово EXECUTE, а во втором случае это возможно.

Несколько сложнее результат выполнения слова —FIND для версии fig-FORTH:

—FIND Слово — — — 0 (слова нет в словаре)

—FIND Слово — — — PFA b1 (слово есть в словаре)

Здесь при наличии слова в словаре —FIND оставляет на вершине стека значение $f=1$ (флаг поднят), байт b длины слова. ($b-128$) и PFA словарной статьи.

Таким образом, слово FIND (или —FIND) также можно использовать для нахождения адреса слова, необходимого для исполнения слова EXECUTE.

Еще одна возможность создания рекурсивных процедур связана с применением слова LATEST, оставляющего адрес поля имени последней вводимой словарной статьи. С помощью слова MYSELF (самого себя)

: MYSELF LATEST PFA CFA , ; IMMEDIATE

адрес NFA преобразуется в PFA, а затем в CFA. Таким образом, обеспечивается обращение в словарной статье к самой себе. Аналогичную возможность в расширениях FORTH-83 обеспечивает слово RECURSE.

В качестве применения слова MYSELF рассмотрим рекурсивную процедуру вычисления факториала:

: FACTORIAL DUP 0 > IF DUP 1—
MYSELF * ELSE DROP 1 THEN ;

Здесь если вводимое перед словом FACTORIAL число $n > 0$, то от него отнимается 1 и оно умножается само на себя, т. е. имеем $n! = n \cdot (n-1) \cdot (n-2) \dots 2 \cdot 1$. Иначе выдается значение 1.

Все сказанное о технике создания рекурсивных процедур относится и к программированию по методу «сверху вниз». Для этого используются слова EXECUTE и MYSELF, позволяющие писать вышестоящие фрагменты программы до того, как определены отдельные их (нижестоящие) части.

Говоря о структурном программировании «сверху вниз» следует отличать технологию программирования от конкретной практической реализации программ. Вряд ли можно возражать против полезности такой технологии, когда вначале мысленно задаются общие крупные блоки программ, а затем проектируются их части.

Однако возьмем более наглядную ситуацию. Пусть нам надо спроектировать автомобиль. Вполне естественно представить (мысленно или в чертежах) его кузов и форму, затем раму, двигатель на раме, колеса и т. д. Однако стоит нам приступить к практической реализации задуманного, как мы будем вынуждены поступать фактически наоборот: вначале из мелких деталей собрать двигатель, поместить его на раму, закрепить на ней кузов, колеса и т. д.

Система ДССП-80, близкая по идеологии к языку Форт, реализует структурное программирование «сверху вниз» не только мысленно, но и на практике [9]. Однако исполнить вышестоящее слово, содержащее еще не заданные нижестоящие слова, все равно нельзя (будет сообщение об ошибке — отсутствие слов в словаре). Обычные версии языка Форт позволяют реализовать технологию программирования «сверху вниз», но не дают возможности осуществить его прямо на практике (хотя косвенно с применением слова MYSELF или EXECUTE это возможно).

§ 7.11. Слова для управления и контроля

Средствами языка Форт легко создавать слова управления и контроля ПЭВМ. Несколько таких слов включено в лист 7.11.

Лист 7.11

Слова для управления и контроля ПЭВМ

```

5 LIST
SCR # 5
  0 ( CONTROL WORDS )
  1 : WAIT ( STO? ) KEY DROP ;
  2 : PAUSE ( N --- 0.01*N SECOND )
  3  0 DO 100 0 DO LOOP LOOP ;
  4 : CDUMP ( A N --- ) BASE @ >R CR 0 DO DUP I +
DUP
5  DECIMAL . DUP C@ DUP 3 .R HEX SWAP DUP
6  6 U.R SWAP 3 .R DECIMAL C@ DUP DUP 31 > SWAP
165 <
7  AND IF 2 SPACES EMIT ELSE
8  DROP THEN CR LOOP R> BASE ! CR ;
9  : SFORTH ." LCODE=" SIZE . FORTH DEFINITIONS D
ECIMAL
10 LATEST 12 +ORIGIN ! HERE 28 +ORIGIN !
11  HERE 30 +ORIGIN ! HERE FENCE !
12  ' FORTH 8 + 32 +ORIGIN MON ;
13  -->
14
15
ок

```

Два первых слова были уже описаны ранее. Поэтому ограничимся упоминанием их функционального назначения.

Слово

WAIT _ _ _ _

обеспечивает останов ПЭВМ до нажатия любой клавиши. Последнее ведет к продолжению работы.

Слово

PAUSE n _ _ _ (пауза $0,01n$ с)

задает паузу в вычислениях, примерно равную $0,01n$ с, используя конечное время исполнения цикла DO...LOOP.

Реализация слова CDUMP существенно отличается от реализации слова DUMP в гл. 6. Слово CDUMP задается в виде

CDUMP A n _ _ _

оно распечатывает содержимое n байтов с начального адреса A в следующем виде:

Адрес	Код	Адрес	Код	Символ
десятичный	десятичный	16-ричный	16-ричный	по ASCII

Распечатка символа по ASCII десятичная, если код символа c лежит в пределах

$$31 < c < 165$$

При этом выводятся символы всех знаков, включая графемы пользователя. Пример применения слова CDUMP дан на листинге 7.12.

Лист 7.12

Иллюстрация применения слова CDUMP

```
28000 10 CDUMF
28000 183 6D60 B7
28001 117 6D61 75 u
28002 44 6D62 2C ,
28003 96 6D63 60 `
28004 210 6D64 D2
28005 117 6D65 75 u
28006 71 6D66 47 G
28007 104 6D67 68 h
28008 14 6D68 E
28009 102 6D69 66 f
```

ok

Слово CDUMP реализовано таким образом, что оно не меняет значения переменной BASE (это значение запоминается вначале в стеке возврата, а затем восстанавливается после выполнения операций словарной статьи слова CDUMP).

Еще одно полезное слово SFORTH необходимо для записи измененной форт-системы на носитель информации (см. § 6.8). Это слово вначале с помощью системной переменной SIZE выдает длину области ОЗУ, занимаемой всей форт-системой:

LCODE=Число

Затем слово SFORTH перемещает указатели словаря FORTH на новое место, что делает новые слова (заданные пользователем) входящими в подсловарь FORTH и защищенными от стирания любого вида

(COLD или FORGET). Таким образом, эти слова становятся базовыми.

Достаточно исполнить слово SFORTH, чтобы получить все данные для записи изменений форт-системы на носители информации. Ниже дан пример применения слова SFORTH :

```
SFORTH LCODE = 13548
```

Происходит возврат в Бейсик:

```
SAVE "FORTH" LINE 1 (Запись управляющей Бейсик-программы)
```

```
SAVE "RALE" CODE USR "A", 168 (Запись графем букв русского алфавита)
```

```
SAVE "FORTH" CODE 24128, 13558 (Запись кодов FORTH)
```

В последнем случае предполагается, что коды форт-системы располагаются в ячейках ОЗУ с начальным адресом 24128. Обычно рекомендуется записывать коды в ячейки, число которых на 5—10 больше указываемого переменной SIZE (и значением LCODE), так что цифра 13558 есть значение LCODE + 10.

§ 7.12. Форт в качестве словаря и базы данных

Организация словаря в целочисленных версиях языка Форт совершенно аналогична принятой для словарей обычных языков. Это резко облегчает создание на базе Форта словарей для перевода с одного языка на другой. Здесь важно отметить и такую полезную особенность современных версий языка Форт, как задание графем. Для некоторых языков, например японского или китайского, это совершенно необходимо, так как их знаки (иероглифы) чаще всего отсутствуют в алфавите ПЭВМ. С этой проблемой приходится сталкиваться и при работе с русским языком на ПЭВМ с только латинскими буквами.

Словарь на языке Форт можно составить различным образом. Простейший — придание словам языка, с которого идет перевод, функций слов Форта. Тогда программа-словарь есть просто совокупность словарных статей следующего вида:

```
: Слово ." Перевод слова " ;
```

Приведенный ниже лист содержит 16 английских слов и статьи перевода их на русский язык.

Лист 7.13

Пример задания словаря-переводчика

```
1 LIST
000 CR # 1
001 : ABODE ." ЖИЛИЩЕ" ;
002 : ABOUT ." ВОКРУГ" ;
003 : BLACK ." ЧЕРНЫЙ" ;
004 : CAT ." ТАКСИ" ;
005 : DEBATE ." ДИСКУССИЯ" ;
006 : DOUBLE ." ДВОЙНИК" ;
007 : I ." Я" ;
008 : AM ." ЕСТЬ" ;
009 : EFFORT ." УСИЛИЕ" ;
010 : MAN ." ЧЕЛОВЕК" ;
```

```

10 : FIRST . " ПЕРВЫЙ" ;
11 : FOOL . " ДУРАК" ;
12 : GLORY . " СЛАВА" ;
13 : INK . " ЧЕРНИЛА" ;
14 : JACKET . " КУРТКА" ;
15 : PAPER . " СТРАНИЦА" ;

```

А вот так выглядит диалог с ПЭВМ:

Лист 7.14

Пример перевода с английского языка на русский

```

I Яок
AM ЕСТЬок
FIRST ПЕРВЫЙок
MAN ЧЕЛОВЕКОк

```

В данном случае на заданную фразу

I AM FIRST MAN

последовал довольно точный и осмысленный перевод

Я ЕСТЬ ПЕРВЫЙ ЧЕЛОВЕК

С помощью этой шуточной программы можно получить и иные, менее лестные оценки (см., к примеру, слово в строке 11 листа 7.13).

Несмотря на крайнюю примитивность описанного подхода, он вполне практичен. Для многих задач (составление телефонной книги, каталога изданий, списка личных научных работ, перечня организаций и т. д.) можно ограничиться им.

Однако (см. § 6.9) Форт позволяет легко строить и более изощренные банки данных. К примеру, можно организовать подсловари с именами латинских букв A, B, ..., Z и задавать слова, начинающиеся с них, в составе этих подсловарей. В результате мы получим организацию англо-русского словаря, принятую для такого рода книг. При этом с помощью команды VLIST можно выводить как приоритетный (т. е. первый) список всех слов на данную букву. Это облегчает поиск нового слова. Для простых ПЭВМ с введенной в их ОЗУ форт-системой программирования можно задать не более 500—1000 слов. В этом случае любое слово разыскивается практически мгновенно. Для более сложных ПЭВМ (с емкостью ОЗУ 128—256К байт и выше) число слов может достигать многих тысяч и организации словаря нужно уделять особое внимание. В частности, в этом случае введение подсловарей становится обязательным.

Приведенный пример, разумеется, не исчерпывает всех возможностей Форты по работе с текстовой информацией. На Форте могут строиться эффективные текстовые редакторы, однако вопросы их создания далеко выходят за рамки данной книги и представляют специальный и самостоятельный интерес. Большие возможности представляет использование для форт-банков данных дисковых накопителей, снимающих ограничение на емкость ОЗУ. Есть все основания полагать, что базы данных на основе форт-системы программирования получат в ближайшее время широкое распространение, так как Форт — язык наиболее приспособленный для реализации их функций (каталогизации и поиска информации).

ВЕРСИИ FORTH ДЛЯ ПЕРСОНАЛЬНЫХ ЭВМ КЛАССА IBM PC

§ 8.1. Операции с редактором и словарем для версии MVP-FORTH

Персональные ЭВМ класса IBM PC являются наиболее распространенными ПЭВМ и имеют обширное программное обеспечение. В него входит ряд форт-систем программирования: MVP-FORTH, PC/FORTH, graf-FORTH и др. Система MVP-FORTH является расширением стандарта FORTH-79 и коммерческой версии fig-FORTH. Ниже дается краткое описание этой версии. Для полного представления следует ознакомиться с версией fig-FORTH (гл. 6). Подробное описание MVP-FORTH имеется также в [24].

Версия MVP-FORTH реализует в основном классические для языка Форт возможности. Она удобна для подготовки системных и прикладных программ с целочисленными арифметическими операциями. Графических средств MVP-FORTH не содержит, что является недостатком этой версии (однако такие средства можно ввести в виде подпрограмм на ассемблере).

После загрузки файла forth из MSDOS можно задавать новые слова и переименовывать имеющиеся, например задав слово CLS для очистки экрана:

```
A>FORTH
MVP-FORTH   VERSION 1.0405.03
: CLS <IBM-PAGE> ; OK
CLS
```

Как и в версии fig-FORTH, ввод редактора выполняется вводом слова

```
EDITOR
```

очистка листа с выбранным номером *s* — словами

```
s CLEAR
```

а вызов листинга листа — словами

```
s LIST
```

Ниже представлен вывод очищенного листа и в конце его запись во вторую строку словарной статьи слова DEMO:

```
OK
EDITOR OK
1 CLEAR OK
1 LIST
SCR #1
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
OK
2 PP : DEMO ." HELLO ! " ; OK
```

Следует обратить внимание, что вместо знака P для ввода строки используется знак PP после ее номера.

Если теперь вызвать листинг листа, слово DEMO будет в строке 2:

```
OK
1 LIST
SCR #1
0
1
2 : DEMO ." HELLO ! " ;
3
4
5
6
7
8
9
10
11
12
13
14
15
OK
1 LOAD OK
DEMO HELLO ! OK
```

Проведя аналогичные операции с листом № 2, занесем в него листинг слова .ASCII, дающего коды ASCII и соответствующие им символы:

```
0 PP ( CODES ASCII -> CHARS ) OK
2 PP : .ASCII OK
3 PP      CR CR OK
4 PP      DO OK
5 PP      I DUP 5 .R ( PRINT CODES ASCII ) OK
6 PP      2 SPACES EMIT ( PRINT CHARS ) OK
7 PP      LOOP OK
8 PP      CR OK
9 PP ; OK
```

После вызова листинг имеет вид:

```
2 LIST
SCR #2
 0 ( CODES ASCII -> CHARS )
 1
 2 : .ASCII
 3      CR CR
 4      DO
 5      I DUP 5 .R ( PRINT CODES ASCII )
 6      2 SPACES EMIT ( PRINT CHARS )
 7      LOOP
 8      CR
 9 ;
10
11
12
13
14
15
```

Теперь команда FLUSH фиксирует лист, а LOAD ведет к его компиляции:

```
FLUSH OK
2 LOAD OK
```

Исполнив команду 32 91 .ASCII, получим левая часть

32		33	!	34	"	35	#	36	\$
42	*	43	+	44	,	45	-	46	.
52	4	53	5	54	6	55	7	56	8
62	>	63	?	64	@	65	A	66	B
72	H	73	I	74	J	75	K	76	L
82	R	83	S	84	T	85	U	86	V

правая часть

37	%	38	&	39	'	40	(41)
47	/	48	0	49	1	50	2	51	3
57	9	58	:	59	;	60	<	61	=
67	C	68	D	69	E	70	F	71	G
77	M	78	N	79	O	80	P	81	Q
87	W	88	X	89	Y	90	Z	91	[

Исполнение слова VLIST выводит на индикацию все слова версии MVP-FORTH.


```

;CODE CODE ENTERCODE ASSEMBLER ?ERROR4 ?ERROR3
?ERROR2 ?ERROR1 --> EDITOR WIPE LINE MATCH
<MATCH> ['] U.R OCTAL H FLUSH ERASE EMPTY
>TYPE >BINARY 2VARIABLE 2SWAP 2OVER 2DUP 2DROP
2CONSTANT 2@ 2! -TEXT 'S SAVE-FORTH .S .SR
.SL .SS VLIST INDEX ID. DVARIABLE DUMPL DUMP
DUMP-HEADER PAUSE .INDEX DMIN DMAX DCONSTANT D@
D> D= D0= D- DU< DSWAP DOVER COPY BMOVE \
TRIAD 'TITLE TITLE THRU <IBM-PAGE> ] [COMPILE]
[ XOR WORD WIDTH WHILE WHERE WARNING
VOCABULARY VOC-LINK VARIABLE USER USE UPDATE UP
UNTIL U< U/MOD U. U* TYPE TRAVERSE TRACK
TOGGLE TIB THEN TEXT T&SCALC SYSCALL SWAP
STATE SPT SPDRV SPBLK SPACES SPACE SP@ SPO
SP! SMUDGE SIGN SET-DRX SEC/TR SEC/BLK SEC-WRITE
SEC-READ SEC SCR SAVE-BUFFERS S0 S->D RPP RP@
RP! ROT ROLL REPEAT R@ R> R0 R/W R# QUIT
QUERY PW@ PW! PREV PP PICK PFA PAGE PAD P@
P! OVER OUT OR OFFSET NUMBER NOT NFA NEXT1
NEXT NEGATE MOVE MOD MIN MAX-DRV MAX M/MOD
M/ M+ M*/ M* LOOP LOAD LITERAL LIT LIST
LIMIT LFA LEAVE LATEST KEY J INTERPRET INTCALL
INIT-USER INIT-FORTH IMMEDIATE IF I' I HOLD
HLD HEX HERE FREEZE FORTH FORGET FLD FIRST
FIND FILL FENCE EXPECT EXIT EXECUTE EPRINT
ENCLOSE EMPTY-BUFFERS EMIT ELSE DUP DROP DRIVE
DR4 DR3 DR2 DR1 DR0 DR-DEN DPUSH DPL DP
DOES> DO DNEGATE DLITERAL DISK-ERROR DIGIT DEPTH
DENSITY DEN DEFINITIONS DECIMAL DDUP DDROP DABS
D< D.R D. D+- D+ D! CURRENT CSP CREATE CR
COUNT CONVERT CONTEXT CONSTANT CONFIGURE. COMPILE
COLD CMOVE CLEAR CHANGE CFA C@L C@ C/L C,
C!L C! BYE BUFFER BRANCH BPDRV BLOCK BLK/DRV
BLK BLANK BL BEGIN BASE APUSH AND ALLOT
AGAIN ABS ABORT" ABORT @L @ ?TERMINAL ?STREAM
?STACK ?PAIRS ?LOADING ?DUP ?CSP ?CONFIGURE
?COMP ? >R >IN > = <WORD> <VOCABULARYFIG>
<VOCABULARY79> <T&SCALC> <R/W> <PAGE> <NUMBER>
<LOOP> <LOAD> <LINE> <KEY> <INTERPRET> <FIND>
<FILL> <EXPECT> <EMIT> <DO> <CR> <CMOVE> <CMOVE>
<BLOCK> <ABORT> <ABORT"> <?TERMINAL> <<CMOVE>
<;CODE> </LOOP> <."> <-FIND> <+LOOP> <#> < ;
: 79-STANDARD 2/ 2- 2+ 2* 2 1- 1+ 1
OBRANCH 0> 0= 0< 0 /MOD /LOOP / .R .LINE
." . -TRAILING -FIND - , +LOOP +BUF +- +!
+ */MOD */ * ( 'WORD 'VOCABULARY 'T&SCALC
'STREAM 'R/W 'PAGE 'NUMBER 'LOAD 'KEY 'INTERPRET
'EXPECT 'EMIT 'CR 'BLOCK 'ABORT '?TERMINAL
'-FIND ' #S #BUFF #> # !L ! OK

```

С помощью слов VOCABULARY и DEFINITION можно создать подсловари и ввести слова внутрь их. Ниже показано создание трех подсловарей: DEMO1 со словами A1, A2 и A3; DEMO2 со словами B1 и B2 и DEMO3 со словом C1:

```

OK
VOCABULARY DEMO1 OK
VOCABULARY DEMO2 OK
VOCABULARY DEMO3 OK
DEMO1 DEFINITIONS OK
: A1 ." WORD A1" ; OK
: A2 ." WORD A2" ; OK
: A3 ." WORD A3" ; OK
DEMO2 DEFINITIONS OK
: B1 ." WORD B1" ; OK
: B2 ." WORD B2" ; OK
DEMO3 DEFINITIONS OK
: C1 ." WORD C1" ; OK
FORTH VLIST
DEMO3 DEMO2 DEMO1 ;CODE CODE ENTERCODE ASSEMBLER
?ERROR4 ?ERROR3 ?ERROR2 ?ERROR1 --> EDITOR WIPE
LINE MATCH <MATCH> ['] U.R OCTAL H FLUSH
ERASE EMPTY >TYPE >BINARY 2VARIABLE 2SWAP 2OVER
2DUP 2DROP 2CONSTANT 2@ 2! -TEXT 'S SAVE-FORTH
.S .SR .SL .SS VLIST INDEX ID. DVARIABLE
DUMPL DUMP DUMP-HEADER PAUSE .INDEX DMIN DMAX
DCONSTANT D@ D> D= D@= D- DU< DSWAP DOVER
COPY BMOVE \ TRIAD 'TITLE TITLE THRU
<IBM-PAGE>

```

Следующий пример иллюстрирует использование этих слов:

```

A1
^^
NOT RECOGNIZED
DEMO1 VLIST
A3 A2 A1 DEMO3 DEMO2 DEMO1 ;CODE CODE
ENTERCODE ASSEMBLER ?ERROR4 ?ERROR3 ?ERROR2
?ERROR1 --> EDITOR WIPE LINE MATCH <MATCH> [']
U.R OCTAL H FLUSH ERASE EMPTY >TYPE >BINARY
2VARIABLE 2SWAP 2OVER 2DUP 2DROP 2CONSTANT OK
A1 WORD A1OK
A3 WORD A3OK
B1
^^
NOT RECOGNIZED
DEMO2 OK
B1 WORD B1OK
VLIST
B2 B1 DEMO3 DEMO2 DEMO1 ;CODE CODE
ENTERCODE ASSEMBLER ?ERROR4 ?ERROR3 ?ERROR2
?ERROR1 --> EDITOR WIPE LINE MATCH <MATCH> [']
U.R OCTAL H FLUSH ERASE EMPTY >TYPE >BINARY
2VARIABLE 2SWAP 2OVER 2DUP 2DROP 2CONSTANT 2@
2! -TEXT 'S SAVE-FORTH .S .SR .SL .SS VLIST
INDEX ID. OK

```

Вначале система MVP-FORTH с текущим словарем FORTH отказалась воспринять слово A1, так как оно не входит в этот словарь (поступает сообщение NOT RECOGNIZED). Указание подсловаря

DEMO1 делает его текущим по поиску. После этого команда VLIST дает распечатку слов, начиная со слов A3, A2 и A1 текущего подсловаря. Слова A1 и A3 нормально исполняются. Однако слово B1 другого подсловаря (DEMO2) не исполняется. После указания имени DEMO2 этого подсловаря слово B1 исполняется, а слово VLIST дает распечатку слов, начиная с B2 и B1. Вывод на печать прерван для того, чтобы повторно не выводить весь большой список слов.

§ 8.2. Основные операции и типы данных версии MVP-FORTH

Поскольку арифметические и логические операции версии MVP-FORTH практически аналогичны описанным для версии FORTH-79, ограничимся лишь несколькими типовыми примерами (см. ниже).

Операции сложения и вычитания:

```
OK
123 456 + . 579 OK
123 1+ . 124 OK
123 2+ . 125 OK
123456. 654321. D+ D. 777777 OK
456 321 - . 135 OK
123 1- . 122 OK
123 2- . 121 OK
654321. 12345. D- D. 641976 OK
```

Операции умножения и деления:

```
OK
17 4 * . 68 OK
40000 50 U* D. 2000000 OK
123 2* . 246 OK
30000 2000 M* D. 60000000 OK
17 4 / . 4 OK
17 4 MOD . 1 OK
17 4 /MOD . . 4 1 OK
1200003. 600 M/MOD D. . 2000 3 OK
124 2/ . 62 OK
15 4 2 */ . 30 OK
17 2 3 */MOD . . 11 1 OK
```

Функции и специальные операции:

```
OK
123 NEGATE . -123 OK
123456. DNEGATE D. -123456 OK
123 -2 +- . -123 OK
123 2 +- . 123 OK
123456. -5 D+- D. -123456 OK
123456. 5 D+- D. 123456 OK
2000 S->D D. 2000 OK
-123 ABS . 123 OK
-123456. DABS D. 123456 OK
123 456 MAX . 456 OK
123 456 MIN . 123 OK
```

Из особых операций нужно отметить операции, реализуемые словами \dagger — (для операндов обычной разрядности) и $D\dagger$ — (для операндов двойной разрядности). Слова \dagger — и $D\dagger$ — дают первое число $n1$ со знаком второго числа, т. е.

```
 $\dagger$  —  $n1\ n2$  — — —  $n1 \cdot \text{sign } n2$   
 $D\dagger$  —  $n1\ n2$  — — —  $d1 \cdot \text{sign } d1$ 
```

Логические операции версии MVP-FORTH порождают предикаты (состояния флагов) в виде чисел 0 и 1:

```
ОК  
2 3 = . 0 ОК  
4 4 = . 1 ОК  
8 6 > . 1 ОК  
5 0 > . 1 ОК  
9 8 = NOT . 1 ОК  
8 8 = NOT . 0 ОК  
1 1 AND . 1 ОК  
1 0 AND . 0 ОК  
0 1 AND . 0 ОК  
0 0 AND . 0 ОК  
0 0 OR . 0 ОК  
1 0 OR . 1 ОК  
0 1 OR . 1 ОК  
1 0 XOR . 1 ОК  
0 0 XOR . 0 ОК  
1 1 XOR . 0 ОК  
0 1 XOR . 1 ОК
```

Их применение подобно описанному для версии FORTH-79.

Версия MVP-FORTH имеет возможности для задания оснований чисел 10 (DECIMAL), 8 (OCTAL) и 16 (HEX). Их использование для преобразования чисел по основанию поясняет следующий пример:

```
ОК  
BASE ? 10 ОК  
16 BASE ! ОК  
A 2+ . C ОК  
ABC 3 + . ABF ОК  
BASE @ DECIMAL . 16 ОК  
HEX ABC DECIMAL . 2748 ОК  
DECIMAL 2748 HEX . ABC ОК  
DECIMAL 100 OCTAL . 144 ОК  
OCTAL 144 DECIMAL . 100 ОК  
HEX A A * DECIMAL . 100 ОК  
HEX A A * . 64 ОК
```

Другой пример:

```
ОК  
: BINARY 2 BASE ! ; ОК  
DECIMAL 100 BINARY . 1100100 ОК  
BINARY 1100100 DECIMAL . 100 ОК
```

иллюстрирует задание слова `BINARY`; устанавливающего основание 2, и его применение для перевода десятичных чисел в двоичные и наоборот.

Для вывода одного числа из стека используются слова `.`, `D.` и `U.`

```
ОК
123 . 123 ОК
40000 . -25536 ОК
40000. D. 40000 ОК
123 D.
  ^ ^
EMPTY STACK
40000 U. 40000 ОК
123 U. 123 ОК
```

Контроль на переполнение не проводится (см. пример неверного вывода числа 40000 словом `.`). Нельзя выводить числа одинарной разрядности словом `D.` (см. пример вывода числа 123 словом `D.`). Версия `MVP-FORTH` имеет слово `.S` для индикации всех чисел в стеке без их уничтожения:

```
ОК
12 34 56 .S
56 34 12
ОК
.S
56 34 12
ОК
ABORT
.S
EMPTY STACK
ОК
```

Для уничтожения чисел в стеке используется слово `ABORT`. Действие слов `DUP` и `?DUP` иллюстрирует следующий пример:

```
ОК
123 DUP .S
123 123
ОК
ABORT
0 DUP .S
0 0
ОК
ABORT
123 ?DUP .S
123 123
ОК
ABORT
0 ?DUP .S
0
ОК
```

Еще ряд примеров показывает действие слов OVER, PICK, DROP, ROT, ROLL и DEPTH:

```
OK
12 34 56 OVER .S
34 56 34 12
OK
4 PICK .S
12 34 56 34 12
OK
DROP .S
34 56 34 12
OK
ROT .S
34 34 56 12
OK
4 ROLL .S
12 34 34 56
OK
DEPTH . 4 OK
```

Из этих примеров можно заключить, что действие этих слов вполне обычное и соответствует ранее рассмотренным версиям языка Форт.

Слово 3DTIMES демонстрирует использование слов >R, R@ и R> для обмена числами между арифметическим стеком и стеком возврата:

```
OK
: 3DTIMES DUP >R * ROT R@ * ROT R> * ROT ; OK
12 34 56 2 3DTIMES .S
112 68 24
OK
```

Это слово умножает три первых числа на четвертое число. Для его временного хранения используется стек возврата.

К основным типам данных в MVP-FORTH относятся константы и переменные. Они задаются, как и в версии FORTH-79, что поясняют следующие примеры.

Задание и применение целочисленных констант PI (одинарной разрядности) и 2PI (двойной разрядности):

```
OK
31416 CONSTANT PI OK
3141593. 2CONSTANT DPI OK
PI . 31416 OK
DPI D. 3141593 OK
PI 2/ . 15708 OK
```

Задание и применение целочисленной переменной ALPFA одинарной разрядности:

```
OK
VARIABLE ALPFA OK
123 ALPFA ! OK
ALPFA ? 123 OK
ALPFA @ . 123 OK
ALPFA @ 2* . 246 OK
25 ALPFA ! OK
ALPFA ? 25 OK
```

Задание и применение целочисленной переменной BETA двойной разрядности:

```
OK
2VARIABLE BETA OK
100000. BETA 2! OK
BETA 2@ D. 100000 OK
BETA 2@ 123. D+ D. 100123 OK
```

Задание и применение USER-переменных:

```
OK
0 USER A OK
1 USER B OK
A U. 22438 OK
B U. 22439 OK
123 A ! OK
A ? 123 OK
456 B ! OK
B ? 456 OK
A ? -14213 OK
```

Переменная B задана с неверным сдвигом (см. пример далее).

Принцип задания одномерного массива с расширением словом ALLOT:

```
OK
VARIABLE VECTOR OK
6 ALLOT OK
100 VECTOR ! OK
101 VECTOR 2 + ! OK
102 VECTOR 4 + ! OK
103 VECTOR 6 + ! OK
VECTOR ? 100 OK
VECTOR 2 + ? 101 OK
VECTOR 4 + ? 102 OK
VECTOR 6 + ? 103 OK
```

Принцип задания блока данных DATA для чисел одинарной разрядности:

```
OK
VARIABLE DATA OK
0 DATA ! OK
1 , 2 , 3 , OK
DATA ? 0 OK
DATA 2 + ? 1 OK
DATA 4 + ? 2 OK
DATA 6 + ? 3 OK
```

Принцип задания блока данных — байт CDATA:

```
OK
VARIABLE CDATA OK
0 CDATA C! OK
1 C, 2 C, 3 C, OK
DATA C@ . 0 OK
DATA 1 + C@ . 0 OK
DATA 2 + C@ . 1 OK
DATA 4 + C@ . 2 OK
DATA 6 + C@ . 3 OK
```

Организация одномерного массива с доступом к любому его элементу:

```
OK
: ARRAY CREATE DUP + ALLOT DOES> SWAP DUP + + ; OK
3 ARRAY DEMO OK
10 0 DEMO ! OK
20 1 DEMO ! OK
30 2 DEMO ! OK
2 DEMO ? 30 OK
1 DEMO @ . 20 OK
0 DEMO ? 10 OK
4 ARRAY VEC OK
123 2 VEC ! OK
456 3 VEC ! OK
2 VEC ? 123 OK
```

Интерактивная форма организации одномерного массива с произвольным доступом:

```
OK
: IVECTOR DUP + VECTOR + ! ; OK
: @VECTOR DUP + VECTOR + @ ; OK
: ?VECTOR @VECTOR . ; OK
50 0 IVECTOR OK
51 1 IVECTOR OK
52 2 IVECTOR OK
53 3 IVECTOR OK
0 @VECTOR . 50 OK
1 @VECTOR . 51 OK
2 ?VECTOR 52 OK
3 ?VECTOR 53 OK
```


Определение адреса компиляции константы PI и замена ее значения с 31416 на значение 314:

```
OK
' PI U. 19228 OK
314 ' PI ! OK
PI . 314 OK
```

Прямое обращение к памяти:

```
OK
30000 50000 ! OK
50000 ? 30000 OK
50000 @ . 30000 OK
65 52000 C! OK
52000 C@ . 65 OK
```

Стирание USER-переменной B и ее повторное задание с правильным сдвигом (на 2, а не на 1):

```
OK
FORGET B OK
2 USER B OK
A U. 22438 OK
B U. 22440 OK
123 A ! 456 B ! OK
A ? 123 OK
B ? 456 OK
A @ B @ + . 579 OK
```

Просмотр памяти (дампинг с помощью слова DUMP):

левая часть

```
OK
1000 50 DUMP
```

ADDRESS	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7
3E8	F	8	18	15	18	89	2D	54	52	41	49	4C	49	4E	C7	DB
3F8	3	C0	5	3E	17	EE	4	DB	7	75	15	53	3	4F	5	29
408	11	FA	F	D2	3	27	5	8	0	B3	1A	6B	10	4	0	4F
418	5	1C	A	E6	FF	15	18	81	AE	ED	3	C0	5	2D	1F	DB

OK

правая часть

```
89ABCDEF01234567
```

```
.....-TRAILIN..
...>.....u.S.O.)
.....'.....k...O
.....-...
```

Помимо рассмотренных средств вывода чисел из стека MVP-FORTH имеет и другие средства вывода. Они иллюстрируются примерами, приведенными ниже.

Символьное преобразование чисел с помощью слов <#, # и #> и их распечатка словом TYPE:

```
OK
: C. <# # # # #> TYPE ; OK
1. C. 001OK
12. C. 012OK
123. C. 123OK
1234. C. 234OK
12345. C. 345OK
```

Задание времени:

```
OK
: HMS <# # # 58 HOLD # # 58 HOLD # # #> CR TYPE ; OK
120715. HMS
12:07:15OK
013456. HMS CR
01:34:56
OK
```

Задание телефонных номеров:

```
OK
: TLF <# # # 45 HOLD # # 45 HOLD #S #> TYPE SPACE ; OK
1234567. TLF 123-45-67 OK
123456. TLF 12-34-56 OK
12345. TLF 1-23-45 OK
```

Преобразование символа в его код по ASCII:

```
OK
: ASCII 32 WORD HERE 1+ C@ ; OK
ASCII A . 65 OK
ASCII B . 66 OK
ASCII Z . 90 OK
ASCII ABCD . 65 OK
```

Ввод в текстовый буфер и преобразование чисел-строк в обычные числа:

```
OK
PAD 10 EXPECT 1234 OK
PAD 1- <NUMBER> D. 1234 OK
PAD 10 EXPECT 1234567 OK
PAD 1- <NUMBER> D. 1234567 OK
```

Контроль длины строк входного потока:

```
OK
1 DUP >IN ? 11 OK
12 DUP >IN ? 12 OK
123 DUP >IN ? 13 OK
1234 DUP >IN ? 14 OK
12345 DUP >IN ? 15 OK
12345 DUP DUP >IN ? 19 OK
```

```

OK
PAD 10 EXPECT 12345      OK
PAD 10 -TRAILING . U. 5 19437 OK
PAD 10 -TRAILING TYPE 12345OK

```

Средства вывода:

```

OK
KEY OK
. 65 OK
65 EMIT AOK
BL . 32 OK
65 EMIT SPACE A OK
65 EMIT 5 SPACES A      OK
50000 5 EXPECT ABCDEOK
50000 5 TYPE ABCDEOK
50000 COUNT . U. 65 50001 OK
PAD U. 19273 OK
PAD 5 EXPECT QWERTOK
PAD 5 TYPE SPACE QWERT OK
123 5 .R 123OK
123 10 .R 123OK
123456. 5 D.R 123456OK
123456. 10 D.R 123456OK

```

Интерактивный ввод целого числа по запросу в текстовый буфер и последующий вывод его в стек с преобразованием из символьной формы в обычную:

```

OK
: INPUT ." ENTER N " QUERY BL WORD <NUMBER> DROP ; OK
INPUT ENTER N 123 OK
. 123 OK

```

Другой вариант интерактивного ввода с присвоением переменной X значения введенного числа (аналог команды INPUT "Комментарий" ; X на языке Бейсик):

```

OK
VARIABLE X OK
: INP ." X=" PAD 6 EXPECT 0. PAD 1- CONVERT 2DROP X ! ; OK
INP X=12 OK
X ? 12 OK
INP X=123 OK
X ? 123 OK
X @ 2 * . 246 OK

```

§ 8.3. Управляющие структуры и системные переменные версии MVP-FORTH

К основным управляющим структурам версии MVP-FORTH относятся условные выражения (реализуемые словами IF, ELSE и THEN) и циклы. Построение управляющих структур иллюстрируется приведенными ниже типовыми примерами.

Проверка числа на равенство 10 с применением полной конструкции IF ... ELSE ... THEN:

```
OK
: X? 10 < IF CR ." X<10 " ELSE CR ." X>10 " THEN ; OK
5 X?
X<10 OK
15 X?
X>10 OK
```

Циклы DO ... LOOP и DO ... n+LOOP :

```
OK
: DEMO1 6 1 DO I . LOOP ; OK
DEMO1 1 2 3 4 5 OK
: DEMO2 11 0 DO I . 2 +LOOP ; OK
DEMO2 0 2 4 6 8 10 OK
: DEMO3 0 10 DO I . -2 +LOOP ; OK
DEMO3 10 8 6 4 2 0 OK
```

Цикл в цикле:

```
OK
: DO1 6 1 DO I . SPACE LOOP CR ; OK
: DEMO4 CR 3 0 DO ." I=" I . CR DO1 LOOP ; OK
DEMO4
I=0
1 2 3 4 5
I=1
1 2 3 4 5
I=2
1 2 3 4 5
OK
```

Статус управляющих переменных I и J в двойном цикле:

```
OK
: DEMO5 DO 4 1 DO ." I=" I . ." J=" J . LOOP CR LOOP ; OK
3 0 CR DEMO5
I=1 J=0 I=2 J=0 I=3 J=0
I=1 J=1 I=2 J=1 I=3 J=1
I=1 J=2 I=2 J=2 I=3 J=2
OK
```

Прерывание цикла с помощью выражения IF LEAVE ELSE THEN:

```
OK
: DL 50 > IF LEAVE ELSE THEN ; OK
: DLEAVE 5 1 DO CR 1 . SPACE 2* DUP DUP . DL LOOP ; OK
1 DLEAVE
1 2
2 4
3 8
4 16 OK
20 DLEAVE
1 40
2 80 OK
```

Циклы вида BEGIN ... f UNTIL и BEGIN f WHILE...REPEAT:

```
OK
: DEMO6 CR 1 BEGIN 2* DUP DUP . 100 > UNTIL DROP ; OK
DEMO6
2 4 8 16 32 64 128 OK
: DEMO7 BEGIN DUP 100 > NOT WHILE 2* DUP . REPEAT DROP ; OK
CR 1 DEMO7
2 4 8 16 32 64 128 OK
```

Прямое исполнение слова + и исполнение с помощью конструкции A EXECUTE, где A — адрес компиляции:

```
OK
12 34 + . 46 OK
' + CFA U. 851 OK
12 34 851 EXECUTE . 46 OK
```

Управление режимом компиляции:

```
OK
: SUM 2 3 + . ; OK
SUM 5 OK
: SUM1 2 3 + . ; IMMEDIATE OK
SUM1 5 OK
: SUM1 SUM ; OK
SUM1 5 OK
: SUM2 SUM1 ; 5 OK
SUM2 OK
: SUM3 [COMPILE] SUM1 ; OK
SUM3 5 OK
: SUM4 [ 2 3 + . ] ; 5 OK
OK
```

Системные переменные версии MVP-FORTH в основном соответствуют применяемым в версиях FORTH-79 и fig-FORTH. Некоторые из системных переменных дают расширенные данные о характеристиках

системы. Например, системная переменная ?CONFIGURE выдает следующую информацию о конфигурации системы:

```
OK
?CONFIGURE

2 DRIVES WITH DENSITIES:  2    2

DENSITY CODE
0 160K DSK
1 320K DSK
2 360K DSK
3
4
5
6 NULL DISK
OK
```

Она указывает на количество дисководов в системе и их тип. Тип задается кодами. Выводится и таблица расшифровки кодов. Так, видно, что примененная ПЭВМ снабжена двумя дисковыми с емкостью каждого 360К байт (код 2).

§ 8.4. Примеры программирования для версии MVP-FORTH

В состав системы MVP-FORTH входит около 40 листов с записанной на них информацией о системе. В частности, даются листы с данными о графических интерфейсах и режимах, системных ошибках, построении ассемблера для микропроцессора 8086, реализации различных дополнительных системных функций, мини-редактора экрана, декомпилятора, задания экранных окон и др. Поскольку эти программы доступны с приобретением MVP-FORTH-системы, не имеет смысла описывать их более подробно.

Поэтому ограничимся несколькими более простыми примерами, подобными рассмотренным в гл. 7.

Ниже представлен пример задания слова CUBE, вычисляющего целочисленную третью степень числа

```
OK
: CUBE DUP DUP * * ; OK
2 CUBE . 8 OK
3 CUBE . 27 OK
FORGET CUBE OK
2 CUBE
  ^^^^
NOT RECOGNIZED
```

Несколько более сложных примеров программирования приведены далее на листах 8.1—8.3.

Лист 8.1

Вычисление факториала при одинарной разрядности $N!$ и двойной разрядности $ND!$

```

OK
3 LIST
SCR #3
0 : UM* >R OVER U* ROT R> * + ;
1 : N! ( N --- N! FOR N<=8 )
2     1 SWAP 1+ 1
3     DO
4     I *
5     LOOP
6 ;
7 : ND! ( N --- D=N! N<=12 )
8     .1 ROT 1+ 1
9     DO
10    I ROT ROT UM*
11    LOOP
12 ;
13
14
15
OK
3 LOAD OK
5 N! . 120 OK
0 ND! D. 1 OK
10 ND! D. 3628800 OK

```

Контрольные примеры приведены под листингом листа.

Лист 8.2

Факторизация (разложение на простые множители) числа

```

OK
3 LIST
SCR #3
0
1 : PRINT ?DUP IF ROT ." *" . DUP 1 > IF ." ^" .
2 ELSE DROP THEN DROP ELSE DROP THEN ;
3 : TOTHE 0 BEGIN >R OVER OVER /MOD R> ROT 0= WHILE
4 1+ >R >R SWAP DROP R> SWAP R> REPEAT PRINT ;
5 : CVECTOR CREATE 0 DO C, LOOP DOES> + C@ ;
6 31 29 23 19 17 13 11 7 8 CVECTOR TRAIL
7 : FACTORS ." =1" 2 TOTHE 3 TOTHE 5 TOTHE DUP 1 =
8 IF CR DROP EXIT THEN DUP
9 0 DO 8 0 DO I TRAIL J + TOTHE LOOP DUP I <
10 IF LEAVE THEN 30 +LOOP DROP ;
11
12
13
14
15
OK
5600 FACTORS =1*2 ^5 *5 ^2 *7 OK

```

Операции над числами с плавающей точкой с интерактивным вводом

```

OK
3 LIST
SCR #3
0 ( FIX-POINT OPERATIONS )
1 : FIX? SWAP OVER DABS <# # # 46 HOLD #S ROT SIGN #> TYPE ;
2 VARIABLE CC 0 CC ! VARIABLE DD 0 DD !
3 VARIABLE AA 0 AA ! VARIABLE BB 0 BB !
4 : D* AA ! BB ! CC ! DD ! DD @ BB @ U* DD @ AA @ U*
5 DROP + CC @ BB @ U* DROP + ;
6 : D/ DROP M/ SWAP DROP S->D ;
7 : FIX+ D+ ; : FIX- D- ;
8 : FIX* D* DUP >R DABS 100 M/MOD R> 0< IF DNEGATE THEN ROT DROP
9   VARIABLE DIV 0 DIV !
10 : FIX/ DROP DUP DIV ! M/ 100 * SWAP DIV @ 100 / / + S->D ;
11 : INTEXT CR ." INPUT NUMBER " QUERY 1 TEXT ;
12 : DINP INTEXT PAD NUMBER ;
13 : SINP INTEXT PAD NUMBER DROP ;
14 : FIXINP DINP ;
15
OK

```

Пример применения слов листа 8.3:

```

OK
DINP
INPUT NUMBER 12345. OK
DINP
INPUT NUMBER 250. OK
D+ D. 12595 OK
FIXINP
INPUT NUMBER 3.14 OK
FIXINP
INPUT NUMBER 2.00 OK
FIX* FIX? 6.28OK

```

В целом можно сделать вывод, что версия MVP-FORTH является типичным вариантом стандартной системы FORTH-79 с присущими последней достоинствами и недостатками. К недостаткам следует отнести отсутствие прямых графических средств, отсутствие прямой поддержки стиля программирования сверху вниз, чрезмерно сложная для ряда применений организации словаря и др. В целом MVP-FORTH — мощная программная система, ориентированная в основном на системное программирование (подготовку мониторов, ассемблеров и дисассемблеров, декомпиляторов, экранных редакторов, баз данных и т. д.).

Детальные сведения о версии MVP-FORTH содержатся в [24] и в других книгах этой серии. Там же приводится и множество прикладных программ, реализованных с помощью версии MVP-FORTH.

§ 8.5. Общая характеристика версии PC/FORTH

Версия PC/FORTH является одной из наиболее мощных целочисленных систем программирования для персональных ЭВМ класса IBM PC. Она является существенно расширенным коммерческим вариантом стандартной версии FORTH-83.

К достоинствам версии PC/FORTH относятся: достаточно развитые средства графики, поддержка стиля программирования сверху вниз, обширные системные возможности, удобный редактор с подсказками, обширная сопровождающая информация, наличие целочисленных реализаций ряда математических функций и др.

Как и версия MVP-FORTH, PC/FORTH ориентирована в первую очередь на разработку системных программ. Однако отмеченные достоинства делают эту версию хорошо приспособленной для разработки высокоэффективных прикладных программ машинной графики и обучающих программ. Полный доступ к ресурсам ПЭВМ обеспечивает эффективное применение версии PC/FORTH для управления разнообразным периферийным оборудованием.

Есть основания полагать, что PC/FORTH становится основной форт-системой программирования одних из наиболее массовых и достаточно серьезных персональных ЭВМ класса IBM PC. Наряду с базовыми моделями IBM PC — XT и AT — к ним относятся многочисленные аналоги таких машин, например отечественные «Искра 1030», ЕС-1841 и др. Поэтому мы подробно разберем эту версию, не считаясь с тем, что отдельные примеры ее применения аналогичны ранее рассмотренным для других версий. Как и для версии MVP-FORTH, рассмотрение ведется на множестве практических примеров, но без подробного описания функций слов (оно дано в гл. 6 при описании версии FORTH-83). Исключение делается лишь в отношении новых возможностей версии PC/FORTH.

§ 8.6. Операции с редактором и словарем PC/FORTH

Версия PC/FORTH и сопровождающая ее документация хранятся в виде файлов на гибком диске. Загрузка производится указанием имени главного файла forth.com. После загрузки на экране появляется следующее сообщение:

```
PC/FORTH 2.00 (Forth-83 Standard)
[PC-DOS version, created 11/15/83]
Copyright 1983 Laboratory Microsystems Inc.
Using file: forth.scr [path support enabled]
Dictionary space available: 22300 bytes
```

ok

Оно напоминает о принадлежности версии PC/FORTH к стандарту FORTH-83, о разработчике версии, о дате разработки (1983 г.), об используемом файле и свободной памяти (22,3К байт). В нижней служебной зоне экрана находится меню подсказок. С его помощью можно перейти к рабочему экрану:

```
PC/FORTH 2.00
Using: forth.scr
01/01/80 00:07:30
Push F1 for help
Caps NumLock
```

Enter command:

push F key to turn on directory

Мы можем также вызвать меню команд:

Command mode functions available:

```
C = Copy single screen
D = Disk directory
E = Edit screen
F = File directory toggle
H = Highlight changes
I = Index to screen file
M = Multiple screen transfer
O = Off highlighting
T = Set tab stops
U = Use new screen file
Esc = Exit to FORTH
```

push any key when ready to proceed

Используя, например, команду D (вызов директории диска), получим на экране распечатку имен файлов диска и их длины в байтах (см. лист на стр. 267).

PC/FORTH 2.00

Enter command:

Using: forth.scr

01/01/80 00:22:37

Push F1 for help

Caps NumLock

Directory for drive C:

forth.scr	52224	scopy.bin	2944	editor.bin	13184	asm86.scr
read.me	164	forth.com	21252	nucleus.com	15360	elective.scr
dosint.scr	87040					

push F key to turn off directory

Загрузив файл forth.scr, мы получаем набор листов с разнообразными словами и информацией. Команда s LIST, как обычно, вызывает печать мнемонического листинга с номером s. Просмотрим несколько листов.

Лист 8.4

Задание текстовых и графических режимов различного разрешения

Screen # 1

```
0 ( Change video modes                05/16/82 )
1 HEX
2 ( color interface, 80 x 25 text mode )
3 : TEXTC          3 MODE B/W ;
4 ( monochrome interface 80 x 25 text mode )
5 : TEXTM          7 MODE B/W ;
6 ( 320 x 200 color graphics )
7 : LRGC           4 MODE 1 PALETTE 3 FOREGROUND ;
8 ( 320 x 200 b/w graphics )
9 : LRGBW          5 MODE 0 PALETTE B/W ;
10 ( 640 x 200 b/w graphics )
11 : HRGBW         6 MODE B/W ;
12 DECIMAL
13
14
15
ok
```

Лист 8.5

Системные ошибки (начало)

Screen # 4

```
0 ( System messages )
1 empty stack
2 dictionary full
3 has incorrect address mode
4 is redefined
5 is undefined
6 disk address out of range
7 stack overflow
8 disk error
9
10
11
12
13 BASE must be DECIMAL
14 missing decimal point
15 PC/FORTH 2.0
ok
```

Пример вывода сообщения об ошибке (стек пуст):-

```
ok
123 123 ok
123 . 123
```

```
Error: "." empty stack
```

```
ok
```

Системные ошибки (конец)

Screen # 5

```

0 ( System messages )
1 compilation only, use in definition
2 execution only
3 conditionals not paired
4 definition not finished
5 in protected dictionary
6 use only when loading
7 off current editing screen
8 declare vocabulary
9
10
11
12 illegal dimension in array definition
13 negative array index
14 array index too large
15
ok

```

Эти листы дают достаточную информацию о графических возможностях PC/FORTH и системных ошибках.

Команда E основного меню переводит систему PC/FORTH в режим редактирования. При этом на экране появляется пустой прямоугольник с номерами строк от 0 до 15 слева и меню подсказок снизу. PC/FORTH предоставляет пользователю полный экранный редактор для ввода и редактирования листингов словарных статей. Ниже представлена запись в редакторе слова .ASCII, ранее реализованного в версии MVP-FORTH :

Screen # 52

Using file: forth.scr Mode = Edit

```

0 ( Print codes ASCII and chars )
1 : .ASCII ( Name words )
2 CR CR
3 DO
4 I DUP 5 .R ( Print codes )
5 2 SPACES EMIT ( Print chars )
6 LOOP
7 CR
8 ;
9
10
11
12
13
14
15

```

ok
16 LOAD

Wait ... loading mini screen editor

TASK is redefined
EDIT is redefined
Compilation of mini screen editor complete.

ok
WORDS

56A6	EDIT	5681	CONTROL	55A7	FXNKEY	5594	CLHC
556C	S.ERASE	554C	!BLK	5539	HOM	551C	+LIN
550C	+.CUR	54F9	+CUR	54DF	!CUR	54BB	.CUR
54B1	F10	54A8	F9	549D	PGDN	5492	PGUP
5482	LEFTARROW	5471	RIGHTARROW	5463	UPARROW	5453	DOWNARROW
5444	HORIZTAB	5436	NEWLINE	5428	HOMEKEY	5419	EXITFLAG
540F	CUR	5404	TASK	5372	CHDIR	523E	RUN
5112	KILL-TASKS	507B	KILL	4FFB	START	4FA2	.TASKS

ok
52 EDIT

В отличие от большинства версий форт-систем в данной версии текст словарной статьи просто заносится в прямоугольник. Нажатие клавиши F10 выводит из редактирования. Дальнейшие операции аналогичны общепринятым: команда FLUSH фиксирует листы в ОЗУ, команда s LIST дает листинг листа s (в обычной форме), команда s LOAD компилирует лист s, команда s CLEAR очищает лист. Так, исполнение команд FLUSH 52 LOAD и 256 52. ASCII вызовет печать кодов и символов ASCII (совершенно аналогичную ранее приведенной для версии MVP-FORTH).

Теперь введем команду 16 LOAD. Получим следующее сообщение (см. лист на стр. 270).

Вначале появляется надпись

```
Wait...loading mini screen editor
```

(Ждите ... загружается экранный мини-редактор)

Затем появляются слова:

```
TASK is redefined (TASK переопределено)
```

```
EDIT is redefined (EDIT переопределено)
```

Это сообщение указывает на возможность переопределения в системе PC/FORTH ранее определенных слов. Завершается ввод мини-редактора указанием о его компиляции.

В версии PC/FORTH нет привычного слова VLIST для печати имен всех слов текущего по поиску словаря. Вместо него используется слово WORDS. Слова выводятся по четыре в строке с указанием в шестнадцатеричной форме их адресов компиляции. Так, после загрузки мини-редактора его словарь становится текущим по поиску и распечатывается при исполнении слова WORDS.

Слова можно задавать и уничтожать как обычно:

```
ok
: CUBE DUP DUP * * ; ok
2 CUBE . 8 ok
3 CUBE . 1B ok
FORGET CUBE ok
2 CUBE
```

```
Error: "CUBE" is undefined
```

```
ok
```

Системное слово

```
s1 s2 INDEX
```

выводит на печать тексты нулевых строк с номерами от s1 до s2. Обычно в них в круглых скобках указываются названия листов и дата их подготовки (см. лист на стр. 272).

ok

1 20 INDEX

- 1 (Change video modes 05/16/82)
- 2 (Reallocate system memory under PCDOS 11/14/83)
- 3 (Reallocate system memory under PCDOS 11/16/83)
- 4 (System messages)
- 5 (System messages)
- 6 (8086 Assembler messages)
- 7 (Integer square root by Newton's method 04/08/82)
- 8 (chop screen file to desired length 01/25/83)
- 9 (Transfer from other system over serial interface HDN)
- 10 (Memory dump, byte format, intrasegment 02/26/82)
- 11 (Memory dump, byte format, intersegment 02/26/82)
- 12 (Memory dump, word format, intrasegment 02/26/82)
- 13 (Memory dump, word format, intersegment 02/26/82)
- 14 (ANSI Cursor Control Functions 10/05/83)
- 15 (Random number generator, by J E Rickenbacker 08/23/82)
- 16 (Mini Screen Editor 11/14/83)
- 17 (Mini Screen Editor, continued 11/14/83)
- 18 (Mini Screen Editor, continued 11/14/83)
- 19 (Stack Utilities by Gerry Mueller 10/17/82)
- 20 (Eratosthenes sieve benchmark program)

ok

QX:

левая часть

ok

QX

0 Change video m	1 Change video m
4 System message	5 System message
8 chop screen fi	9 Transfer from
12 Memory dump, w	13 Memory dump, w
16 Mini Screen Ed	17 Mini Screen Ed
20 Eratosthenes s	21 Interface Age
24 FORTH Decompil	25 FORTH Decompil
28 FORTH Decompil	29 FORTH Decompil
32 Memory allocat	33 window support
36 window example	37 Multitasking d
40	41 ++++++
44 ++++++	45 ++++++
48 ++++++	49 ++++++

ok

правая часть

2 Reallocate sys	3 Reallocate sys
6 8086 Assembler	7 Integer square
10 Memory dump, b	11 Memory dump, b
14 ANSI Cursor Co	15 Random number
18 Mini Screen Ed	19 Stack Utilitie
22 Input number f	23 Console string
26 FORTH Decompil	27 FORTH Decompil
30 FORTH Decompil	31 Memory allocat
34 window support	35 window support
38 Interrupt hand	39 demo for inter
42 ++++++	43 ++++++
46 ++++++	47 ++++++
50 Change video m	

В этом случае из названия выделяются только 14 первых символов, включая пробелы. Остальные символы отсекаются. Круглые скобки в названии опускаются.

Как отмечалось, в версии PC/FORTH возможно переопределение слов, т. е. создание словарных статей с именами, ранее встречавшимися в словаре. При этом старые имена и соответствующие им словарные статьи сохраняются, но перестают быть приоритетными.

Возможность динамического переопределения слов весьма полезна для разработки сложных программ и уменьшения влияния «тирании слов» — этого бича Форт-систем программирования. Переопределение слов избавляет пользователя от необходимости держать в уме содержание словаря, включающего в себя сотни (нередко тысячи) слов.

Поясним это следующим примером:

```
ok
: DEMO ." HELLO MAN !" ; ok
DEMO HELLO MAN !ok
: DEMO ." HELLO FRIEND !" ;
DEMO is redefined ok
DEMO HELLO FRIEND !ok
FORGET DEMO ok
DEMO HELLO MAN !ok
FORGET DEMO ok
DEMO

Error: "DEMO" is undefined

ok
```

Здесь вначале задано слово DEMO, выводящее текст HELLO MAN!. Затем вновь задано слово DEMO, но выводящее текст HELLO FRIEND!. Это сопровождается предупреждением: DEMO is redefined (DEMO переопределено). Теперь указание слова DEMO выводит вторую надпись.

А сохранилось ли первое слово DEMO? Ответ утвердительный. Сотрем слово DEMO командой FORGET, а затем вновь используем его. Получим первую надпись. Еще раз сотрем DEMO командой FORGET. Теперь уже получим сигнал ошибки: "DEMO" is undefined ("DEMO" не определено).

Следующий пример иллюстрирует создание с помощью слова VOCABULARY трех подсловарей DEMO1 (со словами A1, A2 и A3), DEMO2 (со словами B1 и B2) и DEMO3 (со словом C1):

```
ok
VOCABULARY DEMO1 ok
VOCABULARY DEMO2 ok
VOCABULARY DEMO3 ok
DEMO1 DEFINITIONS ok
: A1 ." WORD A1 " ; ok
: A2 ." WORD A2 " ; ok
: A3 ." WORD A3 " ; ok
DEMO2 DEFINITIONS ok
: B1 ." WORD B1 " ; ok
: B2 ." WORD B2 " ; ok
DEMO3 DEFINITIONS ok
: C1 ." WORD C1 " ; ok
```

Для ввода подсловарей в действие и их заполнения словами применяется слово DEFINITION.

После ввода этого слова можно вводить обычным образом новые слова. Все они будут попадать в подсловарь, который был задан последним. Это обеспечивает возможность задания сложной древовидной структуры словарей.

Использовать в данный момент можно только слова, входящие в текущий по поиску словарь. Он указывается своим именем:

```
DEMO1 ok
A1 WORD A1 ok
A2 WORD A2 ok
A3 WORD A3 ok
B1
```

Error: "B1" is undefined

```
ok
DEMO2 ok
B1 WORD B1 ok
B2 WORD B2 ok
C1
```

Error: "C1" is undefined

```
ok
DEMO3 ok
C1 WORD C1 ok
A1
```

Error: "A1" is undefined

ok

Соответственно слово WORDS выводит полный перечень слов указанного приоритетным словаря:

```
ok
DEMO1 ok
WORDS
```

```
545C A3                5448 A2                5434 A1
```

```
ok
DEMO2 ok
WORDS
```

```
5484 B2                5470 B1
```

```
ok
DEMO3 ok
WORDS
```

```
5498 C1
```

ok

Здесь интересно отметить, что слово WORDS, наряду с именами слов, выводит адреса компиляции их словарных статей.

Слово VOCS выявляет текущую структуру словаря и выводит таблицу с составом всех подсловарей:

VOCS

Search order:

Context voc = DEMO3

Current voc = DEMO3

Default voc = FORTH

Defined vocabularies:

link	nfa	name
5432	5424	DEMO3
5422	5414	DEMO2
5412	5404	DEMO1
3CB2	3CA3	HIDDEN
3CA1	3C8F	ASSEMBLER
3C8D	3C7E	EDITOR
1A4E	1A40	FORTH

Для вывода исходного набора слов версии PC/FORTH надо указать имя основного словаря FORTH или использовать только что загруженную систему. Тогда слово WORDS выведет листинг с адресами и именами всех слов основного словаря.

Лист 8.7

Адреса компиляции и имена слов словаря FORTH версии PC/FORTH (см. распечатку слов на стр. 277).

Как видно из листа 8.7, версия PC/FORTH содержит около 450 основных слов, т. е. является одной из наиболее полных версий языка Форт.

Слово FENCE обеспечивает защиту слов от стирания. Перегрузив систему, выполним следующие действия:

```
FENCE U. 44922 ok
: DEMO1 ." ONE " ; ok
: DEMO2 ." TWO " ; ok
: DEMO3 ." FREE " ; ok
: DEMO4 ." FOUR " ; ok
FENCE @ U. 21508 ok
' DEMO2 FENCE ! ok
FENCE @ U. 21535 ok
FORGET DEMO2
Error: "DEMO2" in protected dictionary

ok
FORGET DEMO3 ok
DEMO2 TWO ok
DEMO3
Error: "DEMO3" is undefined
```

5372 CHDIR	523E RUN	5112 KILL-TASKS	507B KILL
4FFB START	4FA2 .TASKS	4EAC VOCS	4DF7 WORDS
4DBB SHOW	4D28 TRIAD	4CCA LIST	4C5E INDEX
4C32 SCOPY	4C07 EDIT	47C6 DMAX	47AD DMIN
477C DU<	476E D0=	4759 2ROT	474B D>=
473D D<=	472F D<>	4714 D/	46FB D*
46D9 2CONSTANT	46B8 2VARIABLE	45F6 SHOWC	45AD .BLK#
4593 ."SCR#"	4537 .BANNER	4526 -ITAL	4516 ITAL
4505 -EMPH	44F5 EMPH	44E4 -COMP	44D4 COMP
44BF ESC-SEQ	44B3 FORMF	443B DIR	41D3 QX
41A8 LOAD-USING	4140 USING	40F0 PREV-FILE	407B .STACK
4008 #IN	3FD6 ENDCASE	3FB0 ENDOF	3F88 OF
3F72 CASE	3E9D DUMP	3D8F SAVE	3D5A FREEZE
3D3E LIT"	3D23 (LIT")	3D10 MYSELF	3CF2 ASCII
3CB4 .T	3CA3 HIDDEN	3C8F ASSEMBLER	3C7E EDITOR
3C73 TASK	3C68 NOOP	3C53 FORTH-83	3C0E DLITERAL
3BE2 LITERAL	3B69 NUMBER	3B1B CONVERT	3B00 LINE
3AE5 !DOT	3ACA @DOT	3AB3 @dot	3A9C !dot
3A85 line	36AC ARCSEG	3680 ARXY	366E SIN
365C COS	3545 BEEP	34F3 ABORT	3381 WARM
32EC COLD	31F5 BYE	31E4 DISPATCH	312E FIND-TASK
310A TASK_LIST	307D .FCB	3002 FORGET	2FE7 THRU
2FC3 -->	2F6C LOAD	2F50 [']	2F3A '
2F14)	2F04 (2EE7 REPEAT	2ECD AGAIN
2EBB WHILE	2EA1 UNTIL	2E8D BEGIN	2E62 ELSE
2E45 THEN	2E2A IF	2E19 BACK	2DF1 +LOOP
2DCA LOOP	2DAE ?DO	2D93 DO	2D74 ABORT"
2D61 .(2D46 ."	2D24 [COMPILE]	2D0A COMPILE

2CF5 CREATE	2CDB DEFINITIONS	2CCB]	2CBB [
2CA3 IMMEDIATE	2C91 LEAVE	2C7D ;	2C5E UNSMUDGE
2C41 SMUDGE	2C2E LATEST	2C18 (;CODE)	2C07 !CSP
2BB7 QUIT	2B55 INTERPRET	2B37 null	2AE6 WORD
2A7F FIND	2A53 QUERY	29AC EXPECT	299D */
2989 */MOD	297B MOD	295A MU/MOD	294D >=
2940 <=	2933 <>	2918 ?LOADING	28FF ?CSP
28E0 ?DECIMAL	28CB ?PAIRS	28B5 ?EXEC	2886 ?STACK
286E ?COMP	2854 MESSAGE	2839 ?ERROR	27F6 ERROR
278C SHOW-ERROR	2739 (ABORT")	271E (."	270C .LINE
26E5 (LINE)	26B8 (TYPE)	26A4 COUNT	2693 .R
2681 U.R	2672 U.	2666 .	2655 D.
2621 D.R	25FE SPACES	25EE SPACE	25D9 #S
25AF #	2598 SIGN	257F HOLD	2570 <#
2559 #>	254B TIB	2539 PAD	24D0 .SCREEN-FILE
248F .DOS	2456 .VERSION	2438 .NAME	2415 .CPU
2400 L>NAME	23EB N>LINK	23DD LINK>	23CF >LINK
23B7 NAME>	239F >NAME	2391 BODY>	2383 >BODY
236E BINARY	235A OCTAL	2348 HEX	2332 DECIMAL
231D C,	2309	22F1 ORIGIN+	22C1 R/W
2276 BUFFER	2204 BLOCK	21E5 UPDATE	219E FLUSH
214D SAVE-BUFFERS	2117 EMPTY-BUFFERS	20F0 +BUF	2058 CLOSE-SCR
2002 OPEN-SCR	1F34 BLK-WRITE	1E76 BLK-READ	1E29 SET-IO
1DC1 lseek	1D99 chdir	1D43 run	1D16 parse-filename
1CFA set-dta	1CDA block-write	1CBB block-read	1CA4 FDOS
1C8B NUL-FILE	1C6F DEFAULT-FILE	1C62 #TASKS	1C52 NEXT-LINK
1C48 C/L	1C3F BL	1C2E SCREEN-FCB	1C22 WSIZE
1C0F VIDEO-PARAMS	1BBB SCREEN-HANDLE	1BA9 BOOT-SCREEN	1B9D B/BUF
1B8C DISK-ERROR	1B82 REC	1B77 PREV	1B6D USE
1B61 LIMIT	1B55 FIRST	1B49 B/SCR	1B3B SEC/BLK

1B2F #BUFF	1B24 SPAN	1B1A HLD	1B10 CSP
1B06 DPL	1AFB BASE	1AEF STATE	1AE1 CURRENT
1AD3 CONTEXT	1AC9 SCR	1ABF BLK	1AB5 >IN
1AAB OUT	1A9C VOC-LINK	1A93 DP	1A87 FENCE
1A79 WARNING	1A6E #TIB	1A62 <TIB>	1A59 R0
1A50 S0	1A40 FORTH	19D4 VOCABULARY	19B2 DOES>
198A :	196C USER	194F VARIABLE	1932 CONSTANT
18FE BUILD	18C4 build	1874 .DATE	183A .TIME
1824 !TIME	180E @TIME	17F8 !DATE	17E2 @DATE
17CE ?DEVICE	17B0 @COM1	1796 PCKEY	177A KEY
1762 ?TERMINAL	172E (KEY)	16FA (?TERMINAL)	16E7 DEVICE
16BE COM1	1692 PRINTER	163E CONSOLE	1621 CR
1610 TYPE	15F7 EMIT	15DF (CEMIT)	15C7 (PEMIT)
156B PALETTE	152E FOREGROUND	14E3 BACKGROUND	14AE BORDER
13B5 MODE	1363 ?MODE	1329 B/W	12DF INTENSITY-OFF
1299 INTENSITY	1279 BLINK-OFF	125D BLINK	1245 XOR-OFF
122E XOR-ON	1214 UNDERLINE	11F8 REVERSE-OFF	11E0 REVERSE
11CD ?VPAGE	10C9 (GEMIT)	0FFB type	0FD1 VPAGE
0EAB (EMIT)	0E85 ?dos	0E78 ?cs:	0E65 ?ramsize
0E52 ?options	0E34 video-io	0E12 CLEARSCREEN	0DFD GOTOXY
0DCA SET-CURSOR	0D9F init-window	0D8A gotoxy	0D32 clrscr
0D19 (+LOOP)	0CF7 (LOOP)	0CDF (LEAVE)	0CC8 (?DO)
0CA9 (DO)	0C98 ;S	0C85 EXIT	0C6D OBRANCH
0C5B BRANCH	0C49 BLIT	0C37 DLIT	0C28 LIT
0C0A ALLOT	0BF1 HERE	0BD0 S=	0BC0 PC!
0BB1 P!	0B9F PC@	0B90 P@	0B4E MATCH
0B2E MOVE	0B0C CMOVEL	0AE2 CMOVE>	0AC1 CMOVE
0A77 -TRAILING	0A67 BLANK	0A57 ERASE	0A3D FILL

0A21 MIN	0A05 MAX	09F1 DABS	09DC ABS
09C5 ?DNEGATE	09AF ?NEGATE	098E DNEGATE	097A NEGATE
0964 C!L	094C C@L	0937 !L	0922 @L
090E 2@	08FA 2!	08EA ><	08DA +!
08CA C!	08B8 C@	08A9 !	089A @
0887 4DROP	0874 3DROP	085B 2OVER	0848 2DROP
0831 2SWAP	081D 2DUP	080A ROT	07F5 ?DUP
07E5 DUP	07D2 OVER	07C0 SWAP	07B0 DROP
0780 ROLL	076A PICK	0750 DEPTH	073F PERFORM
0730 EXECUTE	0720 R@	070E J	06FC I
06EA R>	06D8 >R	06CA RP!	06BC SP!
06AC RP@	069C SP@	0678 D=	065D D>
0642 D<	062B 0<>	0613 0>	05FD 0=
05E7 0<	05D0 U>	05B9 U<	05A3 >
058D <	0577 =	0566 NOT	0554 XOR
0543 OR	0531 AND	0520 S>D	050A D-
04F4 D+	04DF D2/	04CF 2/	04BF 2*
04AF 2-	04A0 1-	0490 2+	0481 1+
0471 -	0461 +	0434 /	0409 /MOD
03F2 M/MOD	039C M*	038C *	0365 UM/MOD
0350 UM*	031A find	02DA TOGGLE	02A4 DIGIT
0261 ENCLOSE			

ok

Вначале выводится адрес начальной ячейки самого слова FENCE. Затем задано четыре слова

DEMO1, DEMO2, DEMO3 и DEMO4.

Слова

' DEMO2 FENCE !

задают защиту слова DEMO2. Теперь попытка стереть слова, начиная со слова DEMO2, словами FORGET DEMO2 оказывается безуспешной — выводится сообщение об ошибке (нельзя стирать запрещенное слово). Однако стирание со слова DEMO3 проходит, естественно, успешно.

Как отмечалось, PC/FORTH позволяет создавать словарные статьи, содержащие отсутствующие в словаре имена. При этом такие имена отмечаются как неопределенные. Естественно, что исполнение таких словарных статей возможно только после определения всех слов. Таким образом, PC/FORTH поддерживает технологию программирования сверху вниз. При работе со словарем и системой PC/FORTH удобны слова WARM (старт с сохранением введенных слов), COLD (старт со стиранием введенных слов), BYE (возврат в ОС) и др.

§ 8.7. Арифметические и логические операции PC/FORTH

PC/FORTH имеет типовой набор арифметических операций над числами одинарной разрядности, присущий целочисленным версиям языка Форт:

```
ok
123 . 123 ok
123 456 + . 579 ok
123 456 - . -333 ok
123 5 * . 615 ok
123 5 / . 24 ok
123 5 MOD . 3 ok
123 5 /MOD . 24 ok
123 456 MIN . 123 ok
123 456 MAX . 456 ok
123 NEGATE . -123 ok
-123 ABS . 123 ok
30 SIN . 5000 ok
30 COS . 8660 ok
123 1+ . 124 ok
123 2+ . 125 ok
123 1- . 122 ok
123 2- . 121 ok
123 2* . 246 ok
123 2/ . 61 ok
```

К новым словам в этом наборе относятся слова для быстрых целочисленных вычислений тригонометрических функций (SIN и COS):

SIN φ° — — — int (10000·sin φ°)

COS φ° — — — int (10000·cos φ°)

Их аргумент задается в градусах, а результат есть целая часть произведения 10000 на значение функции. Включение в состав базового набора слов SIN и COS облегчает построение графиков различных функций в полярной системе координат и реализацию лого-графики.

Имеется и достаточно обширный набор слов для операций с числами двойной разрядности и комбинированных действий (см. примеры ниже):

```
ok
12.345 .654321 D+ D. 666666 ok
12345. 654321. D+ D. 666666 ok
654321. 123455. D- D. 530866 ok
123456. 5 D* D. 617280 ok
123456. 5. D* D. 0 ok
65432. 124 D/ D. 527 ok
40000 125 + . -25411 ok
40000 125 + U. 40125 ok
25000 2 * . -15536 ok
25000 2 * U. 50000 ok
40000 12 UM* D. 480000 ok
900000. 127 UM/MOD . . 7086 78 ok
900000 127 UM/MOD . . -1 -1 ok
```

Следующий пример поясняет действие слов ?NEGATE и ?DNEGATE:

```
ok
123 -1 ?NEGATE . -123 ok
123 1 ?NEGATE . 123 ok
123456. -1 ?DNEGATE D. -123456 ok
123456. 1 ?DNEGATE D. 123456 ok
```

Эти слова выводят на вершину стека первое число со знаком второго числа.

Набор слов для выполнения арифметических операций над числами с одинарной и двойной разрядностью в версии PC/FORTH функционально достаточно полный. Он позволяет создавать новые слова для выполнения операций над числами с фиксированной и плавающей точкой, операций с произвольной разрядностью операндов и результатов вычислений. Таким образом, имеется возможность для реализации достаточно сложных вычислений.

В PC/FORTH входят средства для преобразования чисел с разным основанием (десятичным DECIMAL, двоичным BINARY, восьмеричным OCTAL и шестнадцатеричным HEX). Ниже показано задание слова BASE?, сообщающего о текущем содержании переменной BASE,

несущей значение основания чисел, а также показаны типовые операции преобразования чисел:

```
ok
: BASE? BASE @ DUP DECIMAL . BASE ! ; ok
DECIMAL BASE? 10 ok
BINARY BASE? 2 ok
OCTAL BASE? 8 ok
HEX BASE? 16 ok
HEX 9 1+ . A ok
F 2+ . 11 ok
ABC 3 + . ABF ok
HEX AAA DECIMAL . 2730 ok
DECIMAL 2730 HEX . AAA ok
DECIMAL 1000 BINARY . 1111101000 ok
BINARY 1111101000 DECIMAL . 1000 ok
BINARY 1111101000 HEX . 3E8 ok
DECIMAL 1000 HEX . 3E8 ok
```

О результатах логических операций

AND, OR, XOR и NOT

можно судить по следующим примерам:

```
ok
0 0 AND . 0 ok
1 0 AND . 0 ok
0 1 AND . 0 ok
1 1 AND . 1 ok
0 0 OR . 0 ok
1 0 OR . 1 ok
0 1 OR . 1 ok
1 1 OR . 1 ok
0 0 XOR . 0 ok
1 0 XOR . 1 ok
0 1 XOR . 1 ok
1 1 XOR . 0 ok
0 NOT . -1 ok
-1 NOT . 0 ok
1 NOT . -2 ok
123 NOT . -124 ok
10 10 AND . 10 ok
123 0 XOR . 123 ok
```

Несколько последних примеров из приведенных показывают, что если эти операции выполняются не над состояниями флагов, то результат может оказаться не тривиальным.

Так, при задании отличных от 0 или 1 операндов в трех последних примерах результат отличается от ожидаемых значений 0 или 1. Это обстоятельство необходимо учитывать при использовании слов, реализующих логические операции.

Результатом проверки арифметических условий всегда являются состояния флагов (0, если условие не выполнено, и -1, если оно выполнено):

```

-10 0< . -1 ok
10 0< . 0 ok
0 0= . -1 ok
1 0= . 0 ok
123 0> . -1 ok
-123 0> . 0 ok
12 0<> . -1 ok
0 0<> . 0 ok
1 2 < . -1 ok
2 1 < . 0 ok
23 23 = . -1 ok
22 32 = . 0 ok
32 22 > . -1 ok
22 32 > . 0 ok
2 3 <> . -1 ok
2 2 = . -1 ok
2 2 <> . 0 ok
0. D0= . -1 ok
123. D0= . 0 ok
123. 456. D< . -1 ok
456. 123. D< . 0 ok
40000 50000 U< . -1 ok
50000 40000 U< . 0 ok

```

Большое число условий, реализуемых в версии PC/FORTH, облегчает построение управляющих структур.

§ 8.8. Операции со стеком PC/FORTH

Операции со стеком в версии PC/FORTH в основном обычные. Их смысл можно выявить, используя операцию вывода числа из стека . (точка):

```

1 2 3 4 5 . . . . . 5 4 3 2 1 ok
1 2 3 4 5 DROP . . . . . 4 3 2 1 ok
1 2 3 DUP . . . . . 3 3 2 1 ok
1 2 3 ?DUP . . . . . 3 3 2 1 ok
1 2 0 ?DUP . . . . . 0 2 1 ok
1 2 3 OVER . . . . . 2 3 2 1 ok
1 2 3 4 5 4 PICK . . . . . 1 5 4 3 2 1 ok
1 2 3 4 5 3 PICK . . . . . 2 5 4 3 2 1 ok
1 2 3 4 5 2 PICK . . . . . 3 5 4 3 2 1 ok
1 2 3 4 5 ROT . . . . . 3 5 4 2 1 ok
1 2 3 4 5 SWAP . . . . . 4 5 3 2 1 ok
1 2 3 4 5 DEPTH . . . . . 5 5 4 3 2 1 ok
1 2 3 4 5 4 ROLL . . . . . 1 5 4 3 2 ok
1 2 3 4 5 3 ROLL . . . . . 2 5 4 3 1 ok
1 2 3 4 5 2 ROLL . . . . . 3 5 4 2 1 ok
1 2 3 4 5 ABORT ok
.

```

Error: "." empty stack

В версии PC/FORTH имеется также ряд двойных операций:

```
ok
123. 456. 2DROP D. D. 123
```

```
Error: "D." empty stack
```

```
ok
123. 456. 2SWAP D. D. 123 456 ok
123. 456. 2DUP D. D. D. 456 456 123 ok
123. 456. 789. 2OVER D. D. D. D. 456 789 456 123 ok
```

Не совсем обычен результат действия операции .S (просмотр содержимого стека):

```
ok
12 34 56 78 ok
.S
78 ( 4E h)
56 ( 38 h)
34 ( 22 h)
12 ( C h)
ok
. . . . 78 56 34 12 ok
```

Она наряду с десятичным представлением чисел дает (в скобках) и их шестнадцатеричные значения. Числа в стеке при этом сохраняются (для стирания чисел в стеке используется слово ABORT).

Обмен между арифметическим стеком и стеком возврата (и использование слов >R, R@ и R>) демонстрирует следующий пример:

```
ok
: 3N* DUP >R * ROT R@ * ROT R> * ROT ; ok
12 34 56 3 3N* . . . 168 102 36 ok
```

Таким образом, работа стека в версии PC/FORTH не имеет существенных особенностей в сравнении с ранее описанными версиями.

§ 8.9. Типы данных и общение с памятью в версии PC/FORTH

Задание типов данных и их использование в версии PC/FORTH традиционно для форт-систем. Поэтому ограничимся приведением ряда практических примеров.

Эти примеры, в основном, повторяют приведенные ранее. Следует, однако, обратить внимание на возможность динамического переопределения констант и переменных (по аналогии с переопределением слов).

Задание, использование и стирание констант одинарной разрядности:

```
ok
10000 CONSTANT 10K ok
2718 CONSTANT eK ok
10K . 10000 ok
10K 123 + . 10123 ok
eK . 2718 ok
eK 2* . 5436 ok
1000 CONSTANT M ok
M . 1000 ok
10 CONSTANT M
M is redefined ok
M . 10 ok
FORGET M ok
M . 1000 ok
FORGET M ok
M .
```

Error: "M" is undefined

Задание и применение переменных одинарной разрядности:

```
ok
VARIABLE ALFA ok
VARIABLE BETA ok
12 ALFA ! ok
34 BETA ! ok
ALFA @ BETA @ + . 46 ok
ALFA @ . 12 ok
BETA @ . 34 ok
ALFA 123 ! BETA 456 ! ok
BETA @ ALFA @ / . 2 ok
ALFA @ . 12 ok
BETA @ . 34 ok
: ? @ . ; ok
ALFA ? 12 ok
BETA ? 34 ok
```

Задание и применение константы и переменной двойной разрядности:

```
ok
2718281. 2CONSTANT eM ok
eM D. 2718281 ok
2VARIABLE GAMMA ok
eM GAMMA 2! ok
GAMMA 2@ D. 2718281 ok
```

Средства PC/FORTH обеспечивают задание различных более сложных типов данных. Это обеспечивается традиционными для языка Форт средствами. Ниже даны примеры организации целочисленных массивов, таблиц и матриц.

Screen # 54

```

0 ( ARRAY, CARRAY, DARRAY, TABLE AND MATRIX )
1 : ARRAY ( n ARRAY name )
2     CREATE 2* ALLOT DOES> SWAP 2* +
3 ;
4 : CARRAY ( b CARRAY name )
5     CREATE ALLOT DOES> +
6 ;
7 : TABLE ( TABLE name n1 , n2 , ... nn , )
8     CREATE DOES> SWAP 2* + @
9 ;
10 : DARRAY ( n DARRAY name )
11     CREATE 4 * ALLOT DOES> SWAP 4 * +
12 ;
13 : MATRIX ( m n MATRIX name ) CREATE SWAP 1+
14     SWAP DUP , * 2* ALLOT DOES> ROT OVER @
15     * ROT + 2* + 2+ ;
ok

```

Для организации массивов, таблиц и матриц используется расширение адресного пространства с помощью слова ALLOT. Длина необходимого адресного пространства определяется числом элементов и длиной каждого элемента (в ОЗУ ПЭВМ).

Слова ARRAY, CARRAY и DARRAY задают одномерные массивы соответственно для чисел одинарной разрядности, кодов (байтов) и чисел двойной разрядности. Как это делается, поясняют следующие примеры:

```

ok
3 ARRAY A ok
123 0 A ! ok
456 1 A ! ok
789 2 A ! ok
0 A @ . 123 ok
1 A @ . 456 ok
2 A @ . 789 ok
3 CARRAY B ok
12 0 B C! ok
34 1 B C! ok
56 2 B C! ok
0 B C@ . 12 ok
1 B C@ . 34 ok
2 B C@ . 56 ok
3 DARRAY C ok
123456. 0 C 2! ok
234567. 1 C 2! ok
345678. 2 C 2! ok
0 C 2@ D. 123456 ok
1 C 2@ D. 234567 ok
2 C 2@ D. 345678 ok

```

Применение слов TABLE и MATRIX поясняют другие примеры:

```
ok
TABLE D 12 , 34 , 56 , 78 , ok
0 D . 12 ok
2 D . 56 ok
3 D . 78 ok
3 2 MATRIX E ok
123 0 0 E ! ok
456 1 0 E ! ok
987 2 1 E ! ok
0 0 E @ . 123 ok
1 0 E @ . 456 ok
2 1 E @ . 987 ok
```

Еще один путь задания массива ARRAY в виде переменной иллюстрируется следующими примерами:

```
ok
0 VARIABLE ARRAY ok
4 ALLOT ok
10 ARRAY ! ok
20 ARRAY 2+ ! ok
30 ARRAY 4 + ! ok
ARRAY @ . 10 ok
ARRAY 2+ @ . 20 ok
ARRAY 4 + @ . 30 ok
```

Можно ввести слова для прямого доступа в такой массив через указание индексов элементов:

```
ok
: IARRAY DUP + ARRAY + ! ; ok
: @ARRAY DUP + ARRAY + @ ; ok
: ?ARRAY @ARRAY . ; ok
10 0 IARRAY ok
20 1 IARRAY ok
30 2 IARRAY ok
0 @ARRAY . 10 ok
1 ?ARRAY 20 ok
2 ?ARRAY 30 ok
```

Аналогично можно задать данные в виде таблицы чисел одинарной разрядности:

```
ok
VARIABLE DATA ok
10 DATA ! ok
20 , 30 , 40 , ok
DATA @ . 10 ok
DATA 2+ @ . 20 ok
DATA 4 + @ . 30 ok
DATA 6 + @ . 40 ok
```


или таблицы кодов (байтов):

```
ok
VARIABLE C DATA ok
65 C DATA C! ok
66 C, 67 C, 68 C, ok
C DATA C@ . 65 ok
C DATA 1+ C@ . 0 ok
C DATA 2+ C@ . 66 ok
C DATA 3 + C@ . 67 ok
C DATA 4 + C@ . 68 ok
```

Приведем также ряд примеров прямого общения с памятью. Запись чисел одинарной разрядности (слово !), кодов (слово C!) и чисел двойной разрядности (слово 2!), а также их считывание (слова @, C@ и 2@) иллюстрируют следующие примеры:

```
ok
456 30000 ! ok
30000 @ . 456 ok
125 40000 C! ok
40000 C@ . 125 ok
123456. 50000 2! ok
50000 2@ D. 123456 ok
```

Заполнение ячеек ОЗУ любыми кодами (слово FILL), кодом пробела 32 (слово BLANK) и кодом 0 (слово ERASE) с последующей проверкой содержимого ячеек ОЗУ демонстрирует еще одна группа примеров:

```
ok
50000 3 65 FILL ok
50000 C@ . 65 ok
50001 C@ . 65 ok
50002 C@ . 65 ok
50000 3 BLANK ok
50000 C@ . 32 ok
50001 C@ . 32 ok
50002 C@ . 32 ok
50000 3 ERASE ok
50000 C@ . 0 ok
50001 C@ . 0 ok
50002 C@ . 0 ok
```

Наличие этих слов обеспечивает практический полный доступ к памяти. Однако (без сегментации памяти) максимальное число доступных ячеек ОЗУ составляет 65536. Это ограничение унаследовано от ранних версий языка Форт для 8-разрядных микро-ЭВМ.

Для переноса n чисел одинарной разрядности в области ОЗУ с начальным адресом A_1 в новую область с начальным адресом A_2 используется команда $A_1 A_2 n$ MOVE.

Аналогичная операция для кодов (байтов) выполняется операцией $A_1 A_2 n$ CMOVE. Это иллюстрируют следующие примеры:

```
ok
1234 50000 ! ok
2345 50002 ! ok
4321 50004 ! ok
50000 60000 3 MOVE ok
60000 @ . 1234 ok
60002 @ . 2345 ok
60004 @ . 4321 ok
123 51000 C! ok
45 51001 C! ok
78 51002 C! ok
51000 55000 3 CMOVE ok
55000 C@ . 123 ok
55001 C@ . 45 ok
55002 C@ . 78 ok
```

§ 8.10. Организация ввода и вывода данных в версии PC/FORTH

Помимо вывода чисел из стека словом и просмотра содержимого стека словом `.S` PC/FORTH обладает всеми возможностями для осуществления форматированного вывода данных и интерактивного их ввода.

Для печати табличных данных удобны слова `.R`, `D.R`, `U.R`

```
ok
1 5 .R      1ok
12 5 .R     12ok
123 5 .R    123ok
1234 5 .R   1234ok
12345 5 .R  12345ok
1 1 .R      1ok
1 2 .R      1ok
1 3 .R      1ok
1 4 .R      1ok
1.5 .R      15ok
1. 5 D.R    1ok
12. 5 D.R   12ok
123. 5 D.R  123ok
1 5 U.R     1ok
12 5 U.R    12ok
123 5 U.R   123ok
```

Перевод строки обеспечивает слово `CR`, пробел — слово `SPACE`, а n пробелов — конструкция n `SPACE`. Даже такие простые средства обеспечивают создание довольно сложных табличных данных. Так, в слове `.ASCII` используются лишь слова `CR` и `SPACES` для формирования таблицы кодов ASCII и соответствующих им символов:

```
ok
: .ASCII CR CR DO I DUP 5 .R 2 SPACES EMIT LOOP CR ; ok
92 32 .ASCII
```

Слово CODECHAR иллюстрирует вывод кодов символов:

```
ok
: CODECHAR 32 WORD HERE 1+ C@ . ; ok
CODECHAR A 65 ok
CODECHAR C 67 ok
CODECHAR ABCD 65 ok
QUERY 123 456 + . 579 ok
```

Форматированный вывод с помощью шаблона <# # #> поясняет следующий пример:

```
ok
: C. <# # # # #> TYPE ; ok
1. C. 001ok
12. C. 012ok
123. C. 123ok
1234. C. 234ok
12345. C. 345ok
```

А ниже показано применение шаблона для вывода времени (слово HMS) и номеров телефона (слово TLF):

```
Screen # 71
0 : HMS
1 <# # # 58 HOLD # # 58 HOLD # # #>
2 CR TYPE CR
3 ;
4 : TLF
5 ." TLF NUMBER "
6 <# # # 45 HOLD # # 45 HOLD #S #>
7 TYPE CR
8 ;
9
10
11
12
13
14
15
ok
110845. HMS
11:08:45
ok
1234567. TLF TLF NUMBER 123-45-67
ok
```

Для ПЭВМ со знакогенератором, содержащим буквы кириллицы, PC/FORTH обеспечивает вывод текстов и задание слов с использованием кириллицы:

```
ok
: ДЕМО ." Вывод текста на русском языке " ; ok
ДЕМО Вывод текста на русском языке ok
: ПРИМЕР ." Слово с русским именем " ; ok
ПРИМЕР Слово с русским именем ok
```

Для создания средств выполнения операций с плавающей точкой служит слово DECPOINT, вычисляющее позицию разделительной точки от конца числа двойной разрядности:

```
ok
: DECPOINT DPL @ . ; ok
.123456789 DECPOINT 9 ok
123.456789 DECPOINT 6 ok
12345678.9 DECPOINT 1 ok
123456789. DECPOINT 0 ok
```

Типичные средства ввода PC/FORTH представлены словами KEY (ввод в стек кода нажимаемой клавиши), EMIT (замена кода символом), BL (ввод кода 32 пробела), EXPECT (ввод кодов символьной строки длиной n с адреса A , т. е. в виде $A n$ EXPECT). Ввод возможен и в текстовый буфер, начальный адрес которого хранится в виде значения переменной PAD. Для ввода используется слово EXPECT, вывод осуществляется словом TYPE (в виде $A n$ TYPE), а при удалении пробелов после n становится слово — TRAILING. Эти (и некоторые иные) возможности иллюстрируют следующие примеры:

```
ok
KEY . 65 ok
65 EMIT Aok
65 EMIT SPACE A ok
65 EMIT 5 SPACES A      ok
BL . 32 ok
1000 100 BEEP ok
50000 6 EXPECT QWERTYok
50000 6 TYPE QWERTYok
50000 COUNT . U. 81 50001 ok
PAD @ . 8224 ok
PAD 12 EXPECT ABCDE QWERTYok
PAD 6 -TRAILING TYPE ABCDEok
SPAN @ . 8 ok
TIB U. 43808 ok
#TIB U. 44918 ok
```

Организация интерактивного (по запросу) ввода на PC/FORTH аналогична описанной для версии MVP-FORTH (см. § 8.2).

§ 8.11. Основные управляющие структуры PC/ FORTH

К основным управляющим структурам PC/FORTH относятся условные выражения IF...ELSE...THEN, циклы DO...LOOP, BEGIN...UNTIL, BEGIN...WHILE...REPEAT и другие служебные слова.

С помощью этих слов возможно создание программных структур практически любой сложности, а также более сложных управляющих структур.

Условные выражения иллюстрируются следующим типовым примером:

```
Screen # 72
0
1   : X?
2       CR
3       100 < IF
4           ." NUMBER < 100 "
5       ELSE
6           ." NUMBER >= 100 "
7       THEN
8   ;
9
10
11
12
13
14
15
ok
80 X?
NUMBER < 100 ok
120 X?
NUMBER >= 100 ok
```

В этой программе условное выражение используется для проверки числа X . Если $X < 100$, то выполняется фрагмент словарной статьи, следующий после слова IF, в противном случае — фрагмент после слова ELSE. Таким образом слово $X?$ анализирует значение X и сообщает меньше X числа 100 (или больше либо равно 100).

Вполне обычна и организация циклов:

```
ok
: DEMO1 5 0 DO I . LOOP ; ok
DEMO1 0 1 2 3 4 ok
: DEMO2 10 -4 DO I . 2 +LOOP ; ok
DEMO2 -4 -2 0 2 4 6 8 ok
: DEMO3 -10 4 DO I . -2 +LOOP ; ok
DEMO3 4 2 0 -2 -4 -6 -8 -10 ok
: DEMO4 1 BEGIN 2* DUP DUP . 100 > UNTIL DROP ; ok
DEMO4 2 4 8 16 32 64 128 ok .
: DEMO5 1 BEGIN DUP 100 > NOT WHILE 2* DUP . REPEAT DROP ; ok
DEMO5 2 4 8 16 32 64 128 ok
: DEMO6 3 0 DO ." I=" I . CR 6 1 DO I . LOOP CR LOOP ; ok
DEMO6 I=0
1 2 3 4 5
I=1
1 2 3 4 5
I=2
1 2 3 4 5
ok
```

Циклы обеспечивают увеличение или уменьшение значений целочисленной управляющей переменной I . Статус переменной I различен внутри и вне цикла. Для контроля за значениями управляющих переменных двойных циклов зарезервирована и переменная J .

Статус управляющих переменных I и J в двойных циклах выявляет следующий пример:

Screen # 70

```
0
1 : DEMO7 CR
2   3 0 DO
3     5 2 DO
4         ." J=" J
5         ." I=" I
6             LOOP
7             CR
8         LOOP
```

```
9
10
11
12
13
14
15
```

```
ok
DEMO7
J=0 I=2 J=0 I=3 J=0 I=4
J=1 I=2 J=1 I=3 J=1 I=4
J=2 I=2 J=2 I=3 J=2 I=4
ok
```

Для прерывания программы в цикле используется слово LEAVE. В любом месте исполнение программы прерывается словом EXIT. Их применение иллюстрируют два следующих примера:

```
ok
: DEMO8 10 0 DO I DUP . 5 > IF LEAVE ELSE THEN LOOP ; ok
DEMO8 0 1 2 3 4 5 6 ok
: DEMO9 10 0 DO I DUP . 5 > IF EXIT ELSE THEN LOOP ; ok
DEMO9 0 1 2 3 4 5 6
```

В некоторых случаях полезна организация цикла с заданным числом повторений. Такой цикл реализуется словом REP (от repeat — повторение):

```
ok
: REP 0 DO CR ." HELLO " LOOP ; ok
5 REP
HELLO
HELLO
HELLO
HELLO
HELLO ok
```

В приведенном примере слово REP используется для повторения *n* раз (*n*=5) слова HELLO.

Еще один присущий PC/FORTH тип управляющих структур — переключатели. Структура организации переключателя поясняется следующим примером:

```
Screen # 53
0 ( DEMONSTRATION CASE OPERATION )
1 : DEMO
2   CASE
3     0 OF ." NULL" ENDOF
4     1 OF ." ONE" ENDOF
5     2 OF ." TWO" ENDOF
6     3 OF ." FREE" ENDOF
7     4 OF ." FOUR" ENDOF
8   ENDCASE
9 ;
10
11
12
13
14
15
ok
0 DEMO NULLok
1 DEMO ONEok
2 DEMO TWOok
3 DEMO FREEok
4 DEMO FOURok
```

В данном случае слово DEMO преобразует цифру в ее полное символьное наименование.

§ 8.12. Управление компиляцией и системные переменные PC/FORTH

Действие команд управления режимом компиляции иллюстрирует следующий ряд примеров:

```
ok
: SUM 123 456 + . ; IMMEDIATE ok
SUM 579 ok
: SUM1 SUM ; 579 ok
SUM1 ok
: SUM2 [COMPILE] SUM ; ok
SUM2 579 ok
: SUM3 [ 123 456 + . ] ; 579 ok
STATE @ . 0 ok
: SUM4 123 456 + . ?EXEC ; ok
IMMEDIATE ok
: SUM5 SUM4 ; 579
```

Error: "SUM4" execution only

```
ok
SUM4 579 ok
```

Здесь вначале слову SUM (сложение чисел 123 и 456) с помощью слова IMMEDIATE придан признак немедленного исполнения. Исполнение слова с таким признаком происходит обычным образом (ввод SUM дает результат 579).

Задание слова SUM1, в словарную статью которого входит слово SUM, выявляет действие указанного признака. Теперь, хотя SUM заключено между знаками : и ; словарной статьи и нормально не должно исполняться в ходе ее задания, результат 579 появляется сразу. Слово [COMPILE] в следующем примере перед словом SUM придает последнему признак компиляции, т. е. отменяет действие слова IMMEDIATE.

В другом примере показано, что операции, охваченные квадратными скобками, также приобретают признак немедленного исполнения (см. задание слова SUM3). В остальных примерах показано действие слова ?EXEC, которое помечает слово SUM4 как слово, допустимое к использованию только во входном потоке. В слове SUM5, следовательно, применение слова SUM4 сопровождается выдачей сигнала ошибки. Однако его использование во входном потоке дает результат 579.

PC/FORTH имеет обширный набор системных переменных. Основные из них приведены ниже.

Системные переменные выдачи имени Форта, ДОС, версии языка, тип центрального процессора, времени, даты и номера текущего листа (экрана):

```
ok
FORTH-83 ok
.DOS 3.00ok
.VERSION 2.00ok
.CPU 8088 ok
.TIME 00:33ok
.T 00:34:00.09 ok
.DATE 01/01/80ok
."SCR#" Screen # ok
```

Системные переменные, относящиеся к блокам и буферам:

```
ok
BLK @ . 0 ok
1 BLOCK U. 47098 ok
2 BLOCK U. 48126 ok
2 BLOCK 1 BLOCK - . 1028 ok
1 BUFFER U. 46070 ok
2 BUFFER U. 48126 ok
1 DUP >IN @ . 12 ok
1 DUP DUP >IN @ . 16 ok
1 DUP DUP DUP >IN @ . 20 ok
```


Другие системные переменные:

```
C/L . 80 ok
SCREEN-FCB . 92 ok
WSIZE . 2 ok
SCREEN-HANDLE . 7117 ok
B/BUF . 1024 ok
SEC/BLK . 1 ok
REC U. 7050 ok
PREV U. 7040 ok
USE U. 7029 ok
LIMIT U. 49152 ok
FIRST U. 45040 ok
#BUFF U. 4 ok
SPAN U. 44966 ok
HLD U. 44964 ok
CSP U. 44962 ok
CONTEXT U. 44952 ok
SCR U. 44950 ok
OUT U. 44944 ok
VOC-LINK U. 44926 ok
WARNING U. 44920 ok
#TIB U. 44918 ok
<TIB> U. 44916 ok
R0 U. 44914 ok
S0 U. 44912 ok
```

Назначение этих и ряда других (ограниченных в большинстве применений) системных переменных подробно описано в гл. 6.

Для компиляции словарной статьи по ее адресу компиляции в PC/FORTH используется слово EXECUTE. В следующем примере исполняется слово SIN, а затем выявляется его адрес компиляции и, наконец, это же слово исполняется без указания его имени и с преобразованием имени в адрес компиляции:

```
ok
60 SIN . 8660 ok
' SIN U. 13940 ok
60 13940 EXECUTE . 8660 ok
60 ' SIN EXECUTE . 8660 ok
```

Слово EXECUTE, как отмечалось, облегчает реализацию структурного программирования сверху вниз.

§ 8.13. Примеры программ версии PC/FORTH

Для иллюстрации возможностей программирования на версии PC/FORTH мы воспользуемся несколькими простыми примерами из листов, входящих в состав системы.

Всего в эту систему входит более 50 листов с демонстрационными словарными статьями. Их изучение позволяет практически ознакомиться с многими полезными приемами программирования.

Лист 8.9

Вычисление целочисленного квадратного корня численным методом Ньютона

Screen # 7

```

0 ( Integer square root by Newton's method )
1
2 20 CONSTANT SQRN-#-ITER
3
4 ( n --- square-root )
5 : SQRN      DUP 0<
6             IF ." Illegal argument" 0 ERROR
7             THEN DUP
8             IF DUP 2/
9             SQRN-#-ITER 0
10            DO
11                2DUP / + 2/
12            LOOP
13            SWAP DROP
14            THEN ;
15
ok

```

Лист 8.10

Дампинг памяти (монитор). Лист задан в редакторе

Screen # 10 Using file: forth.scr Mode = Edit

```

0 ( Memory dump, byte format, intrasegment 02/26/82 )
1 FORTH DEFINITIONS DECIMAL
2 : DUMP ( addr n --- )
3   BASE @ >R HEX CR CR 5 SPACES
4   16 0 DO I 3 .R LOOP 2 SPACES
5   16 0 DO I 0 <# # #> TYPE LOOP CR
6   OVER + SWAP DUP 15 AND XOR DO
7   CR I 0 4 D.R SPACE
8   I 16 + I 2DUP
9   DO I C@ SPACE 0 <# # # #> TYPE LOOP
10  2 SPACES
11  DO I C@ DUP 32 < OVER 126 > OR IF DROP 46 THEN
12  EMIT LOOP
13  16 +LOOP CR R> BASE ! ;
14 ;S
15

```

Лист 8.10 иллюстрирует построение слова, по существу выполняющего типовые функции монитора для просмотра содержимого ячеек ОЗУ. Выводятся (в очень удобной форме) адреса ячеек, их содержимое в кодах и символах по ASCII.

Пример дампинга при шестнадцатеричном задании исходных данных:

```
ok
HEX 1480 100 DUMP
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1480	A3	0E	2E	0C	98	0E	BD	08	65	04	BD	08	A9	11	2E	0C
1490	00	00	B1	1A	AD	08	2E	0C	00	00	2E	0C	9B	0E	AD	08
14A0	2E	0C	00	00	2E	0C	9A	0E	CF	08	2F	13	8C	0C	86	42
14B0	4F	52	44	45	52	B5	13	A4	19	6B	13	2E	0C	04	00	91
14C0	05	77	0C	1C	00	DD	06	2E	0C	00	00	2E	0C	00	00	EF
14D0	06	2E	0C	00	0B	3F	0E	50	08	50	08	64	0C	04	00	B7
14E0	07	8C	0C	8A	42	41	43	4B	47	52	4F	55	4E	44	AE	14
14F0	A4	19	91	11	77	0C	1C	00	2E	0C	0F	00	37	05	2E	0C
1500	A1	0E	BD	08	2E	0C	30	00	37	05	48	05	BB	11	64	0C
1510	1C	00	2E	0C	10	00	90	03	2E	0C	99	0E	BD	08	2E	0C
1520	0F	00	37	05	48	05	2E	0C	99	0E	CF	08	8C	0C	8A	46
1530	4F	52	45	47	52	4F	55	4E	44	E3	14	A4	19	6B	13	2E
1540	0C	06	00	7B	05	77	0C	08	00	F0	14	64	0C	1C	00	2E
1550	0C	07	00	37	05	2E	0C	99	0E	BD	08	2E	0C	F8	00	37
1560	05	48	05	2E	0C	99	0E	CF	08	8C	0C	87	50	41	4C	45
1570	54	54	45	2E	15	A4	19	6B	13	2E	0C	04	00	38	29	77

ok

0123456789ABCDEF

```
.....e.....
.....
...../....B
ORDER....k.....
.w.....
.....?.P.P.d....
....BACKGROUND..
....w.....7...
.....0.7.H...d.
.....
..7.H.....F
OREGROUND....k..
...{.w.....d....
...7.....7
.H.....PALE
TTE....k.....8)w
```

Лист 8.11

Генератор случайных чисел

Screen # 15

```
0 ( Random number generator, by J E Rickenbacker )
1
2 FORTH DEFINITIONS DECIMAL
3 VARIABLE SEED
4
```

```

5 : (RAND) SEED @ 259 * 3 + 32767 AND DUP SEED ! ;
6
7 : RANDOM (RAND) 32767 */ ;
8
9
10
11
12
13
14
15
ok
15 LOAD ok
1000 RANDOM . 0 ok
1000 RANDOM . 23 ok
1000 RANDOM . 165 ok
1000 RANDOM . 800 ok
1000 RANDOM . 339 ok

```

Слово RANDOM обеспечивает генерацию целых случайных чисел в интервале $[0, N]$ и используется в виде

N RANDOM

Рекомендуем читателю разобраться в весьма простом алгоритме, реализующем это слово.

§ 8.14. Декомпиляторы PC/FORTH и анализатор памяти

Примером достаточно большой, а главное практически очень полезной, программы является программа-декомпилятор версии PC/FORTH. Она также входит в пакет демонстрационных программ версии PC/FORTH и доступна пользователю системой.

На стр. 300—302 представлены листинги семи листов (с номерами от 24 до 30), реализующих построение декомпилятора версии PC/FORTH:

```

Screen # 24
0 ( FORTH Decompiler, by Ray Duncan )
1
2 CR .( Wait ... loading Decompiler ) CR
3 FORTH DEFINITIONS DECIMAL
4 : TASK ;
5 VARIABLE QUIT.FLAG
6 VARIABLE WORD.PTR
7 VARIABLE NFA.PTR
8 : N. DUP DECIMAL . SPACE HEX 0 ." (" D. ." h) "
9 DECIMAL ;
10 : TAB-OR-RETURN OUT @
11 IF OUT @ 39 >
12 IF CR ELSE OUT @ 20 / 1+ 20 * OUT @ - SPACES
13 THEN
14 THEN ;
15 -->
ok

```

Screen # 25

```
0 ( FORTH Decompiler, continued )
1
2 ( find addresses of all special runtime routines )
3 ' (LOOP)          CONSTANT LOOP.ADR
4 ' LIT             CONSTANT LIT.ADR
5 ' BLIT           CONSTANT BLIT.ADR
6 ' DLIT           CONSTANT DLIT.ADR
7 ' :              @ CONSTANT DOCOL.ADR
8 ' OBRANCH        CONSTANT OBRANCH.ADR
9 ' BRANCH         CONSTANT BRANCH.ADR
10 ' (+LOOP)       CONSTANT PLOOP.ADR
11 ' (." )         CONSTANT PDOTQ.ADR
12 ' (ABORT" )     CONSTANT PABORTQ.ADR
13 ' (LIT" )       CONSTANT PLITQ.ADR
14 -->
15
ok
```

Screen # 26

```
0 ( FORTH Decompiler, continued )
1
2 ( find addresses of all special runtime routines )
3
4 ' (DO)           CONSTANT DO.ADR
5 ' (?DO)         CONSTANT ?DO.ADR
6 ' C/L           @ CONSTANT CONST.ADR
7 ' BASE          @ CONSTANT USERV.ADR
8 ' USE           @ CONSTANT VAR.ADR
9 ' (;CODE)       CONSTANT PSCODE.ADR
10 ' EXIT          CONSTANT EXIT.ADR
11 -->
12
13
14
15
ok
```

Screen # 27

```
0 ( FORTH Decompiler, continued                                     11/01/83 )
1 : PDOTQ.DSP      WORD.PTR @ 2+ DUP >R DUP C@ + 1- WORD.PTR !
2                R> COUNT TYPE ;
3 : WORD.DSP       >NAME DUP 1+ C@ 59 =
4                IF 1 QUIT.FLAG ! THEN .NAME ;
5 : BRANCH.DSP     WORD.PTR @ 2+ DUP WORD.PTR ! DUP @
6                + HEX U. DECIMAL ;
7 : USERV.DSP     ." User variable, current value = "
8                WORD.PTR @ 2+ C@ 8 ORIGIN+ @ + @ N.
9                1 QUIT.FLAG ! ;
10 : VAR.DSP       ." Variable, current value = "
11                WORD.PTR @ 2+ @ N. 1 QUIT.FLAG ! ;
12 : CONST.DSP     ." Constant, value = "
13                WORD.PTR @ 2+ @ N. 1 QUIT.FLAG ! ;
14 -->
15
```

Screen # 28

```
0 ( FORTH Decompiler, continued 11/01/83 )
1
2 : DIS
3 BL WORD FIND 0=
4 IF DROP 3 SPACES ." ? not in glossary " CR QUIT
5 THEN ( CFA ) DUP 2+ OVER @ =
6 IF 3 SPACES ." <primitive>" CR DROP QUIT
7 THEN 0 QUIT.FLAG ! DUP WORD.PTR ! >NAME NFA.PTR ! 2 SPACES
8 BEGIN WORD.PTR @ DUP HEX U. SPACE DECIMAL @ CASE
9 DOCOL.ADR OF CLEARSCREEN NFA.PTR @ REVERSE SPACE .NAME
10 REVERSE-OFF ." contains: " CR ENDOF
11 OBRANCH.ADR OF ." Jz " BRANCH.DSP ENDOF
12 BRANCH.ADR OF ." Jmp " BRANCH.DSP ENDOF
13 LOOP.ADR OF ." Loop " BRANCH.DSP ENDOF
14 PLOOP.ADR OF ." +Loop " BRANCH.DSP ENDOF
15 -->
```

Screen # 29

```
0 ( FORTH Decompiler, continued 11/01/83 )
1 LIT.ADR OF WORD.PTR @ 2+ DUP WORD.PTR ! @ N. ENDOF
2 BLIT.ADR OF WORD.PTR @ DUP 1+ WORD.PTR ! 2+ C@ N. ENDOF
3 DLIT.ADR OF WORD.PTR @ 2+ DUP 2+ WORD.PTR !
4 2@ SWAP D. ENDOF
5 PDOTQ.ADR OF ." Print: " PDOTQ.DSP ENDOF
6 PABORTQ.ADR OF ." Abort: " PDOTQ.DSP ENDOF
7 PLITQ.ADR OF ." String: " PDOTQ.DSP ENDOF
8 USERV.ADR OF USERV.DSP ENDOF
9 VAR.ADR OF VAR.DSP ENDOF
10 CONST.ADR OF CONST.DSP ENDOF
11 PSCODE.ADR OF WORD.PTR @ @ WORD.DSP 1 QUIT.FLAG ! ENDOF
12 DO.ADR OF WORD.PTR @ @ WORD.DSP 2 WORD.PTR +! ENDOF
13 ?DO.ADR OF WORD.PTR @ @ WORD.DSP 2 WORD.PTR +! ENDOF
14 -->
15
```

Screen # 30

```
0 ( FORTH Decompiler, continued 11/01/83 )
1
2 ( default case )
3 DUP EXIT.ADR = IF 1 QUIT.FLAG !
4 THEN DUP WORD.DSP
5 ENDCASE TAB-OR-RETURN
6 2 WORD.PTR +!
7 QUIT.FLAG @ ?TERMINAL OR
8 UNTIL CR ;
9
10 CR CR .( Decompiler loaded. ) SP@ DP @ - U. .( bytes left.)
11 CR .( To decompile word xxx type: DIS xxx <return> ) CR
12
13
14
15
```

Для запуска декомпилятора используется предложение

DIS Имя

завершаемое нажатием клавиши перевода строки.

Ниже дан пример декомпиляции словарной статьи системной переменной с именем VAR.DSP :

```
VAR.DSP contains:
5699 Print: Variable, current value = 56B6 WORD.PTR
56B8 @ 56BA 2+ 56BC @
56BE N. 56C0 1 (1 h) 56C3 QUIT.FLAG
56C5 ! 56C7 ;S
ok
HEX ' VAR.DSP U. 5697 ok
```

Декомпилятор дает распечатку адресов каждого слова словарной статьи (в шестнадцатеричном виде) и символьное представление слова. На двух листах с номерами 31 и 32 (см. стр. 304) задана реализация анализатора распределения памяти.

Для анализа памяти (вывода карты памяти) используется слово MAP. Результат его выполнения имеет вид:

```
ok
MAP

Address of NEXT = 230H
Top of dictionary = 7831H
Available room in dictionary = 13039 bytes
Base of data stack = AB20H
Start of terminal input buffer = AB20H
Size of terminal buffer = 80 bytes
Base of return stack = AF70H
Size of return stack = 1024 bytes
Start of user variables = AF70H
Start of disk buffer area = AFF0H
End of disk buffer area = C000H
Disk buffers available = 4
Disk buffer size = 1024 bytes
```

ok

Представленные данные дают наглядное представление о распределении ОЗУ на нужды форт-системы PC/FORTH.

§ 8.15. Экранные окна

Одним из удобных средств представления информации на экране дисплея ПЭВМ являются окна в экране. В каждое окно (а их может быть несколько) выводится своя информация (тестовая или графическая). В пакете демонстрационных программ версии PC/FORTH имеются и средства для задания окон — листы 33—36 (см. стр. 305—306).

Screen # 31

11/06/83)

```
0 ( Memory allocation map
1 FORTH DEFINITIONS HEX ~
2
3 : H. 0 4 D.R ." H" ;
4 : MAP HEX CR
5 CR ." Address of NEXT = " NEXT-LINK H.
6 CR ." Top of dictionary = " DP @ H.
7 CR ." Available room in dictionary = " SP@ DP @ -
8 DECIMAL U. ." bytes" HEX
9 CR ." Base of data stack = " S0 @ H.
10 CR ." Start of terminal input buffer = " TIB H. DECIMAL
11 CR ." Size of terminal buffer = " C/L . ." bytes" HEX
12 CR ." Base of return stack = " R0 @ H. DECIMAL
13 CR ." Size of return stack = " R0 @ TIB 50 +
14 - . ." bytes" HEX
15 -->
```

Screen # 32

```
0 ( Memory allocation, continued )
1
2 CR ." Start of user variables = " R0 @ H.
3 CR ." Start of disk buffer area = " FIRST H.
4 CR ." End of disk buffer area = " LIMIT H. DECIMAL
5 CR ." Disk buffers available = " #BUFF .
6 CR ." Disk buffer size = " B/BUF . ." bytes"
7 CR CR ; DECIMAL ;S
8
9
10
11
12
13
14
15
```


Screen # 33

```
0 ( window support                                     11/06/83 )
1 FORTH DEFINITIONS HEX
2
3 ( create wpb      compilation:  xul yul xlr ylr --- )
4 (                execution:    --- wpb-addr )
5 : WINDOW          CREATE 100 * + , 100 * + , 700 , DOES> ;
6
7 ( fetch parameters for VIDEO-IO call:  wpb --- dx cx bx )
8 : WPAR@           DUP @ SWAP 2+ DUP @ SWAP 2+ @ ;
9
10 ( change initializing attribute:  attrib wpb --- )
11 : W-ATTRIB       SWAP 100 * SWAP 4 + ! ;
12
13 ( execute window function:  dx cx bx ax --- )
14 : W-EXEC         video-io 2DROP 2DROP ;
15 -->
```

Screen # 34

```
0 ( window support, continued                          07/12/82 )
1
2 ( initialize window:  wpb --- )
3 : W-CLEAR           WPAR@ 0600 W-EXEC ;
4
5 ( scroll window up:  wpb --- )
6 : W-UP             WPAR@ 0601 W-EXEC ;
7
8 ( scroll window down:  wpb --- )
9 : W-DOWN           WPAR@ 0701 W-EXEC ;
10
11 ( cursor addressing within window:  x y wpb --- )
12 : W-GOTOXY        2+ @ DUP OFF AND SWAP 100 / D+ GOTOXY ;
13
14 ( move cursor to window home position:  wpb --- )
15 : W-HOME          2+ @ DUP OFF AND SWAP 100 / GOTOXY ;  -->
```

Screen # 35

```
0 ( window support, continued                                11/06/83 )
1
2 ( move cursor to window lower left corner:   wpb --- )
3 : W-LLC          DUP 2+ @ OFF AND SWAP @ 100 / GOTOXY ;
4
5 ( draw border around window:   wpb --- )
6 : W-BORDER      DUP >R 2+ @ DUP OFF AND SWAP 0 100 UM/MOD SWAP
7                DROP R> @ DUP OFF AND SWAP 0 100 UM/MOD SWAP DROP
8 ( top )        1 PICK 1+ 4 PICK DO I 3 PICK 1- GOTOXY C4 EMIT LOOP
9 ( bottom)     1 PICK 1+ 4 PICK DO I 1 PICK 1+ GOTOXY C4 EMIT LOOP
10 ( right )    DUP 1+ 3 PICK DO 1 PICK 1+ I GOTOXY B3 EMIT LOOP
11 ( left )     DUP 1+ 3 PICK DO 3 PICK 1- I GOTOXY B3 EMIT LOOP
12 ( ur )      1 PICK 1+ 3 PICK 1- GOTOXY OBF EMIT
13 ( ll )      3 PICK 1- 1 PICK 1+ GOTOXY OCO EMIT
14 ( lr )      1+ SWAP 1+ SWAP GOTOXY OD9 EMIT
15 ( ul )      1- SWAP 1- SWAP GOTOXY DA EMIT ; DECIMAL
```

Screen # 36

```
0 ( window examples                                        07/12/82 )
1 5 5 30 10 WINDOW W1
2 40 7 70 15 WINDOW W2
3 10 21 70 23 WINDOW W3
4
5 : DEMO  CLEARSCREEN  W1 W-BORDER W2 W-BORDER W3 W-BORDER
6         W1 W-CLEAR W2 W-CLEAR W3 W-CLEAR
7         W3 W-HOME ." We will scroll the left window up"
8         0 1 W3 W-GOTOXY ." and the right window down"
9         100 0 DO  W1 W-UP W1 W-LLC ." Line # " I .
10                W2 W-DOWN W2 W-HOME ." Line # " I .  LOOP
11         W3 W-CLEAR W3 W-HOME ." Demonstration is finished."
12         0 0 GOTOXY ;
13
14
15
```

Листы содержат необходимые комментарии, облегчающие их анализ (на английском языке).

Лист № 36, показанный на стр. 306, поясняет задание трех окон и демонстрационной программы DEMO. При указании слова DEMO на экране наблюдаются три окна с бегущими надписями в них. В конечном итоге наблюдается следующая картина:

ok

```
Line # 94
Line # 95
Line # 96
Line # 97
Line # 98
Line # 99
```

```
Line # 99
Line # 98
Line # 97
Line # 96
Line # 95
Line # 94
Line # 93
Line # 92
Line # 91
```

```
Demonstration is finished.
```

Представленные выше (§ 8.11—8.15) примеры программирования на языке PC/FORTH показывают его обширные возможности. Они эффективно дополняются средствами графики.

§ 8.16. Средства цветной графики и синтеза звука PC/FORTH

Версия PC/FORTH содержит типовые средства графики — установки цвета символов, основы, окаймления экрана (бордюра), установки точки и построения отрезков прямых.

В режиме среднего разрешения (320×200 точек) может выбираться одна из двух цветовых палитр, устанавливаемых словом

p PALETTE !

где $p=0$ и $p=1$ — требуемая палитра. В монохромном графическом режиме (высокого разрешения) число точек достигает 640×200 .

Для установки цвета знаков, основы и бордюра используются слова:

c FOREGROUND !

c BACKGROUND !

c BORDER !

где *c* — код цвета (от 0 до 3). Для каждой палитры существуют четыре цвета (один — цвет основы).

Слово CLEAR SCREEN обеспечивает очистку экрана, а слово

$x y$ GOTOXY

устанавливает курсор в позицию (x, y) , где x — номер столбца, y — номер строки.

Отображение знаков может быть реверсивным (слово REVERSE). Слово REVERSE — OFF отменяет режим реверсирования.

Для установки точки с координатами (x, y) и цветом с кодом c служит слово

$x y c$!DOT.

Слово

$x y$ @DOT

выдает код цвета c , если точка (x, y) есть на экране, и 0, если точки нет.

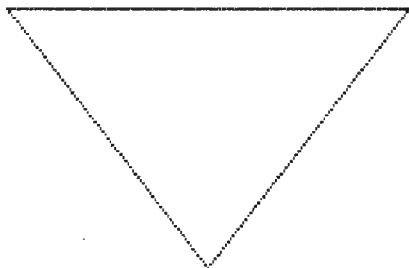
Слово

$x_1 y_1 x_2 y_2 c$ LINE

обеспечивает построение отрезка прямой, переходящей через точки (x_1, y_1) и (x_2, y_2) с цветом, заданным кодом c .

При переходе в графический режим текстовая информация отображается в формате 40×24 знаков. Следующие примеры иллюстрируют построение треугольника с помощью слов LINE, точки внутри нее словом !DOT и контроль наличия точек словами @DOT:

```
ok
250 10 500 10 1 LINE ok
500 10 375 90 1 LINE ok
375 90 250 10 1 LINE ok
375 50 1 !DOT ok
375 50 @DOT . 1 ok
375 51 @DOT . 0 ok
375 10 @DOT . 1 ok
```



В целом графические средства версии PC/FORTH соответствуют минимально необходимым для реализации прикладных графических программ. Однако расширения графических средств позволяют получать любые возможности. В частности, в § 9.8 описан полный комплект лого-графики для данной версии.

Для синтеза звуковых сигналов в версии PC/FORTH используется слово ВЕЕР. Звуковой сигнал с частотой f (в герцах) и длительностью $n \cdot 0,01$ задается словом ВЕЕР в виде

f n ВЕЕР

Наличие графических средств и возможности синтеза звуковых сигналов существенно расширяют области применения версии PC/FORTH и позволяют разрабатывать с ее помощью эффективные прикладные программы.

ЦВЕТНАЯ ГРАФИКА НА РАЗЛИЧНЫХ ВЕРСИЯХ ЯЗЫКА ФОРТ

§ 9.1. Концепции быстрой лого-графики

Типовые средства графики обычных версий языка Форт для персональных ЭВМ обычно имеют графические возможности, характерные для языка Бейсик: построение точки с заданными декартовыми координатами, отрезка прямой между двумя точками и, иногда, построение окружности. Создание более сложных и комбинированных геометрических фигур наталкивается на существенные трудности — нужно описать математически координаты всех характерных точек фигур и задать вычисления их в программе.

Одними из наиболее удачных по своей выразительности, простоте и удобству применения сейчас считаются графические средства лого-графики. Они являются главной отличительной чертой языка программирования Лого, первоначально разработанного С. Пейпертом для обучения детей основам культурного программирования и машинной графики. Эти средства в последние годы получили всеобщее признание и стали включаться в другие языки программирования: Бейсик, Паскаль, Си и Микро-пролог. В данной главе описываются расширенные пакеты слов лого-графики, реализованной на двух версиях языка Форт (с операциями над числами с плавающей запятой и целыми числами).

Как и в графике Бейсика, лого-графика имеет команды для задания точек с заданными координатами (x , y) и векторов — отрезков прямых. Однако отличительной чертой лого-графики является то, что эти средства рассматриваются как дополнительные.

Основным объектом лого-графики является точка в полярной системе координат. Эта точка может поворачиваться вокруг своей оси и передвигаться взад и вперед вдоль направления оси, ее радиус-вектора. Во многих лого-системах вместо точки применяется наблюдаемый на экране дисплея небольшой графический объект, отдаленно напоминающий черепашку (*turtle*). Отсюда происходит другое название лого-графики — TURTLE-графика (или графика черепашки). Черепашка снабжена световым пером. Опущенное перо чертит графики, поднятое — не чертит.

На рис. 9.1 показана условная черепашка в системе координат лого-графики. Угол поворота черепашки φ (*heading* или сокращенно HD) может задаваться любым (но приводится в отрезок $[0, 360^\circ]$). Главная задача лого-графики заключается в том, чтобы перевести точку-черепашку из исходного положения — точка (x, y) в новую точку (x_{NEW}, y_{NEW}) , лежащую на расстоянии L от исходной точки, задав вместо координат x_{NEW} и y_{NEW} угол φ и расстояние L (рис. 9.1).

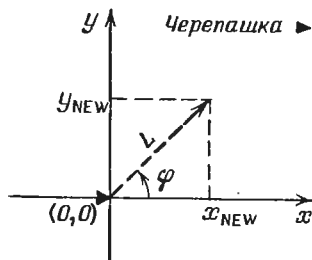


Рис. 9.1. «Черепашка» лого-графики в исходном состоянии и после ее перемещения

Поскольку язык Форт имеет слова DRAW или LINE для построения отрезков прямых, достаточно определить новые координаты:

$$x_{NEW} = x + LM_x \cos \varphi = x + \Delta x, \quad (9.1)$$

$$y_{NEW} = y + LM_y \sin \varphi = y + \Delta y. \quad (9.2)$$

Здесь (x, y) — старые декартовы координаты точки, M_x и M_y — масштабирующие множители для приращений по осям x и y координатной системы.

Построение отрезка прямой, соединяющего точки (x, y) и (x_{NEW}, y_{NEW}) , выполняется в следующем порядке.

1. По заданным L, M_x, M_y и φ находятся приращения:

$$DX = LM_x \cos \varphi = \Delta x, \quad (9.3)$$

$$DY = LM_y \sin \varphi = \Delta y. \quad (9.4)$$

2. Находятся новые координаты

$$\begin{aligned} X_{NEW} &= XCOR + DX, \\ Y_{NEW} &= YCOR + DY, \end{aligned} \quad (9.5)$$

где приняты обозначения координат системными переменными лого-графики (§ 9.2).

3. Если значение специальной управляющей переменной статуса светового пера PS (от слова *pen-status*) равно 0, перо считается опущенным и выполняется построение отрезка прямой, соединяющей точки $(XCOR, YCOR)$ и (X_{NEW}, Y_{NEW}) . Однако в этом случае построенные линии могут выглядеть неопрятными и содержать характерные «заусеницы». Это связано с тем, что при построении линий

обычными словами PLOT и DRAW происходит округление координат до целых чисел. Оно может быть различным для конца текущего и начала следующего отрезка.

В этом случае в месте стыковки отрезков и возникает характерная «заусеница» или скачок. Частично от этого недостатка можно избавиться, задавая XCOR и YCOR всегда целыми и вычисляя DX и DY из соотношений

$$DX = \text{Round}(LM_x \cos \varphi),$$

$$DY = \text{Round}(LM_y \sin \varphi),$$

где $\text{Round } x = \text{Int}(x + 0,5)$ — операция округления (Int — выделение целой части).

Однако и этот способ не гарантирует плавности переходов от одного отрезка к другому. Заметное улучшение качества построений достигается при использовании вместо слова DRAW специального слова DRAWXY, протягивающего линию от ранее установленной точки (оператором PLOT) до точки с координатами (XNEW, YNEW). Характерная особенность слова DRAWXY заключается в том, что в нем приращения Δx и Δy в слове

$\Delta x \ \Delta y \ \text{DRAW}$

находятся из соотношений

$$\Delta x = XNEW - XCOR',$$

$$\Delta y = YNEW - YCOR',$$

где XCOR' и YCOR' — значения XCOR и YCOR, получаемые как значения специальных системных переменных ПЭВМ, хранящих координаты построенной последней точки.

Подобный способ построения отрезков прямой исключает большие разрывы при переходе от одного отрезка к другому, поскольку построение каждого нового отрезка начинается точно с конца предыдущего. Погрешности при этом не накапливаются, так как координаты XNEW и YNEW все время обновляются (конец каждого отрезка строится по обновленным координатам).

4. Выполняется обмен значений x и y :

$$XCOR \leftarrow XNEW, \ YCOR \leftarrow YNEW$$

Эти операции реализуются словом FD (от *forward* — вперед). Оно обеспечивает перемещение черепашки вперед на расстояние L вдоль вектора с углом φ (рис. 9.1). Слово BK (*back* — назад) эквивалентно FD с заменой L на $-L$.

Слово SETX перемещает черепашку вдоль оси x без изменения φ . Для этого приращение DX задается прямо без вычислений по формулам (9.1) и (9.3). После этого выполняются пп. 2, 3 и 4 описанного алгоритма.

Слово SETY перемещает черепашку вдоль оси y без изменения φ . Для этого приращение DY задается непосредственно — формулы (9.2) и (9.4) не применяются, а затем реализуются пп. 2, 3 и 4 алгоритма.

Остальные слова лого-графики описаны ниже. Следует отметить, что вычисление DX и DY средствами языка Форт представляет определенные трудности. Так, нужно на каждом шаге вычислять $\sin \varphi$ и $\cos \varphi$. Это легко реализовать только в версиях языка Форт, имеющих функции вычисления синуса и косинуса с аргументом φ (HD) в виде чисел с плавающей запятой.

Кроме того, тригонометрические функции вычисляются довольно медленно. Поскольку точность их вычисления в системе FSP88 явно избыточна, для ускорения построений целесообразно использовать быстрые методы вычисления тригонометрических функций. Наиболее подходящими являются: применение табличного задания функции с интерполяцией их в промежутках между узлами и использование аппроксимаций. Это, однако, заметно усложняет программную реализацию лого-графики. В связи с этим в систему FSP88 включена реализация лого-графики с обычным вычислением тригонометрических функций. В то же время отмеченные методы вычисления этих функций используются при реализации лого-графики на целочисленных версиях языка Форт, также описанной в этой главе.

Следует отметить, что даже при поднятом пере все перемещения черепашки подчиняются заданным законам ее перемещения. Это позволяет легко создавать сложные фигуры, получаемые с помощью как видимых, так и невидимых фрагментов перемещения черепашки.

Масштабирующие множители M_x и M_y позволяют менять масштаб изображений по осям x и y . Это полезно, например, при построении эллипсов в виде сплюснутых окружностей. Если используется версия Форт с операциями над числами с плавающей запятой, целесообразно задавать M_x и M_y в виде обычных коэффициентов — в исходном состоянии $M_x = M_y = 1$. Для лого-графики на основе целочисленных версий языка Форт целесообразно задавать M_x и M_y в процентах от числа 100 и использовать целочисленные операции умножения на эти множители.

§ 9.2. Системные переменные лого-графики

Состояние графической лого-системы описывается рядом ее системных переменных. Их обозначения полностью соответствуют принятым сокращенным обозначениям языка Лого. Задание системных переменных представлено строками 2—8 листа 9.1.

Первый пакет слов лого-графики (LOGO1)

0 : c87 87 ;
 1 : c127 127 ;
 2 : HD H ;
 3 : MX V ;
 4 : MY U ;
 5 : PC Z ;
 6 : BG Z %1+ ;
 7 : BR Z %2+ ;
 8 : PS Z c3 %+ ;
 9 : DOT c87 + SW c127 + SW PLOT ;
 10 : PD C0 PS C! ;
 11 : PU C1 PS C! ;
 12 : SETPC DU PC C! INK ;
 13 : SETBG DU BG C! PAPER ;
 14 : SETBR DU BR C! BORDER ;
 15 : SETH H ! ;

Слова c87 и c127 — константы, дающие координаты точки в середине экрана. Переменные X и Y отведены под хранение текущих координат точки, а Z (побайтно) под хранение атрибутов точки.

Слово

HD — — — A_φ

оставляет на вершине стека начальный адрес ячеек ОЗУ, хранящих значение угла φ .

Слово

MX — — — A_{M_x}

оставляет на вершине стека начальный адрес ячеек ОЗУ, хранящих значение масштабного множителя M_x оси x .

Слово

MY — — — A_{M_y}

оставляет на вершине стека начальный адрес для масштабного множителя M_y по оси y .

Слово

PC — — — $A_{c_{\text{INK}}}$

оставляет на вершине стека начальный адрес для кода цвета c_{INK} .

Слово

BG — — — $A_{c_{\text{PAPER}}}$

оставляет на вершине стека начальный адрес для кода цвета страницы c_{PAPER} .

Слово

BR — — — $A_{c_{\text{BORDER}}}$

оставляет на вершине стека начальный адрес для кода цвета бордюра

c_{BORDER}

Слово

PS — — — $A_{c_{\text{PS}}}$

оставляет на вершине стека начальный адрес статуса светового пера (при $c_{\text{PS}}=0$ перо опущено и чертит линию, при $c_{\text{PS}}=1$ перо поднято и при движении линию не чертит).

Ниже дается краткое и полное обозначение системных переменных лого-графики:

Краткое название	Полное название	Назначение
X	XCOR	Координата x
Y	YCOR	Координата y
HD	HEADING	Угол поворота φ
MX	—	Масштаб по оси x
MY	—	Масштаб по оси y
PC	PENCOLOUR	Цвет пера
BG	BACKGROUND	Цвет основы
BR	BORDER	Цвет бордюра
PS	PENSTATUS	Статус пера

В описываемой здесь системе лого-графики x , y , φ , m_x и m_y могут задаваться в виде целых и дробных чисел. Это позволяет повысить точность предварительных вычислений новых координат. Остальные системные переменные целочисленные.

Системные переменные лого-графики обеспечивают оперативный контроль за ходом выполнения графических построений и полезны как для упрощения и повышения наглядности записи последующих слов лого-графики, так и для создания прикладных программ на ее основе.

В системе FSP88 как системные переменные лого-графики используется ряд глобальных переменных. Поэтому необходимо следить, чтобы они правильно использовались в прикладных программах лого-графики. При необходимости можно выделить под эти переменные отдельную область памяти и сделать работу лого-графики независимой от основных глобальных переменных.

§ 9.3. Задание атрибутов и исходного состояния лого-графики

Строки 10—15 листа 9.1 содержат слова лого-графики, необходимые для задания атрибутов, т. е. цветовых и других признаков графических объектов. Ниже подробно описывается их назначение.

Слово

PD — — — ($\text{PS} \leftarrow 0$)

придает системной переменной статуса светового пера PS значение 0 (перо опущено).

Слово

PU _ _ _ (PS←1)

придает системной переменной статуса светового пера PS значение 1 (перо поднято).

Слово

SETPS c_{INK} _ _ _ (PC← c_{INK})

задает цвет светового пера (кодом c_{INK}) и придает системной переменной PC значение c_{INK} .

Слово

SETBG c_{PAPER} _ _ _ (BG← c_{PAPER})

задает цвет страницы и придает системной переменной BG значение c_{PAPER} .

Слово

SETBR c_{BORDER} _ _ _ (BR← c_{BORDER})

задает цвет бордюра и придает системной переменной BR значение c_{BORDER} .

Слово

SETH φ _ _ _ (HD← φ)

придает системной переменной HD значение угла φ .

Еще три команды задания атрибутов приведены в строках 0, 1 и 2 листа 9.2 (см. § 9.4). Эти слова описаны ниже.

Слово

SETSCR M_x M_y _ _ _ (MX← M_x , MY← M_y)

задает значение M_x системной переменной MX и значение M_y — системной переменной MY.

Слово

SETPOS x y _ _ _ (XCOR← x , YCOR← y , NEWX← x , NEWY← y)

задает системной переменной X значение x , системной переменной Y — значение y , системной переменной NEWX — значение x и системной переменной NEWY — значение y .

Слово

CS _ _ _ (общая готовность)

обеспечивает общую готовность лого-системы. Очищает рабочую часть экрана, обнуляет ячейки области ОЗУ системных переменных, задает исходное состояние точки-черепашки в центре экрана (точка с $\varphi=0$ и

с координатами (127, 87) в растровой системе координат дисплея) и устанавливает $M_x = M_y = 1$.

Полные наименования слов для задания атрибутов указаны ниже:

Краткое название	Полное название	Назначение
PD	PENDOWN	Спуск светового пера
PU	PENUP	Подъем светового пера
SETPC	SETPENCOLOUR	Установка цвета пера
SETBG	SETBACKGROUND	Установка цвета основы
SETBR	SETBORDER	Установка цвета бордюра
SETH	SETHEADING	Установка угла φ
SETSCR	SETSCRUNCH	Установка масштабов по осям x и y
SETPOS	SETPOSITION	Установка x и y
CS	CLEARSCREEN	Очистка экрана и полная готовность

Словом CS рекомендуется пользоваться при первом применении лого-системы. Если нужно стереть экран, не меняя положения черепашки, следует применять слово CLS (в некоторых версиях Лого ему соответствует команда CLEAN).

§ 9.4. Операции построения графиков

В строках 3—15 листа 9.2 представлены листинги слов основных операций лого-графики.

Лист 9.2

Второй пакет слов лого-графики (LOGO2)

```

0 : SETSCR U ! V ! ;
1 : SETPOS DU Y ! W ! DU X ! Q ! ;
2 : CS CLS C0 DU SETPOS C1 DU SETSCR C0 SETH PD ;
3 : LT HD + ! ;
4 : RT NEG LT ;
5 : XYDRAW PC C@ INK c87 + YPLOT - SW c127 + XPLOT -
  SW DRAW ;
6 : dX L @ V @ * HD @ COSD * ;
7 : dY L @ U @ * HD @ SIND * ;
8 : newXY X@ dX + Q ! Y@ dY + W ! ;
9 : swapXY Q @ X ! W @ Y ! ;
10 : POS X@ Y@ ;
11 : fd POS DOT Q @ W @ XYDRAW ;
12 : FD L ! newXY PS C@ IF ELSE fd THEN swapXY ;
13 : BK NEG FD ;
14 : SETX X@ + Q ! PS C@ IF ELSE fd THEN swapXY ;
15 : SETY Y@ + W ! PS C@ IF ELSE fd THEN swapXY ;

```

Слово

LT $\Delta\varphi$ — — — (HD ← HD + φ)

обеспечивает поворот черепашки влево на угол $\Delta\varphi$ путем увеличения на величину $\Delta\varphi$ значения системной переменной HD.

Слово

RT $\Delta\varphi$ — — — (HD \leftarrow HD $-\varphi$)

обеспечивает поворот черепашки вправо на угол $\Delta\varphi$ путем уменьшения на величину $\Delta\varphi$ значения системной переменной HD.

Слово

XYDRAW x y — — —

строит отрезок прямой от начальной точки с координатами (XPLOT, YPLOT) до конечной точки с координатами (127 + x , 87 + y). Особенность построения — отсутствие разрыва в начальной точке. Напоминаем, что XPLOT и YPLOT — машинозависимые системные переменные, вызывающие значения координат начальной точки из ячеек ОЗУ с адресами 23677 и 23678.

Слово

dx — — — $LM_x \cos \varphi$

вычисляет проекцию вектора перемещения на ось x (DX), а слово

dy — — — $LM_y \sin \varphi$

— проекцию вектора перемещения на ось y (DY).

Слово

newXY — — — $x + DX$ $y + DY$

вычисляет новые координаты черепашки в виде суммы старых координат (значений переменных X и Y) с приращением DX и DY, вычисленными с помощью слов dx и dy. Значения NEWX и NEWY соответствуют вычисляемым по формулам (9.5).

Слово

swapXY — — — ($x \leftarrow x_{NEW}$, $y \leftarrow y_{NEW}$)

обеспечивает обмен значений X и Y, т. е. присвоение им значений x_{NEW} и y_{NEW} соответственно.

Слово

POS — — — x y

выводит на вершину стека значения системных переменных X и Y, т. е. текущие координаты.

Слово

fd — — —

строит точку с координатами (x , y) и соединяет ее отрезком прямой

с точкой (x_{NEW} , y_{NEW}) с помощью слова XYDRAW, независимо от статуса светового пера, т. е. значений переменной PS.

Слово

FD L _ _ _ _

перемещает черепашку вперед на расстояние L вдоль радиус-вектора с углом φ . Придает системной переменной L значение L . Поскольку переменная L является глобальной, ее применение вне лого-графики ограничено, так как приводит к изменению значения L .

Если системная переменная статуса светового пера PS имеет значение 0, слово FD оставляет след, в противном случае (значение PS есть 1) след не остается.

Слово

BK L _ _ _ _

перемещает черепашку назад на расстояние L (в остальном действия аналогичны описанным для слова FD).

Слово

SETX x _ _ _ _

перемещает черепашку вдоль оси x на расстояние x (вправо, если $x > 0$, и влево, если $x < 0$). Угол φ при этом не меняется. Если перо поднято, перемещение не оставляет след, если опущено — оставляет.

Слово

SETY y _ _ _ _

перемещает черепашку вдоль оси y на расстояние y (вверх, если $y > 0$, и вниз, если $y < 0$). Угол φ не меняется, при поднятом перо перемещение идет без следа, при опущенном — остается след.

Полное обозначение стандартных для Лого слов этой группы приводится ниже.

Краткое название	Полное название	Назначение
LT	LEFT	Поворот налево
RT	RIGHT	Поворот направо
FD	FORWARD	Перемещение вперед
BK	BACK	Перемещение назад
SETX	SETX	Перемещение вдоль оси x
SETY	SETY	Перемещение вдоль оси y

В описанную лого-систему не включены слова, ограничивающие перемещение черепашки рабочими участками экрана. Если черепашка с поднятым пером выходит за эти края, включается присущий Бейсику

ПЭВМ контроль, построения прерываются, выводится сигнал ошибки и система выходит из режима Форт в режим непосредственного исполнения команд на языке Бейсик. Исполнив команду GOTO 9 (или 1—8), можно вернуться в форт-систему и продолжить работу. С поднятым пером черепашка может совершать движение и за пределами экрана.

Исключены также два крайне редко используемых слова лого-графики: TOWARDS (дает значение угла φ вектора, вдоль которого должна двигаться черепашка, чтобы попасть в точку с координатами x и y) и WRAP (если черепашка встречает край экрана, она появляется с противоположной стороны). При необходимости эти команды легко реализовать в рамках данной системы.

Таким образом, описанная система лого-графики функционально повторяет графику языка Лого и имеет стандартную мнемонику команд. Это резко облегчает перевод программ с языка Лого на язык Форт, сводя его по существу к замене инфиксной формы записи команд на постфиксную. Прецизионная лого-графика системы FSP88 обеспечивает высокое качество построения как больших, так и малых графических объектов со скоростью в 1,3–1,5 раза большей, чем на языке Лого и при существенно меньших затратах памяти ОЗУ на реализацию прикладных программ.

§ 9.5. Программирование графических операций лого-графики системы FSP88

Программирование графических операций лого-графики рассмотрим с традиционного для этого вида графики примера. Пусть нужно построить 20 прямоугольников с соотношением сторон 2/1, вращающихся вокруг левого нижнего (для исходного прямоугольника) угла. Алгоритм построения и его реализация средствами лого-графики выглядят следующим образом:

1. Установка соотношения сторон

1.6 0.8 SETSCR

В данном случае $M_x=1,6$, $M_y=0,8$, $M_x/M_y=1,6/0,8=2$.

2. Установка исходной точки, например:

0 —30 SETPOS

3. Задание двадцатикратного цикла

20 0 % [... 18 RT %]

Здесь на место многоточия должны вписываться слова построения квадрата, из которого получаются многоугольники. В конце цикла стоят слова 18 RT поворота на угол $360^\circ/20=18^\circ$.

4. Построение квадрата четырехкратным перемещением черепашки вперед на 50 шагов и поворотом влево на угол 90°:

```
4 0 % [ 50 FD 90 LT % ]
```

В строках 0 и 1 приведенного ниже листа реализован этот алгоритм. Основным является слово DLOGO (демонстрационный пример лого-графики).

Лист 9.3

Слова расширенной графики для системы FSP88

```
0 : dlogo CS BLACK SETPC 1.6 0.8 SETSCR 0 -10 SETPOS  
1 ;  
1 : DLOGO dlogo 20 0 % [ 4 0 % [ 50 FD 90 LT % ] 18 RT  
% ] ;  
2 : xaxis 260 7 % [ DU %I SW PLOT 0 2 DRAW 24 % ] ;  
3 : XAXIS DU 7 SW PLOT 240 0 DRAW xaxis 127 SW PLOT  
0 4 DRAW ;  
4 : yaxis 164 12 % [ DU %I PLOT 2 0 DRAW 15 % ] ;  
5 : YAXIS DU 12 PLOT 0 150 DRAW yaxis 87 PLOT 4 0 DR  
AW ;  
6 : cs1 7 YAXIS 12 XAXIS C0 DU AT . ;  
7 : CS1 cs1 DU 2/ 21 15 AT . 21 30 AT . C0 21 C0 AT  
- ;  
8 : cs2 7 YAXIS 87 XAXIS DU 0 DU AT . NEG ;  
9 : CS2 cs2 21 C0 AT . DU 2/ 12 15 AT . 12 30 AT . ;  
10 : cs4 127 YAXIS 87 XAXIS DU C0 15 AT . NEG ;  
11 : CS4 cs4 21 15 AT . DU NEG 12 C0 AT . 12 30 AT .  
;  
12 : LCIRC 2ROT 87 + SW 127 + SW ROT CIRCLE ;  
13 : gf1 CLS M ! SW DU ROT CS1 M @ / 7 C0 F@ 150 * 1  
2 + PLOT M @ ;  
14 : gf2 * F@ 150 * 12 + BLACK DLINE ;  
15 : GF1 gf1 %I+ C1 %C DU %I 240 %* M @ / 7 + SW %I  
gf2 %J DR ;
```

На рис. 9.2 показан результат обращения к слову DLOGO. Не правда ли примечательно, что столь сложная фигура строится по

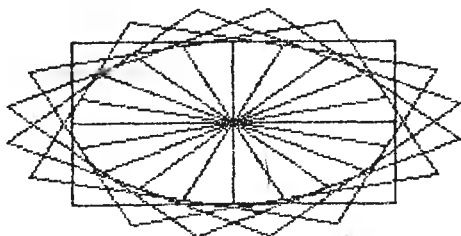


Рис. 9.2. Результат выполнения слова DLOGO

такой простой программе? В этом отчетливо проявляются преимущества лого-графики при построении сложных фигур вращения. Эту программу легко перестроить под любой многоугольник, получив таким образом множество новых красочных фигур.

Компоненты лого-графики можно (и нужно) компоновать с обычными компонентами графики, присущими современным форт-системам программирования. Такой путь позволяет существенно увеличить скорость построения графиков, поскольку из-за постоянного обращения к вычислению тригонометрических функций $\sin \varphi$ и $\cos \varphi$ лого-графика проигрывает по скорости построения простых фигур обычной графике. Тут термин «черепашня» графика иногда приобретает несколько иронический характер.

В качестве примера применения обычной графики в составе лого-графики рассмотрим построение координатных осей с масштабными делениями и их оцифровкой. Такое построение реализуется несколькими словами (указываем смысл основных слов).

Слово

HAXIS y_0 — — —

обеспечивает построение горизонтальной оси с координатой по вертикали y_0 . Меняя y_0 , можно поместить эту ось в любое место экрана. Ось имеет 10 делений, среднее деление вдвое длиннее, чем остальные.

Слово

YAXIS x_0 — — —

обеспечивает построение вертикальной оси с координатой по горизонтали x_0 . Меняя x_0 , можно поместить эту ось в любое место экрана. Ось имеет 10 делений, среднее вдвое длиннее остальных.

Слово

CS1 x_{\max} y_{\max} — — —

строит координатную систему с одним квадрантом и выполняет оцифровку масштабных делений, проставляя значения x_{\max} и y_{\max} у конечных делений и $x_{\max}/2$ и $y_{\max}/2$ у средних делений осей x и y .

Слово

CS2 x_{\max} y_{\max} — — —

строит координатную систему с двумя квадрантами (первым и четвертым) и выполняет оцифровку координатных осей, проставляя значения y_{\max} у верхнего конца оси y , $-y_{\max}$ у нижнего, $x_{\max}/2$ у среднего деления оси x и x_{\max} у крайнего (справа) деления оси.

Слово

CS4 x_{\max} y_{\max} — — —

строит координатную систему с четырьмя квадратами и центром, соответствующим центру координатной системы лого-графики, выполняет оцифровку координат, проставляя $-x_{\max}$ и x_{\max} у концевых делений оси x и $-y_{\max}$ и y_{\max} у концевых делений оси y .

Слово

LCIRCL x y r — — —

строит окружность радиуса r с координатами центра

$$x_0 = x + 127 \text{ и } y_0 = y + 87$$

(т. е. в координатной системе лого-графики).

На базе лого-графики окружность можно создать построением правильного многоугольника с достаточно большим числом сторон n (обычно $n \geq 36$). Однако в этом случае построение происходит в несколько раз медленнее, чем при использовании слова CIRCLE, входящего в состав словарной статьи слова LCIRCL.

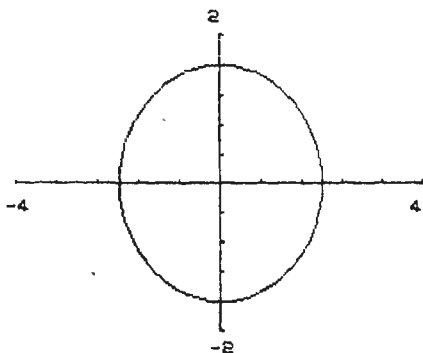


Рис. 9.3. Иллюстрация к совместному применению слов CS4 и LCIRCL

На рис. 9.3 представлена картинка, полученная исполнением следующих команд

```
CS 4 2 CS 4 0 0 60 LCIRCL:
```

В результате их действия экран очищается (CS), строится система координат (4 2 CS 4) и окружность (0 0 60 LCIRCL). В данном случае окружность выглядит как эллипс, поскольку рис. 9.3 получен графическим принтером, несколько вытягивающим изображение по вертикали.

Отметим, что слово LCIRCL строит окружность независимо от значений M_x и M_y . В то же время, построение окружности заменой ее многоугольником (т. е. средствами лого-графики) позволяет менять масштабы по осям M_x и M_y , легко превращая окружности в эллипсы.

Следует помнить, что изменение масштабов в данном случае возможно только по координатным осям — вертикальной и горизонтальной.

Еще более сложный случай — построение эллипса с наклонной главной осью реализует слово ELLIPSE, словарная статья его и результат исполнения слова даны ниже:

```

Ad=37096      Ar=59174
: ELLIPS C90  SETH C2 C0 %I 45 C0
%I C1 FD C2 RT %J 45 C0 %I C2 F
D C2 RT %J %J ;
ELLIPS ;

```



Весьма удобно вводить в состав графики слова, обеспечивающие возможность построения графиков различных функций на фоне координатных осей. В строках 13—15 листа 9.3 представлена реализация такой возможности для функций, графики которых расположены в первом квадранте обычной системы координат.

Основным является слово GF1, используемое в конструкции:

```
xM yM M GF1
```

Здесь: x_M — максимальное значение x на отрезке $[0, x_M]$, y_M — максимальное значение y , проставляемое у конца оси y , M — число точек графика. Значения x_M и y_M проставляются у концов осей x и y системы координат CS1. Значение x_M задает, кроме этого, шаг $\Delta x = x_M/M$, с которым меняется x в ходе построения графика. Вначале задается $x=0$, вычисляется $y(0)$ и строится начальная точка графика. Затем x получает приращение Δx , вычисляется $y(x)$, определяются ее координаты $x=240 \cdot I + 7$ и $Y=150 \cdot y(x) + 12$, где I — номер шага (от 1 до N) и с помощью слова DXY прежняя точка соединяется с новой. Далее процесс повторяется и позволяет строить кривую $y(x)$ без разрывов.

Максимальное значение $y(x)$ должно быть равно $y_{\max}=1$. При этом точка графика соответствует ординате y_M . Таким образом, $y(x)$ нужно задавать в нормированном виде, умножая или деля $y(x)$ в конце вычислений на соответствующий коэффициент.

Пусть нужно построить график параболы $y=x^2$ при x в отрезке $[0,10]$. Следовательно, $y_{\max}=10^2=100$ и нужно задать функцию $y(x)=x^2/100$, где 100 — масштабный коэффициент. Как и ранее, определим функцию $y(x)=Y(X)$ и ее начальный адрес (см. листинг ниже):

```

Ad=38424      Ar=59366
: Y(X) 2^ 100 / ;
: Y(X) FA! ;

```

Теперь исполним предложение

```
10 100 20 GF1 ;
```

Результат этого — график функции $y=x^2$ на фоне координатных осей представлен на рис. 9.4. Отметим, что можно строить любые графики $y(x)$, используя произвольные наименования для их функций, например,

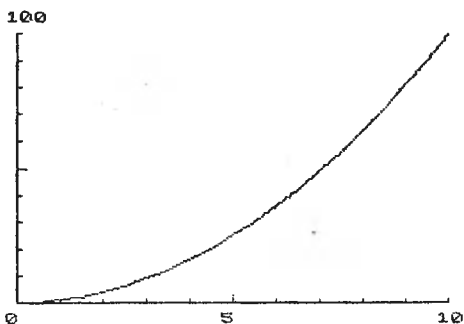


Рис. 9.4. График функции $y=x^2$ на фоне координатных осей

$Y_1(X)$, $F(X)$ и т. д. Нужно лишь установить начальный адрес компиляции соответствующей функции и помнить, что максимальное значение функции должно быть близко к 1. В противном случае масштаб графика будет отличным от принятого — когда $y_{\text{макс}}$ дает ординату точки на графике y_M .

§ 9.6. Рекурсивные графические построения

Рекурсией в программировании называется обращение из какой-либо процедуры к самой себе. Не все языки программирования допускают рекурсию. Хотя рекурсия нередко может заменяться обычными нерекурсивными процедурами, она заслуженно именуется магией программирования, ибо при творческом применении открывает новые интересные возможности в программировании. Особый интерес представляет реализация рекурсивных процедур графики.

Чтобы понять возможность реализации на Форте рекурсивных процедур, нужно ответить на вопрос — допустимо ли указывать имя слова в его собственной словарной статье? В ряде форт-систем (FP50, FSP88, ДССП-80, PC/FORTH) это возможно. Для этого необходимо, чтобы заголовок слова формировался прежде, чем заканчивается создание его словарной статьи.

В системах FP50 и FSP88 последнее происходит автоматически, ибо длина имени фиксирована и заголовок содержит помимо символов имени только указание на адрес компиляции словарной статьи. Он известен к моменту задания нового слова. В связи с этим после ввода конструкции

: Имя ...

в последующей словарной статье можно указывать имя данного слова.

В некоторых форт-системах, например PC/FORTH, заголовок содержит признаки словарной статьи, которые становятся известными лишь после ее полного завершения. В этих случаях рекурсия реализуется благодаря возможности определения слов в ходе их исполнения.

Ограничимся рассмотрением простого примера — построения графика раскручивающейся шестиугольной спирали. На версии Форты FSP88 это реализуется словом SPI :

```
Ad=38416   Ar=59334
: SPI DU FD 60 RT 1.05 * SPI ;
```

Для пуска слова используется команда

$L_{нач}$ SPI ;

Здесь $L_{нач}$ — начальный шаг L перемещения черепашки. После каждого перемещения L умножается на множитель 1,05 и помещается на вершину стека. Затем идет обращение к слову SPI, т. е. к самому себе.

Если бы множитель был равен 1, то процедура строила бы обычный шестиугольник. Умножение L на множитель, больший 1, обеспечивает каждый раз удлинение шага L , на который перемещается черепашка. Поэтому формируется раскручивающаяся спираль (рис. 9.5).

3 SPI ;

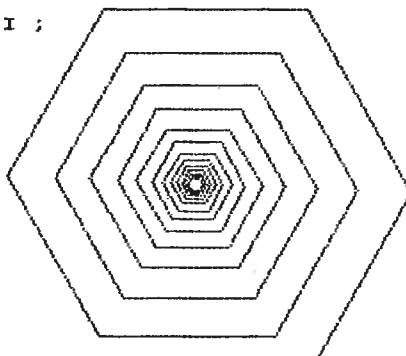


Рис. 9.5. Спираль, полученная с помощью рекурсивной процедуры

Ряд версий Форты прямо не реализует рекурсивное обращение. Однако (см. § 7.10) особыми и довольно простыми приемами рекурсивные процедуры можно реализовать и на них.

§ 9.7. Трехмерная графика в системе FSP88

Построение эстетично воспринимаемых трехмерных фигур — одна из наиболее сложных задач машинной графики. Не пытайтесь исчерпывающе описать даже основы трехмерной графики (для этого есть спе-

циальная литература), остановимся лишь на практически полезном примере реализации трехмерной графики типичными графическими средствами форт-системы FSP88.

Для построения трехмерных объектов используется система координат с тремя направлениями (осями) x , y и z . На плоскости такую систему можно представить различным образом, например, как показано на рис. 9.6. Трехмерная поверхность может быть представлена функцией $z=f(x, y)$, где z , x и y — координаты каждой точки.

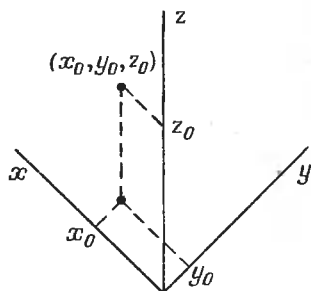


Рис. 9.6. Расположение осей x , y и z в аксонометрической системе координат

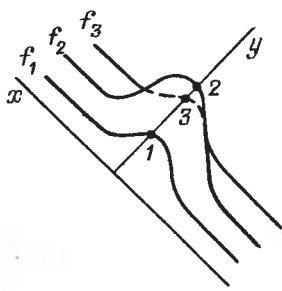


Рис. 9.7. Иллюстрация к алгоритму скрытия невидимых линий при построении трехмерных фигур

Построение всей совокупности точек функции $z=f(x, y)$ бессмысленно, ибо наложение точек друг на друга превратит поверхность на экране дисплея в сплошное пятно. Поэтому для получения эстетически приемлемого восприятия трехмерной поверхности ее расслаивают, строят ряд кривых $z=f(x_i, y)$, где x_i — постоянное для каждой кривой значение. Таким образом, построение трехмерной поверхности сводится к построению ряда кривых $z=f_i(y)$ при $x_i = \text{const}$. Объем вычислений при этом резко увеличивается в сравнении с нужным для построения одной кривой.

Однако и в этом случае эстетическое восприятие поверхности не всегда приемлемо, поскольку она напоминает прозрачный каркас фигуры, построенный из проволочек. Для дальнейшего улучшения восприятия трехмерных поверхностей нужно использовать специальные алгоритмы скрытия невидимых линий.

На рис. 9.7 представлено три линии, иллюстрирующие построение пика. В данном случае построение идет от переднего плана (кривая f_1) к заднему (кривая f_3). В области пика показаны три точки (1, 2 и 3) на каждой из кривых для определенного значения y . Нетрудно заметить, что если соответствующая точка 3 задней кривой лежит выше точки 2 предшествующей кривой (f_2), то она не должна строиться.

Имеется ряд алгоритмов скрытия невидимых линий. Один из наиболее распространенных заключается в построении полных кривых от переднего плана к заднему. После построения кривой массив ее ординат (значений z) запоминается. При построении следующей кривой этот массив поточечно анализируется в соответствии с описанными правилами.

Недостаток этого алгоритма в том, что нужно хранить довольно большой объем значений ординат, обновляя их при переходе от одной кривой к другой.

Мы будем использовать более изящный алгоритм. Его суть заключается в том, что вместо последовательного построения всех точек одной кривой, будем строить по одной точке каждой кривой. К примеру на рис. 9.7 это соответствует построению вначале точки 1, затем 2 и 3. В этом случае достаточно запомнить лишь значение ординаты предшествующей точки, и если она меньше ординаты данной точки, строить последнюю (иначе не строить). При этом алгоритме не нужно хранить массив точек, а сама фигура выявляется более быстро.

Описанный алгоритм реализован словами, представленными в строках 0—5 листа 9.4.

Лист 9.4

Реализация демонстрационной трехмерной графики

```

0 : bb A @ DU ABS + c10 S @ * / INT S @ * c10 * NEG
A @ + ;
1 : xy S @ A @ B @ + * X! S @ B @ A @ - * Y! ;
2 : pl Z@ M @ < IF ELSE Z@ M ! 120 A @ + Z@ S @ * 76
+ PLOT THEN ;
3 : dg 0.7071068 S ! CLS 88 -84 ;
4 : dg1 bb 84 + DU NEG ;
5 : 3DG dg [ -175 M ! I A ! dg1 [ I B ! xy F@ B @ +
Z! pl 7 +] ] ;
6 : ZXY1 X@ 2^ Y@ 2^ + -.001 * EXP 100 * ;
7 : rxy X@ 2^ Y@ 2^ + SQR .001 + R ! ;
8 : ZXY2 rxy R @ S / SIN 180 * R @ / ;
9 : ZXY3 rxy X@ Y@ * X@ Y@ - * X@ Y@ + * 3000 R @ *
/ ;
10 : ZXY4 rxy R @ .08 * SIN 55 * ;
11
12
13
14
15

```

Основным является слово 3DG. Прежде чем его использовать, нужно задать соответствующую функцию:

```
' Имя FA! ;
```

Здесь Имя — имя слова, вычисляющего $z=f(x, y)$. В строках 6—10 заданы слова ZXY1, ZXY2, ZXY3 и ZXY4, вычисляющие следующие

четыре функции:

$$z = 100 \exp(x^2 + y^2),$$

$$z = 180 \sin(R/5)/R,$$

где

$$R = \sqrt{x^2 + y^2} + 0,001,$$

$$z = x \cdot y \cdot (x - y)(x + y) / (3000 \cdot R),$$

$$z = 55 \cdot \sin(0,08 \cdot R).$$

На рис. 9.8 представлены трехмерные фигуры, построенные с помощью слова 3DG при определении четырех приведенных выше функций. Время построения каждой фигуры достигает нескольких минут

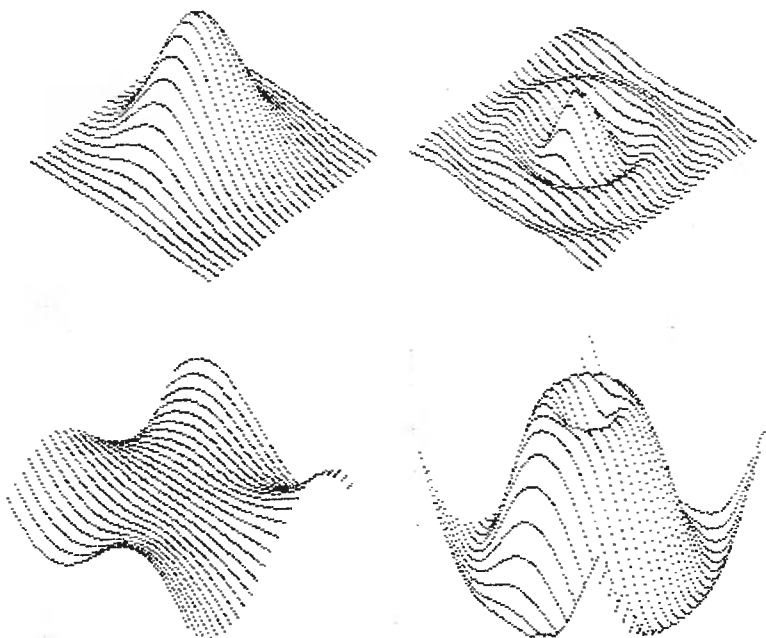


Рис. 9.8. Трехмерные фигуры, полученные с помощью слова 3DG: функция ZXY1 (а), ZXY2 (б), ZXY3 (в) и ZXY4 (г)

в связи с большим числом точек графика и сравнительно медленным вычислением функций \exp , \sin и квадратного корня. В данном случае также актуальна замена вычислений этих функций прямым обращением к ним на вычисления с применением быстрых аппроксимаций и таблиц.

§ 9.8. Быстрая лого-графика для целочисленных версий языка Форт

Описание далее быстрая лого-графика ориентирована на целочисленные версии языка Форт. Полный пакет слов форт-графики представлен ниже на двух листах.

Лист 9.5

Пакет слов 1 лого-графики

7 LIST

SCR # 7

```
0 ( EXTRA-LOGO-GRAPHICS 1 )
1 0 VARIABLE XCOR 0 VARIABLE YCOR
2 0 VARIABLE HD 0 VARIABLE MX
3 0 VARIABLE MY 0 VARIABLE PC
4 0 VARIABLE BG 0 VARIABLE BR
5 0 VARIABLE PS 0 VARIABLE L
6 0 VARIABLE NEWX 0 VARIABLE NEWY
7 : DOT 87 + SWAP 127 + SWAP PLOT ;
8 : PD 0 PS ! ;
9 : PU 1 PS ! ;
10 : SETPC DUP INK PC ! ;
11 : SETBG DUP PAPER BG ! ;
12 : SETBR DUP BORDER BR ! ;
13 : SETH HD ! ;
14 : SETSCR MY ! MX ! ;
15 : SETPOS DUP YCOR ! NEWY ! DUP XCOR ! NEWX !
; -->
ok
```

В этом пакете заданы следующие системные переменные (аналогичны по мнемонике описанным в § 9.2):

XCOR — адрес ячейки ОЗУ, хранящей координату x ,

YCOR — адрес ячейки ОЗУ, хранящей координату y ,

HD — адрес ячейки ОЗУ, хранящей угол поворота φ ,

MX — адрес ячейки ОЗУ, хранящей масштабный коэффициент M_x (в %),

MY — адрес ячейки ОЗУ, хранящей масштабный коэффициент M_y (в %)

PC — адрес ячейки, хранящей код цвета светового пера,

BG — адрес ячейки, хранящей код цвета страницы,

BR — адрес ячейки, хранящей код цвета бордюра,

PS — адрес ячейки, хранящей статус светового пера,

NEWX — адрес ячейки, хранящей новую координату $x - x_{NEW}$.

NEWY — адрес ячейки, хранящей новую координату $y - y_{NEW}$.

Таким образом, системные переменные указывают не на параметры, а на адреса ячеек памяти, хранящих эти параметры. Такой подход типичен для аппарата задания системных переменных версии fig-FORTH.

Кроме указанных системных переменных лист 10.5 содержит ряд определений слов:

DOT $x y$ — — — (построение точки)

Строит точку с координатами x и y в системе координат лого-графнки (точка с координатами (0,0) соответствует центру рабочей области экрана).

PD — — — (перо вниз)

опускает световое перо ($PS=0$).

PU — — — (перо вверх)

поднимает световое перо ($PS=1$).

SETPC n_f — — — (установка цвета пера)

устанавливает цвет пера (n_f — код цвета).

SETBG n_p — — — (установка цвета страницы)

устанавливает цвет страницы (основы), на которой появляются изображения (n_p — код страницы).

SETBR n_b — — — (установка цвета бордюра)

устанавливает цвет бордюра (n_b — код бордюра).

SETH n_h — — — (установка угла HD)

устанавливает целочисленное значение n_h угла поворота HD.

SETSCR $M_x\%$ $M_y\%$ — — — (установка масштабов)

устанавливает соотношение сторон M_x и M_y в процентах (без долей).

SETPOS $x y$ — — — (установка позиции)

устанавливает позицию точки (x, y).

Остальные слова быстрой целочисленной лого-графнки определены в листе 9.6.

Л и с т 9.6

Пакет слов 2 лого-графика

В LIST

SCR # 8

0 (EXTRA-LOGO-GRAPHICS 2)

1 : CS CLS 127 X1 ! 87 Y1 ! 0 HD ! 100 DUP SETSC
R 0 0 SETPOS PD ;

2 : ARG DUP 360 - 0< IF ELSE 360 MOD THEN ;

3 : SINL HD @ DUP 0> IF ARG SIN ELSE MINUS ARG
SIN MINUS THEN ;

4 : COSL HD @ DUP 0> IF ARG COS ELSE MINUS ARG
COS THEN ;

5 : DX L @ MX @ * COSL 10000 */ 100 / ;

```

6 : DY L @ MY @ * SINL 10000 */ 100 / ;
7 : NEWXY DX XCOR @ + NEWX ! DY YCOR @ + NEWY
! ;
8 : XYSWAP NEWX @ XCOR ! NEWY @ YCOR ! ;
9 : POS XCOR @ YCOR @ ;
10 : FDL POS DOT NEWX @ 127 + NEWY @ 87 + DRAW
;
11 : FD L ! NEWXY PS @ IF ELSE FDL THEN XYSWAP
;
12 : BK MINUS FD ;
13 : LT HD +! ; : RT MINUS LT ;
14 : SETX XCOR @ + NEWX ! PS @ IF ELSE FDL THEN
XYSWAP ;
15 : SETY YCOR @ + NEWY ! PS @ IF ELSE FDL THEN
XYSWAP ; -->
ok

```

Этот пакет задает слова:

CS _ _ _ (установка исходного состояния)

обеспечивает очистку экрана, задает $M_x = M_y = 100$ и позицию точки-черепашки в центре экрана.

ARG φ° _ _ _

обеспечивает приведение аргумента $\varphi^\circ > 0$ (φ° задано в градусах) в отрезке $[0, 360^\circ]$. Если φ° больше 360° , то приведение осуществляется выделением (слово MOD) остатка от деления φ° на 360° .

SINL φ° _ _ _ $10000 \cdot \sin(\varphi^\circ)$

вычисляет $10000 \cdot \sin(\varphi^\circ)$ для любых φ° , как положительных, так и отрицательных. Если $\varphi^\circ < 0$, то используется формула $\sin(-\varphi^\circ) = -\sin(\varphi^\circ)$.

COSL φ° _ _ _ $10000 \cdot \cos(\varphi^\circ)$

вычисляет $10000 \cdot \cos(\varphi^\circ)$ для любых φ° , как положительных, так и отрицательных. Если $\varphi^\circ < 0$, то используется формула $\cos(-\varphi) = \cos \varphi$.

Необходимость вычисления $\sin(\varphi^\circ)$ и $\cos(\varphi^\circ)$ в широком диапазоне φ° (как $\varphi^\circ > 0$, так и $\varphi^\circ < 0$) специфична для лого-графики, поскольку при построении сложных фигур вращения значения φ° могут далеко выходить за пределы отрезка $[0, 360^\circ]$, установленные для функций SIN и COS.

DX _ _ _ Δx

обеспечивает вычисление DX по формуле (9.3) с применением аппарата целочисленной арифметики. Функция $(\cos \varphi^\circ) \cdot 10000$ вычисляется с применением таблицы ее значений (см. § 7.8).

DY _ _ _ Δy

обеспечивает целочисленное вычисление DY по формуле (9.4). Для вычисления $(\sin \varphi^\circ) \cdot 10000$ используется таблица значений этой функции.

NEWXY _ _ _ _

вычисляет новые координаты черепашки x_{NEW} и y_{NEW} (целочисленные).

XYSWAP _ _ _ _

обеспечивает замену x на x_{NEW} и y на y_{NEW} .

POS _ _ _ _ x y (оставляет x , y)

выдает текущие координаты x и y .

FDL _ _ _ _

строит отрезок прямой с концевыми точками (x, y) и (x_{NEW}, y_{NEW}) .

FD L _ _ _ _

перемещает черепашку на L шагов вперед с оставлением следа при опущенном пере и без следа при поднятом пере.

BK L _ _ _ _

перемещает черепашку на L шагов назад (остальное, как при действии слова FD).

LT $\Delta\varphi^\circ$ _ _ _ _ (поворот влево на угол $\Delta\varphi^\circ$)

задает поворот черепашки на угол $\Delta\varphi^\circ$ влево, увеличивая на $\Delta\varphi^\circ$ значение системной переменной HD.

RT $\Delta\varphi^\circ$ _ _ _ _

задает поворот черепашки на угол $\Delta\varphi^\circ$ вправо, уменьшая на величину $\Delta\varphi^\circ$ значение системной переменной HD.

SETX L _ _ _ _

перемещает черепашку по оси x на L шагов (влево, если $L < 0$, и вправо, если $L > 0$).

SETY L _ _ _ _

перемещает черепашку по оси y на L шагов (вниз, если $L < 0$, и вверх, если $L > 0$).

Таким образом, функциональные возможности лого-графики целочисленных версий языка Форт полностью аналогичны таковым для версии FSP88, использующей операции над числами с плавающей запятой. Скорость построения графиков в целочисленной реализации в 2—3 раза превосходит таковую для языка Лого. Однако точность

построения графических объектов несколько хуже. Это особенно заметно при малых объектах. Например, форма правильного многоугольника с малым радиусом описанной окружности при построении его целочисленными версиями Лого выглядит заметно отличающейся от идеальной — стороны могут иметь разную длину.

§ 9.9. Демонстрационные примеры целочисленной лого-графики

Рассмотрим несколько характерных примеров применения целочисленной лого-графики. Они задаются словами, определенными на листах 9.7 и 9.8.

Лист 9.7

Построение координатных осей и *N*-угольников

```

9 LIST
SCR # 9
0 ( COORDINATES AXIS )
1 : CORX PU -120 3 SETPOS PD
2 12 0 DO -3 SETY 20 FD 3 SETY LOOP PU ;
3 : COMX 12 0 AT ." -120 " 12 29 AT ." 120 " ;
4 : CORY 3 80 SETPOS PD 8 0 DO -3 SETX -20 SETY
3 SETX LOOP PU ;
5 : COMY 20 12 AT ." -80 " 1 13 AT ." 80 " ;
6 : COORD CORX COMX COMY CORY ;
7 ( FIGURE WITH N ANGLES )
8 : DEMO0 CS COORD
9 0 0 SETPOS PD
10 3 0 DO 40 FD 120 LT LOOP
11 4 0 DO 40 FD 90 LT LOOP
12 5 0 DO 40 FD 72 LT LOOP
13 6 0 DO 40 FD 60 LT LOOP
14 0 0 AT MON ;
15 -->
ok

```

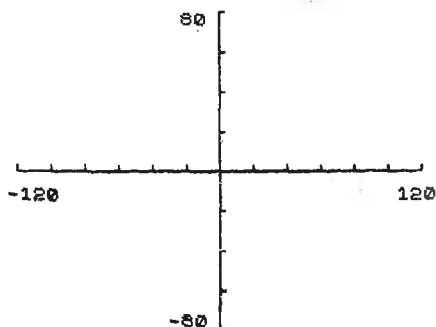


Рис. 9.9. Координатные оси, построенные исполнением слова COORD

В строках 0—6 листа 10.7 задано построение координатных осей, характерных для лого-системы координат. Они строятся масштабными делениями и оцифровкой концевых делений. Слово CORX строит ось

x , слово COMX выводит комментарий (оцифровку оси). Слово CORY строит ось y , слово COMY выводит комментарий этой оси. Основное слово COORD создает при своем исполнении график, показанный на рис. 9.9.

В строках 7—14 определено слово DEMO0, обеспечивающее построение системы координатных осей на фоне очищенного экрана (строка 8), возврат черепашки в исходное положение (строка 9), построение треугольника (строка 10), квадрата (строка 11), пятиугольника (строка 12), шестиугольника (строка 13), вывод курсора в позицию (0,0) и выход из форт-системы (строка 14).

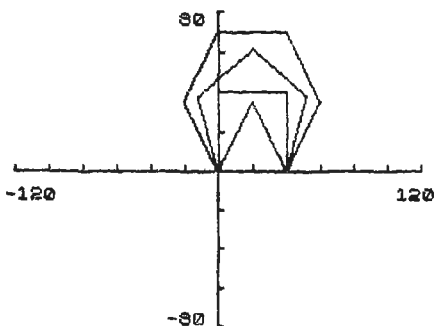


Рис. 9.10. Фигуры на фоне координатных осей, построенные исполнением слова DEMO0

На рис. 9.10 представлено построение, полученное исполнением слова DEMO0. Следует обратить внимание на характерное положение фигур. Все они смещены вправо на величину $L/2$ (в данном случае $L=40$) и находятся сверху от оси x . Это связано с тем, что все фигуры получены перемещением черепашки на L шагов с исходной точки (0,0) и при исходном $\varphi=0$. Для построения всех фигур используются циклы DO — LOOP.

Еще два демонстрационных примера представлено на листе 9.8.
Лист 9.8

Вращение квадрата вокруг угла и построение эллипсов

```
10 LIST
SCR # 10
0 ( ROTATION QADRATE )
1 : DEMO1 CS
2 20 0 DG
3      50 FD 90 LT
4      50 FD 90 LT
5      50 FD 90 LT
6      50 FD 72 LT
7      LOOP
8 MGN ;
9 ( ELLIPS )
```

```

10 : DEMO2 CS
11 -14 -50 SETPOS
12 200 100 SETSCR
13 24 0 D0 14 FD 15 LT LOOP
14 -4 -50 SETPOS 100 200 SETSCR
15 24 0 D0 7 FD 15 LT LOOP MON ;

```

ok

В строках 0—8 этого листа определено слово DEMO1. Оно обеспечивает построение 20 квадратов, вращающихся вокруг угла с координатами (0,0). Квадрат может строиться четырехкратным повторением команд 50 FD 90 LT (вперед на 50 шагов и повернуть влево на 90°). Однако чтобы следующий квадрат был повернут на угол $360/20 = 18^\circ$ вправо, нужно в конце каждого построения задать команду 18 RT. Очевидно, что две следующие друг за другом команды LT 90 и RT 18 можно заменить одной LT 72 ($90^\circ - 18^\circ = 72^\circ$). Именно поэтому построение квадрата задано в строках 3—5 командами 50 FD 90 LT, а в строке 6 командами 50 FD и 72 LT.

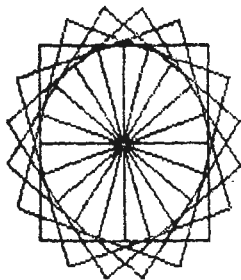


Рис. 9.11. Фигура, построенная исполнением слова DEMO1

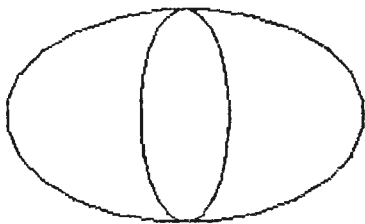


Рис. 9.12. Эллипсы, полученные исполнением слова ELLIPS

На рис. 9.11 показан результат выполнения слова DEMO1. Примененный для построения рис. 9.10 и 9.11 графический принтер, как уже отмечалось, несколько вытягивает изображение по вертикали. Отметим, что это искажение легко учесть, если уменьшить M_y .

Изменение M_x и M_y полезно и при построении эллипсов (см. строки 9—15 листа 9.8). В этих строках задано построение двух эллипсов (в одном случае задано $M_x=200$ и $M_y=100$, а в другом — $M_x=100$ и $M_y=200$). Эллипсы строятся как деформированные по оси x или y многоугольники ($N=24$) — см. рис. 9.12.

Как и в описанной выше версии лого-графки для системы FSP88, средства лого-графики целочисленных версий языка Форт целесообразно совмещать с типовыми графическими средствами их. В ряде случаев это позволяет значительно увеличить скорость построения графических объектов.

§ 9.10. Лого-графика на версии PC/FORTH

Версия Форт PC/FORTH для персональных ЭВМ класса IBM PC имеет дополнительные возможности, облегчающие создание быстрой лого-графики. Достаточно отметить включенные в ее состав слова SIN и COS для организации быстрых целочисленных вычислений синуса и косинуса. Ниже представлен типовой набор средств лого-графики для этой версии, занимающей три листа.

Screen # 40

```
0 ( B/W LOGO-GRAPHICS 1 )
1 : HRGBW HEX 6 MODE B/W DECIMAL
2 VARIABLE XCOR VARIABLE NEWX
3 VARIABLE YCOR VARIABLE NEWY
4 VARIABLE MX VARIABLE MY
5 VARIABLE PS VARIABLE L VARIABLE HD
6 : DOT 100 + SWAP 320 + SWAP 1 !DOT
7 : PD 0 PS ! ;
8 : PU -1 PS ! ;
9 : SETH HD ! ;
10 : SETSCR MY ! MX ! ;
11 : SETPOS DUP YCOR ! NEWY ! DUP XCOR ! NEWX ! ;
12 : ROUND 10 /MOD SWAP 5 / + 10 * ;
13 : CS HRGBW 0 0 SETPOS 250 100 SETSCR 0 SETH PD ;
14 : DX L @ MX @ * HD @ COS 10000 */ 10 / ROUND 10 / ;
15 : DY L @ MY @ * HD @ SIN 10000 */ 10 / ROUND 10 / ; -->
ok
```

Screen # 41

```
0 ( B/W LOGO-GRAPHICS 2 )
1 : NEWXY DX XCOR @ + NEWX ! DY YCOR @ + NEWY ! ;
2 : XYSWAP NEWX @ XCOR ! NEWY @ YCOR ! ;
3 : POS XCOR @ YCOR @ ;
4 : FDL XCOR @ 320 + 100 YCOR @ - NEWX @ 320 +
5 100 NEWY @ - 1 ARCSEG ;
6 : FD L ! NEWXY PS @ IF ELSE FDL THEN XYSWAP ;
7 : BK NEGATE FD ;
8 : LT HD +! ;
9 : RT NEGATE LT ;
10 : SETX MX @ * 100 / XCOR @ + NEWX !
11 PS @ IF ELSE FDL THEN XYSWAP ;
12 : SETY MY @ * 100 / YCOR @ + NEWY !
13 PS @ IF ELSE FDL THEN XYSWAP ;
14 : GLOGO1 362 0 DO 0 0 SETPOS I SETH 80 FD 10 +LOOP ;
15 : GLOGO2 20 0 DO 4 0 DO 50 FD 90 LT LOOP 18 LT LOOP ; -->
ok
```

Screen # 42

```
0 ( B/W LOGO-GRAPHICS 3 )
1 : CIRCLES 36 0 DO DUP 5 RT FD 5 RT LOOP DROP ;
2 : CIRCLEL 36 0 DO 5 LT DUP FD 5 LT LOOP DROP ;
3 : GLOGO3 9 1 DO I CIRCLES I CIRCLEL LOOP ;
4 : GLOGO4 GLOGO3 90 RT GLOGO3 ;
5 : GLOGO5 150 75 SETSCR GLOGO3 ;
6 : COORDX 20 100 620 100 1 LINE 621 20 DO
7 I 101 I 102 1 LINE 60 +LOOP ;
8 : COORDY 320 0 320 199 1 LINE 200 0 DO 317 I 319 I
9 1 LINE 10 +LOOP 317 199 319 199 1 LINE ;
10 : COMMENT 1 13 GOTOXY ." -300" 75 13 GOTOXY ." +300"
11 38 13 GOTOXY ." 0" 35 0 GOTOXY ." +100"
12 35 24 GOTOXY ." -100" 0 0 GOTOXY ;
13 : COORD COORDX COORDY COMMENT ;
14 : GLOGO6 CS COORD 621 20 DO I I 320 - SIN 125 / 100 SWAP -
15 I 10 + I 310 - SIN 125 / 100 SWAP - 1 LINE 10 +LOOP ;
ok
```

Из новых слов, специфичных для лого-графики этой версии отметим слово HRGBW. Оно служит для установки графического режима высокого разрешения (640×200 точек).. При этом формируется монохромное изображение с высокой четкостью.

В пакет лого-графики для версии PC/FORTH включен ряд демонстрационных программ. Слово GLOGO1 (строка 14 листа 41) — ряд отрезков прямой. Один из концов каждого отрезка находится в точке (0,0) системы координат лого-графики. Изменением угла HD от 0 до 360° с шагом 10° в цикле обеспечивается поворот каждого отрезка вокруг исходной точки (см. рис. 9.13).

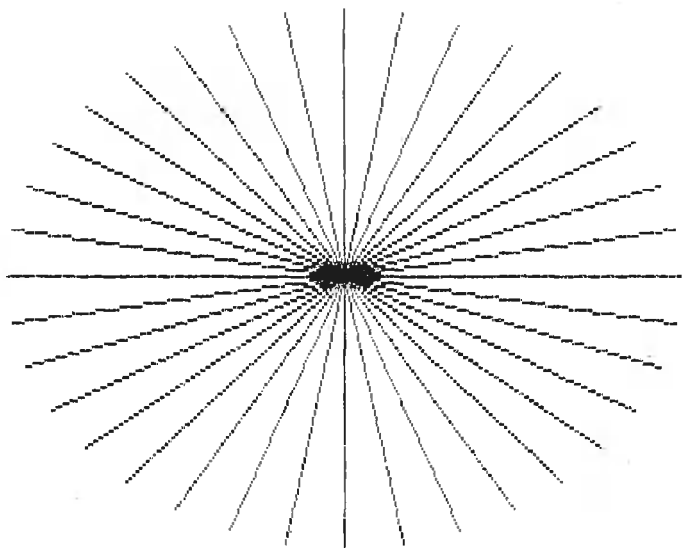


Рис. 9.13. Фигура, полученная вращением отрезка прямой

Слово GLOGO2 строит 10 квадратов, вращающихся вокруг одной из своих вершин. Угол поворота $360^\circ/20=18^\circ$. Рис. 9.14 иллюстрирует построенную фигуру.

Слова CIRCLER и CIRCLEL (строки 1 и 2 листа 42) строят окружность в виде 36-угольников помещением черепашки с поворотом вправо и влево. Параметром этих слов является длина перемещения (она определяет радиус окружности). Слово GLOGO3 строит 9 окружностей разного радиуса обращением к слову CIRCLER и еще 9—обращением к слову CIRCLEL. Слово GLOGO4 дает построения, описанные для слова GLOGO3, затем, развернув черепашку на 90° , повторяет эти построения. Результат этих комбинированных построений показан на рис. 9.15. Слово GLOGO5 иллюстрирует изменение масштабов по осям x и y .

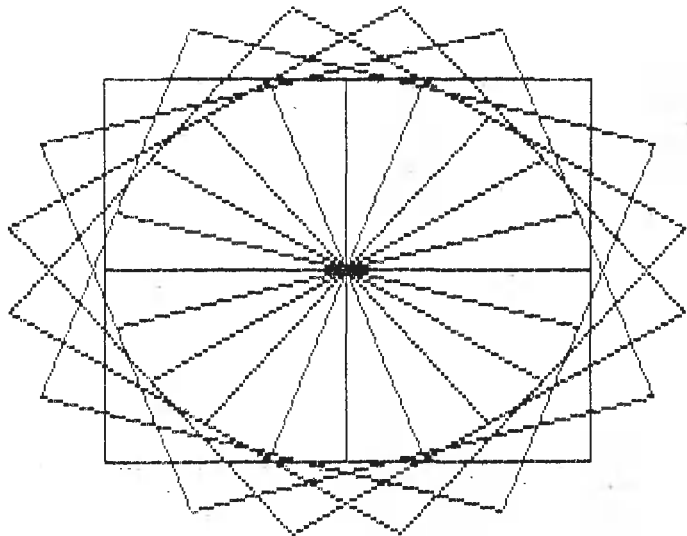


Рис. 9.14. Фигура, полученная вращением 10 квадратов

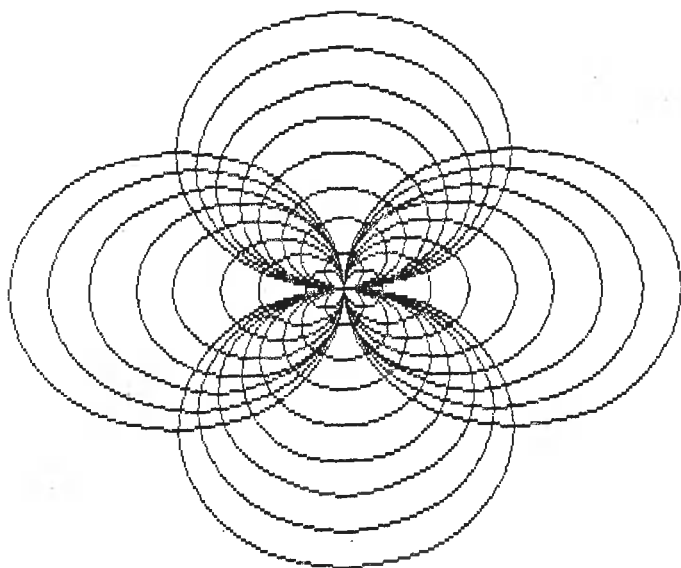


Рис. 9.15. Комбинированная фигура, полученная исполнением слова GLOGO3

В строках 6—13 листа 42 даны слова, реализующие построение координатных осей лого-графики с делениями и оцифровкой их. Наконец, в строках 14 и 15 дана словарная статья слова GLOGO6, обеспечивающего построение графика функции $y = (\sin x) \cdot 100$ на фоне координатных осей лого-графики (см. рис. 9.16).

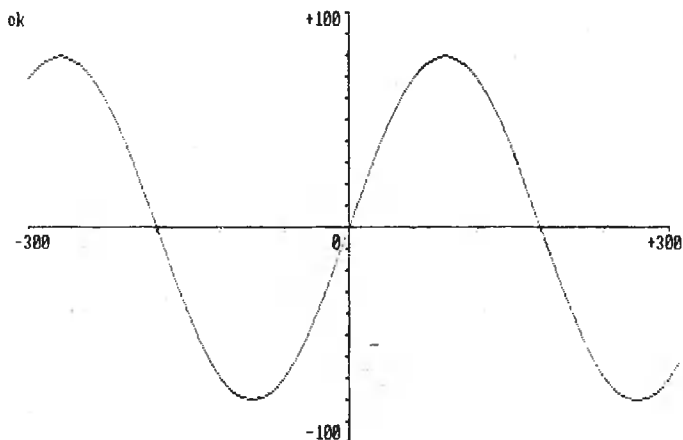


Рис. 9.16. Синусоида на фоне координатных осей, полученная исполнением слова GLOGO6

Приведенные примеры наглядно свидетельствуют о больших возможностях графики форт-систем программирования. Имеются специальные системы форт-графики (например graf-FORTH PC) с повышенной скоростью построения графиков. В демонстрационном пакете graf-FORTH PC изображается быстро летящий двухкрылый самолет, положение которого в пространстве и размеры непрерывно изменяются. Таким образом реализуется наиболее сложная задача машинной графики — изображение реалистического вида трехмерных объектов, быстро передвигающихся в пространстве. Система graf-FORTH PC, однако, относится к специализированным форт-системам и имеет заметные отличия в мнемонике команд в сравнении со стандартами FORTH-79 и FORTH-83.

СВОДНЫЕ ДАННЫЕ О НЕКОТОРЫХ ФОРТ-СИСТЕМАХ ПРОГРАММИРОВАНИЯ ПЕРСОНАЛЬНЫХ ЭВМ

Ниже приведены краткие сводные данные о некоторых форт-системах программирования отечественных и зарубежных ПЭВМ [1—6, 30, 39]. В [1] имеются данные по реализации и иных версий (для ЭВМ серий ЕС, «Эльбрус» и др.).

П1.1. Фиг-Форт для диалоговых вычислительных комплексов ДВК-2М и ДВК-3.

Версия Фиг-Форт для ПЭВМ ДВК-2М и ДВК-3М является вариантом стандартной версии fig-FORTH, разработанной в 1980 г. группой по Форту (FORT Interest Group). На листе П1.1 представлена распечатка (с помощью слова VLIST) полного набора базовых слов данной версии

Лист П1.1

Слова версии Фиг-Форт ПЭВМ ДВК-2М и ДВК-3 (см. стр. 342)

Эта версия используется также для ЭВМ серии СМ с операционными системами ОС РВ и РАФОС. Объем словаря в ОЗУ около 8К байт (свыше 200 слов). Система имеет загружаемый ассемблер и текстовый редактор. Недостаток системы — отсутствие графических возможностей. Для ПЭВМ ДВК-2М, ДВК-3, «Электроника-60» и «Электроника БК-0010» используется также система ФОРТ-СМ на базе стандарта FORTH-83 (словарь 10,5К байт, около 350 слов) [1].

П1.2. ФОС для ПЭВМ «Искра-226»

Форт-операционная система (ФОС) ПЭВМ «Искра-226» построена на основе стандарта FORTH-79 [4]. Базовые слова ее описаны в гл. 6. В расширенной системе около 400 слов, включая слова для реализации операций над числами с плавающей запятой, вычисления элементарных функций, параллельного выполнения операций, адресации к ОЗУ с объемом 128К байт, работы с файлами и организации банков данных. Имеются данные о переводе системы на стандарт FORTH-83 [1].

П1.3. Диалоговая система структурного программирования ДССП-80.

Является отдельной ветвью форт-систем программирования [2, 3]. Мнемоника команд ДССП-80 существенно отличается от принятой в стандартных версиях. Она характеризуется большой краткостью команд и их дальнейшим приближением к требованиям структурного программирования. Реализует принцип программирования сверху вниз. В расширенном варианте имеет встроенный ассемблер, операции над числами с плавающей запятой, слова для вычисления элементарных функций и инфиксный процессор, позволяющий записывать арифметические выражения в инфиксной форме. Успешно используется в диалоговой системе обучения «Наставник» (МГУ [2]). Реализована на ПЭВМ серии ДВК, ведутся работы по ее реализации на других

отечественных и зарубежных ПЭВМ (ZX-Spectrum, MSX, Commodore и др.).

П1.4. Целочисленные версии Форта для ПЭВМ ZX-Spectrum.

Одна из самых массовых и дешевых зарубежных ПЭВМ ZX-Spectrum (и ZX-Spectrum⁺) широко используется в личном пользовании в СССР и в странах Восточной Европы. Имеет обширное программное обеспечение, включая ряд целочисленных версий языка Форт.

Версия Spectrum-FORTH занимает в ОЗУ (48К байт) объем около 8К байт. Экранный редактор (около 3К байт) загружается отдельно. Имеет полный набор графических возможностей, характерных для встроенного в ПЗУ языка Бейсик. Полный перечень слов приводится в распечатке листа П1.2.

Лист П1.2

Слова версии Spectrum-FORTH

```
VLIST
TASK DEF COPY PRINT BORDER
BEEP CIRCLE DRAW PLOT AT PERM
INV BRIGHT FLASH PAPER GOVER
INK NEXT MEM CLS HOME .CPU
LIST .LINE BYE VLIST U. ?
. D. .R D.R #S # SIGN #> <#
SPACES WHILE ELSE IF REPEAT
AGAIN END UNTIL +LOOP LOOP
DO THEN ENDIF BEGIN BACK FORGET
R/W --> LOAD FLUSH BLOCK-WRITE
BLOCK-READ BLOCK EMPTY-BUFFERS
P! P@ MESSAGE (LINE) M/MOD
*/ */MOD MOD / /MOD * M/ M*
MAX MIN DABS ABS D+- +- S->D
COLD WARM ABORT QUIT ( DEFINITIONS
FORTH VOCABULARY IMMEDIATE
INTERPRET ?STACK DLITERAL
LITERAL [COMPILE] CREATE ID.
ERROR (ABORT) -FIND NUMBERC
(NUMBER) WORD PAD HOLD BLANKS
ERASE FILL ? QUERY EXPECT
." (. ") -TRAILING TYPE COUNT
DOES> <BUILDS ;CODE (;CODE)
DECIMAL HEX RSMUDGE SMUDGE
] [ COMPILE ?LOADING ?CSP
?PAIRS ?EXEC ?COMP ?ERROR
!CSP PFA NFA CFA LFA LATEST
TRAVERSE -DUP SPACE ROT >
UK < = - C, , ALLOT HERE 2+
1+ HLD R# CSP FLD DPL BASE
STATE CURRENT CONTEXT OFFSET
SCR OUT IN BLK VOC-LINK DP
FENCE WARNING WIDH TIB R0
S0 +ORIGIN B/SCR B/BUF LIMIT
FIRST C/L BL 3 2 1 0 USER
VARIABLE CONSTANT NOOF ; :
2! C! ! 2@ C@ @ TOGGLE +!
2DUP DUP SWAP DROP OVER DMINUS
MINUS D+ + 0< 0= R R> >R LEAVE
;S RP! RP@ SP! SP@ XOR OR
AND U/ U* CMOVE CR ?TERMINAL
KEY EMIT ENCLOSE (FIND) DIGIT
I (DO) (+LOOP) (LOOP) 0BRANCH
BRANCH EXECUTE LIT OK
```

Версия fig-FORTH фирмы Abersoft [30] для данной ПЭВМ содержит не только полный набор базовых слов стандартной версии fig-FORTH, но и ряд расширений, включая средства цветной графики и задания звуковых сигналов. Имеет встроенный экранный редактор. Занимает в ОЗУ объем памяти около 9К байт. Распечатка всех слов словаря дана ниже.

Лист П 1.3

Слова версии fig-FORTH ПЭВМ ZX-Spectrum.

V. IST

```
C TILL X B F N DELETE FIND
1LINE MATCH -TEXT COPY CLEAR
TOP I P R L T M D S E H
-MOVE #LAG #LEAD #LOCATE
  LDG INIT-DISC INKEY ENDCASE
ENDOF OF CASE DRAW INCY
INCX Y1 X1 PLOT EXIT 2OVER
U.R 2VARIABLE 2CONSTANT
J I' NOT INVERSE GOVER BRIGHT
FLASH INK POINT ATTR PAPER
BLEEP BORDER AT SCREEN OUTF
INF PUSHDE PUSHHL NEXT WHERE
EDITOR TRIAD INDEX FORGET
FREE SIZE 2SWAP 2DROP VERIFY
SAVET LOADT LINE TEXT MON
(TAPE) .CPU CLS LINK LIST
VLIST U. ? . D. .R D.R #S
# SIGN #> <# SPACES WHILE
ELSE IF REPEAT AGAIN END
UNTIL +LOOP LOOPD THEN
ENDIF BEGIN BACK ' --> LOAD
FLUSH R/W HI LO BLOCK BUFFER
DRW EMPTY-BUFFERS UPDATE
+8UF #8UFF PREV USE MESSAGE
.LINE (LINE) M/MOD */ */MOD
MOD / /MOD * M/ M* MAX MIN
DABS ABS D+- +- S->D COLD
WARM ABORT QUIT ( DEFINITIONS
FORTH VOCABULARY IMMEDIATE
INTERPRET ?STACK DLITERAL
LITERAL [COMPILE] CREATE
ID. ERROR (ABORT) -FIND
NUMBER (NUMBER) WORD PAD
HOLD BLANKS ERASE FILL
QUERY EXPECT ." (.) -TRAILING
TYPE COUNT DOES> <BUILDS
;CODE (;CODE) DECIMAL HEX
SMUDGE ] [ COMPILE ?LOADING
?CSP ?PAIRS ?EXEC ?COMP
?ERROR !CSP PFA NFA CFA
LFA LATEST TRAVERSE -DUP
SPACE ROT > UK < = - C,
, ALLOT HERE 2+ 1+ HLD R#
CSP FLD DPL BASE STATE CURRENT
CONTEXT OFFSET SCR OUT IN
BLK VOC-LINK DF FENCE WARNING
WIDTH TIB R0 S0 +ORIGIN
B/SCR B/BUF LIMIT FIRST
C/L BL 3 2 1 0 USER VARIABLE
```



```

CONSTANT NDOF ; : 2! C!
! 2@ C@ @ TOGGLE +! 2DUP
DUP SWAP DROP OVER DMINUS
MINUS D+ + 0< 0= R R> >R
LEAVE ;S RP! RP@ SP! SP@
XOR OR AND U/MOD U* CMOVE
CR ?TERMINAL KEY EMIT ENCLOSE
(FIND) DIGIT I (DO) (+LOOP)
(LOOP) 0BRANCH BRANCH EXECUTE
LIT ok

```

Версия 48/80 FORTH. Построена на основе версии fig-FORTH с рядом изменений. Содержит наглядные команды для реализации цветной графики, отсчета временных интервалов и выполнения ряда системных функций. Имеет встроенный экраный редактор. Занимает в ОЗУ объем памяти около 13,5К байт. Распечатка всех слов словаря дана на листе П1.4.

Лист П1.4

Слова версии 48/80 FORTH (см. стр. 347).

```

VLIST
TASK POINT CIRCLE DRAW LDB
TIME. TIME@ TIME@ FREE BORDER
WHITE YELLOW CYAN GREEN MAGENTA
RED BLUE BLACK TAB AT GOVER
INVERSE BRIGHT FLASH PAPER INK
PLOT BEEP CLS LOADSCREENS
SAVESCREENS LOADTAPE SAVETAPE
HEADER .CPU TRIAD INDEX LIST
BYE VLIST U. ? . D. .R D.R #S #
SIGN #> <# SPACES WHILE ELSE IF
REPEAT AGAIN END UNTIL +LOOP
LOOP DO THEN ENDIF BEGIN BACK
FORGET ' --> LOAD FLUSH R/W
BLOCK BUFFER EMPTY-BUFFERS
UPDATE +PUF #BUFF PREV USE P!
P@ MESSAGE .LINE (LINE) M/MOD
*/ */MOD MOD / /MOD * M/ M* MAX
MIN DABS ABS D+- +- S->D CzLD
WARM ABORT QUIT ( DEFINITIONS
FORTH VOCABULARY IMMEDIATE
INTERPRET ?STACK DLITERAL
LITERAL [COMPILE] CREATE ID.
ERROR (ABORT) -FIND NUMBER
(NUMBER) WORD PAD HOLD BLANKS
ERASE FILL ? QUERY EXPECT ."
(." ) -TRAILING TYPE COUNT DOES>
<BUILDS ;CODE (;CODE) DECIMAL
HEX SMUDGE ] [ COMPILE ?LOADING
?CSP ?PAIRS ?EXEC ?COMP ?ERROR
!CSP PFA NFA CFA LFA LATEST
TRAVERSE -DUP SPACE ROT > UK <
= - C, , ALLT HERE 2+ 1+ HLD
R# CSP FLD DFL BASE STATE
CURRENT CONTEXT OFFSET SCR OUT

```

```
IN BLK VOC-LINK DP FENCE
WARNING WIDTH TIB R0 S0 +ORIGIN
BySCR B/BUF LIMIT FIRST C/L BL
3 2 1 0 USER VARIABLE CONSTANT
NOOP ; : LTAPE STAPE 2! C! ! 2@
C@ @ TOGGLE +! 2DUP DUP SWAP
DROP OVER DMINUS MINUS D+ + 0<
0= R R> >R LEAVE ;S RP! RF@ SF!
SF@ XOR OR AND U/ U* CMOVE CR
?TERMINAL KEY EMIT ENCLOSE
(FIND) DIGIT K J I (DO) (+LOOP)
(LOOP) 0BRANCH BRANCH EXECUTE
LIT OK
```

П1.5. Форт-система программирования микрокалькулятора HP-28C.

Форт-система программирования микрокалькулятора HP-28C создана фирмой Hewlett-Packard (США) [39]. Является примером встроенной системы. Работает в составе операционной системы RPL и объединяет в себе возможности языков Форт и Лисп. Применение данной системы позволило впервые создать программируемый микрокалькулятор для сложных математических и научных расчетов, выполняющий аналитические преобразования и ад математическими формулами, заданными в символьном виде (например, вычисление производной или интеграла от заданной функции). Как и большинство калькуляторов фирмы Hewlett-Packard модель HP-28 использует постфиксную форму записи выражений, имеет малые габариты и массу.

СПИСОК ЛИТЕРАТУРЫ

1. Баранов С. Н., Ноздрунов Н. Р. Язык Форт и его реализация.— Л.: Машиностроение, 1988.—157 с.
2. Баранов С. Н. Стандарты языка Форт.— Л., 1987.—88 с.— (Препр./АН СССР, Ленинградский ин-т информ. и автоматиз., № 4012—В87).
3. Банди Б. Методы оптимизации. Вводный курс: Пер. с англ./Под ред. В. А. Волинского.— М.: Радио и связь, 1988.—128 с.
4. Болдырев А. Ю., Вертенова Н. Н., Лезин Г. В., Силина Е. Ф. ФОС: Интерактивная система программирования для ЭВМ «Искра 226».— Л., 1986.—73 с.— (Препр./АН СССР, Ленинградский научн. центр, Ин-т социально-экономических проблем, № 94926).
5. Брусенцов Н. П. Микрокомпьютеры.— М.: Наука, 1986.—208 с.
6. Бахвалов Н. С., Жидков Н. П., Кобельков Г. М. Численные методы.— М.: Наука, 1987.—600 с.
7. Вирт Н. Алгоритмы + структуры данных = программы: Пер. с англ./Под ред. Д. Б. Подсвилова.— М.: Мир, 1985.—406 с.
8. Вульф А. Операционные системы реального времени в русле развития вычислительной техники.— Электроника, 1985.— № 17.— С. 46—56.
9. Диалоговые микрокомпьютерные системы/Под ред. Н. П. Брусенцова и А. М. Шаумана.— М.: МГУ, 1986.—152 с.
10. Дьяконов В. П. Справочник по алгоритмам и программам на языке Бейсик для персональных ЭВМ.— М.: Наука, 1989.
11. Дьяконов В. П. Справочник по расчетам на микрокалькуляторах.— М.: Наука, 1989.—466 с.
12. Каррен Л. Новое изделие фирмы Harris, соединяющее в себе RISC-архитектуру и язык Форт.— Электроника, 1987.— № 11.— С. 6—7.
13. Семенов Ю. А. Язык 4 поколения FORTH.—1984.— (Препр./ИГЭФ).
14. Современный компьютер: Пер. с англ.— М.: Мир, 1986.—212 с.
15. Справочник по специальным функциям, с формулами, графиками и математическими таблицами.— М.: Наука, 1979.—832 с.
16. Пеайл Я. Р. О свойствах языка Форт: Труды ВЦ.— Тарт. ун-т, 1986.— № 54.— С. 68—84.
17. Фаулджер Р. Программирование встроенных микропроцессоров.— М.: Мир, 1985.—275 с.
18. Фролов Г. Д., Олюнин В. Ю. Практический курс программирования на языке PL-1.— М.: Наука, 1987.—384 с.
19. Хелмс Г. Л. Языки программирования. Краткое руководство: Пер. с англ.— М.: Радио и связь, 1985.—176 с.
20. Anderson, Amta. Mastering FORTH. Brady comm.— Comp., New York, 1984.—216 p.

21. Brodie L. Starting FORTH. Prentice — Hall.— Inc. Englewood Clift, New Jersey, 1987.— 346 p.
22. FORTH. User's Manual. Abersoft, 1983.—28 p.
23. Ham Michael. Factoring in FORTH.— Dr. Dobb's J., 1986.— 11, N 10.— P. 104, 111—113, 120—125.
24. Haudon G. B. All about Forth MVP-FORTH. Series. 1.— Mountain View Press Inc., 1983.—278 p.
25. Henric Coll M. La pratique du FORTH avec HESTOR.— Corbeil — Essonnes, Micromiguel, 1984.—287 p.
26. James J. S. What is FORTH? A tutorial introduction.— Byte, 1980.— 11, N 8.— S. 100—126.
27. Kelly M. G., Spies N. FORTH. A text and reference. Prentice — Hall, 1986.—487 p.
28. Paguent E. FORTH: Another dimension.— Comp. Lang., 1986.—3, N 12.— P. 75, 76, 80.
29. Patton Charles M. Symbolic computation for hand help calculator.— Hewlett Packard J., 1987.—38, N 8.— P. 21.
30. Rather E. D. FORTH takes aim at real time application.— Comput. Des., 1986.— 25, N 19.— P. 85—90.
31. Roberts S. D. FORTH Applications.— Elcomp Publ., Inc. 1985.—185 p.
32. Robert Van Loo. Programmer le forth.— Marabout, 1984.— 288 p.
33. Ron Geere. Forth: The NEXT Step.— Addison-Wesley Publ. Comp., 1986.—89 p.
34. Salman W. R., Tisserand O., Toulout B. FORTH Eurolles.— Paris, 1984.—246 p.
35. Scanlon Leo J. FORTH Programming.— Howard W. Sams. Co. Inc., 1983.—246 p.
36. Thom Hagan. Discover FORTH. Learning and Programming the FORTH Language.— Osborne/Mc Baw Hill, 1982.—146 p.
37. Toppen D. L. FORTH: An applications approach — N. Y. ets: Ms Graw — Hill, 1985.—230 p.
38. Vachha F. A. Spectrum floating point FORTH.— User manual. CP. Software, 1983.
39. Winfield A. J. The complete FORTH.— Sigma Technical Press, 1983.—136 p.
40. Zech R. Programmiers prache FORTH.— Franzics — Verlag Gmb H., München, 1985.—333 s.

ОГЛАВЛЕНИЕ

Предисловие	3
Глава 1. Введение в язык программирования Форт	5
§ 1.1. Общая характеристика языка Форт	5
§ 1.2. Алфавит и слова языка Форт	8
§ 1.3. Определение и переопределение слов	9
§ 1.4. Операции с редактором	11
§ 1.5. Понятие о стеке и текстовом буфере	13
§ 1.6. Постфиксная форма выполнения операций	15
§ 1.7. Режимы работы	17
§ 1.8. Доступ к памяти	18
§ 1.9. Сравнение языка Форт с другими языками программирования	20
Глава 2. Форт-система программирования для научных и инженерных приложений	24
§ 2.1. Общая характеристика форт-систем для научных и инженерных приложений	24
§ 2.2. Слова ОС и редактора	29
§ 2.3. Монитор и декомпиляторы системы FSP88	37
§ 2.4. Слова для ввода-вывода	40
§ 2.5. Слова для управления стеком	45
§ 2.6. Слова для адресации, управления памятью, задания переменных и констант	51
§ 2.7. Арифметические операции	58
§ 2.8. Арифметические функции	60
§ 2.9. Логические операции	62
§ 2.10. Условные выражения	64
§ 2.11. Организация циклов	66
§ 2.12. Слова для управления	73
§ 2.13. Задание слов в машинных кодах и общение с другими языками	75
§ 2.14. Команды графики и синтеза звука	77
Глава 3. Расширение форт-систем с операциями над числами с плавающей точкой	81
§ 3.1. Константы и расширения графики	81
§ 3.2. Системные функции, включая реализацию монитора и декомпиляторов	84
§ 3.3. Табуляция произвольных функций одной переменной ..	88
§ 3.4. Вычисление производной функции одной переменной ...	90
§ 3.5. Вычисление гиперболических и обратных гиперболических функций	91
§ 3.6. Вычисление значений факториала и полинома	92

Глава 4. Операции с векторами и матрицами	95
§ 4.1. Ввод и вывод векторов	95
§ 4.2. Операции с векторами	99
§ 4.3. Ввод и вывод матриц	102
§ 4.4. Элементарные операции с матрицами	106
§ 4.5. Обращение матриц и решение систем линейных уравнений	107
§ 4.6. Одномерная статистика вектора	111

Глава 5. Реализация основных численных методов общего назначения	116
§ 5.1. Вычисление определенного интеграла функции одной переменной	116
§ 5.2. Вычисление корней и экстремумов нелинейных функций	119
§ 5.3. Решение систем обыкновенных дифференциальных уравнений	124
§ 5.4. Спектральный анализ	126
§ 5.5. Операции с комплексными числами	132
§ 5.6. Функции с комплексным аргументом	134
§ 5.7. Линейная и квадратичная интерполяция	136
§ 5.8. Вычисление специальных функций	138
§ 5.9. Полиномиальная аппроксимация	142

Глава 6. Целочисленные версии форт-систем программирования (FORTH-79, fig-FORTH и FORTH-83)	145
§ 6.1. Числа и арифметические операции с ними	145
§ 6.2. Операции со стеком	153
§ 6.3. Операции с памятью, организация констант, переменных и массивов	157
§ 6.4. Операции ввода-вывода и преобразования чисел	165
§ 6.5. Организация условных выражений, логических операций и циклов	172
§ 6.6. Операции редактирования	178
§ 6.7. Задание и сохранение слов, работа с электронным квазидиском	183
§ 6.8. Организация словарей и подсловарей	190
§ 6.9. Управление компиляцией и интерпретацией	192
§ 6.10. Обработка входного потока	197
§ 6.11. Дополнительные данные об организации памяти, системных переменных и слов управления	201
§ 6.12. Графика и звук целочисленных версий языка Форт	211

Глава 7. Программирование на целочисленных версиях языка Форт	216
§ 7.1. Расширение версии fig-FORTH	216
§ 7.2. Организация массивов, таблиц и матриц	218
§ 7.3. Специфика целочисленных вычислений	220
§ 7.4. Целочисленные операции и функции	223
§ 7.5. Сортировка данных	226
§ 7.6. Операции над числами с фиксированной точкой	229
§ 7.7. Интерактивный ввод данных	232
§ 7.8. Быстрое вычисление синуса и косинуса выбором из таблицы	234
§ 7.9. Вычисление других элементарных функций	236

§ 7.10.	Рекурсивные процедуры и программирование сверху вниз	239
§ 7.11.	Слова для управления и контроля	242
§ 7.12.	Форт в качестве словаря и базы данных	244
Глава 8. Версии FORTH для персональных ЭВМ класса IBM PC		
§ 8.1.	Операции с редактором и словарем для версии MVP-FORTH	246
§ 8.2.	Основные операции и типы данных версии MVP-FORTH	246
§ 8.3.	Управляющие структуры и системные переменные версии MVP-FORTH	251
§ 8.4.	Примеры программирования для версии MVP-FORTH	260
§ 8.5.	Общая характеристика версии PC/FORTH	262
§ 8.6.	Операции с редактором и словарем PC/FORTH	265
§ 8.7.	Арифметические и логические операции PC/FORTH	265
§ 8.8.	Операции со стеком PC/FORTH	281
§ 8.9.	Типы данных и общение с памятью в версии PC/FORTH	284
§ 8.10.	Типы данных и общение с памятью в версии PC/FORTH	285
§ 8.11.	Организация ввода и вывода данных в версии PC/FORTH	290
§ 8.12.	Основные управляющие структуры PC/FORTH	292
§ 8.13.	Управление компиляцией и системные переменные PC/FORTH	295
§ 8.14.	Примеры программ в версии PC/FORTH	297
§ 8.15.	Декомпиляторы PC/FORTH и анализатор памяти	300
§ 8.16.	Экранные окна	303
§ 8.16.	Средства цветной графики и синтеза звука PC/FORTH	307
Глава 9. Цветная графика на различных версиях языка Форт ...		
§ 9.1.	Концепции быстрой лого-графики	310
§ 9.2.	Системные переменные лого-графики	310
§ 9.3.	Задание атрибутов и исходного состояния лого-графики	313
§ 9.4.	Операции построения графиков	315
§ 9.5.	Программирование графических операций лого-графики системы FSP88	317
§ 9.6.	Программирование графических операций лого-графики системы FSP88	320
§ 9.7.	Рекурсивные графические построения	325
§ 9.8.	Трехмерная графика в системе FSP88	326
§ 9.9.	Быстрая лого-графика для целочисленных версий языка Форт	330
§ 9.9.	Демонстрационные примеры целочисленной лого-графики	334
§ 9.10.	Лого-графика на версии PC/FORTH	337
Приложение. Сводные данные о некоторых форт-системах программирования персональных ЭВМ		
Сводные данные о некоторых форт-системах программирования персональных ЭВМ		341
Список литературы		347