ОГЛАВЛЕНИЕ

ББК 32.973
С 30
УДК 681.3.06:800.92

Рецензент: канд. физ.-мат. наук С. Н. Баранов

Редакция литературы по информатике и вычислительной технике

Семенов Ю. А.

С 30 Программирование на языке Форт. — М.: Радио и связь, 1991. — 240 с.: ил.

ISBN 5-256-00547-2.

Описан язык Форт, эффективный при решении задач управления, диагностики и отладки аппаратуры в реальном масштабе времени, а также при создании компактных баз данных на мини- и микроЭВМ. Анализируются особенности автономной и многозадачной версии языка Форт. Рассмотрена технология написания трансляторов для Форта. Приведены тексты программы, наиболее употребительные библиотеки.

Для программистов, специалистов в области автоматизации.

$С\frac{2404010000-103}{046(01)-91}143-91$

ББК 32.973

Производственное издание

Семенов Юрий Алексеевич

Программирование на языке Форт

Заведующая редакцией *Г. И. Козырева*
Редактор *Т. М. Толмачева*
Художественный переплет *А. В. Проценко*
Переплет художника *В. Н. Забайрова*
Технический редактор *А. Н. Золотарева*
Корректор *Т. В. Дземидович*

ИБ № 2191

Подписано в печать с оригинал-макета 21. 02. 91. Формат 60×88 1/16. Бумага офсетная № 2.
Гарнитура пресс — роман. Печать офсетная. Усл.печ.л. 14,70. усл.кр.-отт. 15,19.
Уч.изд.л. 13,43. Тираж 50 000 экз. Изд. № 22962. Зак. № 6352. Цена 3 р.

Издательство "Радио и связь". 101000 Москва, Почтамт, а/я 693

Ордена Октябрьской Революции и ордена Трудового Красного Знамени МПО "Первая Образцовая типография" Государственного комитета СССР по печати. 113054, Москва, Валовая, 28

ISBN 5-256-00547-2

Чарльз Мур разработал язык для управления оптическим телескопом и, считая его языком четвертого поколения, назвал FOURTH (четвертый). Однако на ЭВМ, на которой он работал, символьные имена могли иметь только пять букв. Так FOURTH стал FORTH (Форт). Несмотря на конкуренцию других языков программирования, в частности языка Си, Форт мало-помалу стал завоевывать популярность, особенно при решении задач управления сложными объектами в реальном масштабе времени.

Язык Форт использовался для математического обеспечения корабля многоразового использования типа Шаттл, разведывательного 1802 (Avco Inc.) и других искусственных спутников Земли, для разработки телеигр (CameFORTH), при создании мультфильмов Star Wars, Batlle Beyond the Stars и Star Trek, для системы управления полетами в аэропорту Эр-Рияда (400 ЭВМ и 36 000 датчиков) [24]. В 1976 г. Комитет международного астрономического союза принял Форт в качетсве стандартного языка программирования. Позднее Форт применялся для создания экспертных систем, систем искусственного зрения, автоматизации анализа крови и кардиологического контроля, систем машинного перевода с 20 языков (Craig M100 – карманный переводчик) и т. д.

В СССР этот язык используется для систем управления базами данных экономических задач, для программ управления экспериментом, мониторирования состояния пациентов.

Несмотря на ощутимые успехи в использовании языка Форт, делать прогноз о беспредельном расширении сферы его применения вряд ли можно. В то же время, безусловно, существуют области, где Форт имеет несомненные преимущества перед другими языками. Форт эффективен прежде всего для управления небольшими экспериментами и системами, при диагностике сложной электронной аппаратуры с помощью микроЭВМ или микропроцессоров, для создания дешевых поисковых систем, программ машинной графики, трансляторов с других языков. Это, разумеется, не означает, что Форт неприменим в других областях, его возможности еще не раскрыты полностью.

Язык Форт иногда называют системой Форт, так как он содержит программы для работы с внешними устройствами, файлами, средства обработки прерываний, редактор и т. д. Преимущество Форта заключается прежде всего в скорости написания и отладки программ, а также в их компактности. Если программу на Фортране или Паскале можно написать и отладить за неделю, то такую же программу на Форте – за несколько часов. По сравнению с Бейсиком и некоторыми другими интерпретаторами Форт позволяет составить программу в несколько

раз более быстродействующую. Он проигрывает Ассемблеру по скорости исполнения программы не более чем в 1,5 – 2 раза. Применение же Форт-ассемблера позволяет получить еще больший выигрыш в быстродействии.

Экономное использование оперативной памяти ЭВМ и внешней памяти (диска), возможность автономного функционирования (в отсутствие операционной системы), интерпретивный характер делают Форт особенно привлекательным в небольших автоматизированных системах измерения и контроля, хотя известны случаи использования Форта на ЕС ЭВМ для организации сложных вычислительных процессов [36]. Хорошо написанная программа на Форте занимает в памяти меньше места, чем аналогичная, составленная на Ассемблере (здесь не учитывается место, занимаемое процедурами базового словаря Форта).

Модульность интерпретатора и системы в целом позволяет легко адаптировать ее к новым процессорам и задачам. По простоте обучения Форт соперничает с Бейсиком, что делает его привлекательным для непрофессиональных программистов.

Какие же особенности обеспечили Форту шанс выжить в конкурентной борьбе с другими системами?

Для Форта, включая программы управления терминалом и диском, требуется 5 – 8 Кбайт оперативной памяти, а для Паскаля 48 Кбайт (здесь, правда, не учитывается место, занимаемое операционной системой). С учетом этого выигрыш по памяти оказывается существенно больше. При необходимости базовый словарь Форта может быть сокращен до 1 Кбайт. Программа на Форте может быть исполнена сразу после написания, так как не требует редактора связей.

Форт допускает рекурсию, т. е. программа может обращаться к самой себе (что недопустимо в Фортране).

После выполнения программы и возврата управления системе Форт сохраняется доступ к любой переменной или массиву с помощью символьных имен, что не допускают многие другие языки.

При работе с Форт-ассемблером исполняемая программа практически идентична программе, написанной в машинных кодах, но программист избавлен от длительной трансляции и редактирования связей (программа сразу готова к исполнению). Проигрыш по памяти и скорости исполнения программы в этом случае не превышает 20 – 30 %.

Список операторов Форта открыт для пользователя и может быть расширен по его усмотрению. Это касается самого Форта, Форт-ассемблера, редактора и, разумеется, пакетов прикладных программ.

При работе с многозадачным (многопользовательским) Фортом используется общий словарь процедур для нескольких задач.

К недостаткам Форта относят:

отсутствие в базовой версии операторов для работы с числами с плавающей точкой;

недостаточные по современным требованиям средства диагностики ошибок (хотя это и компенсируется отчасти другими возможностями, в частности доступностью этих средств для пользователя);

непривычная для многих обратная польская (постфиксная) нотация, широкое применение стеков;

отсутствие в базовой версии развитой системы для работы с файлами (отчасти устранен в интерпретаторах для персональных ЭВМ).

Простота расширения списка операторов и легкость модификации интерпретатора сводят эти недостатки к минимуму. Удобная мнемоника и хорошая адаптация ко вкусам программиста могут сделать работу на Форте легким и даже приятным занятием. Далее будет рассматриваться в основном версия FIG-FORTH, как наиболее распространенная.

Форт-интерпретатор, как и многие другие, имеет несколько уровней:

работа системы в качестве настольного калькулятора;

программирование с использованием базового словаря Форта;

создание с помощью редактора программ, сохраняемых на магнитном диске, которые могут загружаться по мере необходимости.

Принадлежность Форта четвертому поколению часто вызывает споры. С одной стороны, он содержит структурные операторы типа DO...LOOP, BEGIN...UNTIL и т. д., с другой – допускает работу с адресами, что характерно для языков низкого уровня. Главное – это баланс достоинств и недостатков, а этот баланс, на мой взгляд, благоприятен для Форта.

В США создано общество пользователей Форта (FORTH Interest Group – FIG) и фирма FORTH Inc. – главный поставщик программных продуктов Форт, в том числе многопользовательской версии PolyFORTH-2. Существует ряд стандартов: FORTH-79, FORTH-83 [18], FIG [17], MMS, MVP [25,26]. В какой-то мере этому способствует простота их создания, ведь мало кто решится переделывать или расширять базовый словарь операторов Фортрана просто из-за чудовищной трудоемкости. Наметилась тенденция к созданию процессоров, ориентированных на Форт (фирмы Harris, Silicon Composers, FORTH Inc. США и Институт кибернетики АН ЭССР).

Существуют версии языка Форт для отечественных ЭВМ СМ1420, "Электроника-60", СМ1810, ДВК, персональных ЭВМ ЕС1840, ЕС1841, ЕС1842, микропроцессоров К580 и даже ЭВМ серии ЕС.

Глава 1. Структура языка и режимы работы

Основной конструкцией языка Форт для описания процедур является

: <NAME> <A1 A2 ... An> ;

где <A1 A2 ... An> – список имен слов (операторов), описания которых хранятся в словаре Форта и которые должны быть исполнены при обращении к слову с именем <NAME>. Описание *слова* начинается с ":" и завершается ";". Единственным разрешенным разделителем слов в описании является пробел. В некоторых версиях разделителем может быть <ВК> (Возврат каретки).

Имя может содержать любые символы, кроме пробелов, отделяется от двоеточия и последующего списка имен пробелом. В имени не должно быть более 31 символа, но этот предел (в версии FIG) можно регулировать, изменяя значение системной переменной WIDTH. При загрузке слова с именем, уже имеющимся в словаре, "старое" слово не будет забыто, но доступа к нему в последующих описаниях не будет.

Список имен в описании любого слова Форта может содержать операторы базового словаря или слова, описанные и загруженные пользователем ранее. В новом описании одно имя слова от другого отделяется пробелом. Между последним именем в списке и ";" также должен быть пробел. Например, записано слово

: SQUARE DUP * . ;

Обратиться к нему можно немедленно: 10 SQUARE <ВК>, и на экране появится 100 OK. Здесь и далее <ВК> означает нажатие клавиши "Возврат каретки" или ее заменяющей. Подчеркиванием отмечается отклик на экране терминала, чтобы его можно было отличить от текста, вводимого самим программистом.

При работе с Фортом нужно очень внимательно относиться к знакам препинания (":", ".", ";", "?", ","), так как они также слова базового словаря Форта, а не просто значки, упрощающие чтение текста. В примерах присутствует оператор ".". Он извлекает число из стека, преобразует его с учетом действующей системы счисления в строку символов и выводит на экран.

Одной из важнейших особенностей Форта является постфиксная (обратная польская) нотация – сначала записываются операнды, над которыми должно быть выполнено действие, и только затем знак операции. К примеру, нужно вычислить произведение 12×11. В этом

случае вы должны записать 12 11 * . <ВК>, и на экране будет 132 ОК, где ОК означает, что операция выполнена успешно (с точки зрения ЭВМ). Постфиксная запись -- одна из причин неприятия некоторыми программистами Форта, ведь с начальной школы мы усвоили, что следует писать 2×2 = 4, а не 2 2 *.

Для хранения исходных данных и результатов в Форте используется *стек*. Особенностью такой структуры является последовательная запись и чтение чисел. Причем число, записанное в стек первым, будет считано последним. При этом никакого перемещения информации не происходит, изменяется только содержимое регистра-указателя стека (иногда его функции исполняет специальная ячейка памяти, но это менее удобно). В некоторых ЭВМ арифметические операции выполняются над числами в аппаратном стеке. Операции со стеком являются основным инструментом любого интерпретатора, именно поэтому Форт так удобен для написания трансляторов. Применение стека и постфиксной нотации органически согласуется с аппаратными возможностями современных ЭВМ, что делает Форт весьма эффективным.

Операнды записываются в стек непосредственно или в результате выполнения предшествующих процедур. Оператор вызывает их из стека, а на их место записывается результат работы – число или числа.Схема преобразования стека следующая:

ОП1 ОП2 ... ОПМ – – Р1 Р2 ... РК ,

где ОП1 ОП2 ... ОПМ – М операндов, находящихся в стеке до операции (М обычно фиксировано для каждого оператора, может быть равно нулю); Р1 Р2 ... РК – коды результата, их число К, как правило, постоянно и также может равняться нулю.

Все числа, вводимые с терминала, в конце концов попадают в стек. Так, если набрать 1 2 3 4 5 . . . . . <ВК>, ЭВМ выдаст 5 4 3 2 1 ОК. Здесь пять точек – это не многоточие, а пять операторов отображения содержимого стека на терминале. Каждая точка должна быть отделена от предшествующего и последующего текста пробелами. Количество точек должно быть равно количеству чисел в стеке, которые мы намерены удалить из стека и напечатать. При попытке записать в стек слишком большое количество чисел или извлечь что-то из "пустого" стека возникнет сообщение об ошибке. Перед каждой записью значение указателя стека уменьшается, а после чтения увеличивается на 2. Это связано с тем, что стек в Форте обычно содержит только 16-разрядные числа. Последнее справедливо для микро- и мини-ЭВМ, в общем случае приращение указателя стека может быть иным. Еще один пример:

: ONEMORE 1+ ;

Если теперь набрать 5 ONEMORE . <ВК>, то в ответ получим 6 OK. Вновь определенное слово при обращении к нему добавляет 1 к числу, находящемуся на вершине стека. При этом число в стеке заменяется полученным результатом. В данном случае в стеке вместо 5 окажется 6, это число можно использовать в дальнейших расчетах. Наберем

7 3 * ONEMORE . <ВК>

и немедленно получим на экране 22 OK.

Постфиксная нотация тесно связана с использованием стека, куда записываются как исходные данные, так и результаты. В вышеприведенном примере до выполнения операции в стеке хранились числа 7 и 3 (здесь и далее число, записанное справа, расположено в стеке "выше" числа, записанного слева). После выполнения операции умножения (*) в стеке остается 21, оператор ONEMORE заменяет его на число 22, оператор "." удаляет 22 из стека, возвращая его в состояние, которое он имел до ввода числа 7. При постфиксной нотации возможны операции типа a b c * + . При этом результат будет равен a + (в * с). Как преобразуется содержимое стека при той или иной операции? Воспользуемся схемой a1 a2 a3 → a, где a1, a2 и a3 – числа, находящиеся в стеке до операции, причем a3 на верху стека; справа от стрелки отображается содержимое стека после выполнения операции. В традиционных языках для хранения промежуточных результатов используются специальные рабочие переменные. В Форте это также возможно, но весьма нерационально, для этой цели используется стек.

Базовый словарь Форта оперирует только целыми числами. Причем, если специально не оговорено, то подразумевается, что эти числа одинарной длины (16-разрядные). Допустимый диапазон результатов −32768...+32767. Именно такие числа называются в Форте числами одинарной длины со знаком. Поэтому выполнение операции 1024 128 * дает совершенно бессмысленный результат – произведение больше 32768. Определив слово SECONDS, которое вычисляет число секунд в часах:

: SECONDS 3600 * ;

мы можем рассчитывать на правильный ответ, только если число часов ≤9. Так, 9 SECONDS . <ВК> даст 32400 OK. При попытке воспользоваться этим словом для числа 10 и более получим неверный результат. Для правильного решения подобных задач требуются операции с числами двойной длины (здесь и далее для определенности предполагается, что память ЭВМ 16-разрядная).

Список стандартных арифметических операторов приведен в табл. 1. Во второй колонке показана трансформация стека в процессе исполнения команды: слева от стрелки состояние до операции, справа — после.

Арифметические операции над числами выполняются в соответствии с порядком их записи. Здесь не действуют никакие правила иерархии операций. Так, выполнение операции a b c + * даст результат (b+c)*a.

Таблица 1. Арифметические операторы

Имя	Состояние стека	Версия	Функция
+	n1 n1 → n1+n2	9, 3, F	Сложение
—	n1 n2 → n1—n2	9, 3, F	Вычитание
*	n1 n2 → n1*n2	9, 3, F	Умножение
/	n1 n2 → n1/n2	9, 3, F	Деление
/MOD	u1 u2 → u-rem u-guot	9, 3, F	Деление с остатком
MOD	u1 u2 → u-rem	9, 3, F	Получение остатка
1+	n → n+1	9, 3, F	Прибавление 1
1—	n → n—1	9, 3, F	Вычитание 1
2+	n → n+2	9, 3, F	Прибавление 2
2—	n →n—2	9, 3	Вычитание 2
2*	n → n*2	9, 3, F	Умножение на 2
2/	n → n/2	9, 3, F	Деление на 2
ABS	n →|n|	9, 3, F G	Получение абсолютного значения
MINUS NEGATE	n → —n	F 9, 3	Смена знака числа в стеке
MIN	n1 n2 → n-min	9, 3, F	Выделение минимума
MAX	n1 n2 → n-max	9, 3, F	Выделение максимума
*/	n1 n2 n3 → (n2 * n1)/n3	9, 3, F	Умножение с последующим делением. Результат умножения— 32-разрядное число
*/MOD	u1 u2 u3 → u-rem u-result	9, 3, F	u1*u2/u3 — остаток и частное, u1*u2 — 32-разрядное число
U*	u1 u2 → ud	9, F	Числа u1 и u2 — 16-разрядные без знака, u1*u2 — 32-разрядное число
U/	ud u1 → u2	F	Деление 32-разрядного числа без знака на 16-разрядное

Примечание. 9 и 3 — стандарты Форт-79 и Форт-83; F — FIG-FORTH; G — GraFORTH.

Для ускорения выполнения простых, но часто встречающихся операций в базовый словарь некоторых версий Форта введены однооперандные операторы 1+,1-,2+,2-,2* и 2/. Разумеется, эти функции можно реализовать с помощью еще более простых операторов: 1 +; 1 -; 2 +; 2 -; 2 *; и 2 /. Однако такая реализация в 1,5 - 2 раза медленнее, так как требует сначала записи числа 1 (или 2). Кроме того, операции 2* и 2/ выполняются простым арифметическим сдвигом и никакого реального умножения или деления не производится. К этому ряду процедур примыкают операторы ABS - замена числа в стеке его абсолютным значением и NEGATE (в некоторых версиях MINUS) - замена знака числа в стеке на противоположный.

Большинство арифметических операций достаточно просты и способ их записи очевиден из табл. 1. Некоторого пояснения требуют, может быть, только вычитание ("-") и деление ("/"), для которых небезразлично положение операндов в стеке. Так, 10 2 / . <ВК>дает 5 OK, а 2 10 / . <ВК> дает 0 OK. Во втором случае с учетом целочисленности операций получается нулевой результат. Аналогично 5 3 - . <ВК> дает на экране 2 OK, а 3 5 - . <ВК> дает -2 OK. К этому же типу операторов относятся /MOD и MOD. Первый выполняет деление целых положительных чисел одинарной длины с остатком, например

10 3 /MOD . . <ВК> 3 1 OK.

Обратите внимание - частное наверху стека. У некоторых читателей может возникнуть недоумение, ведь ранее говорилось, что "верхнее" число в стеке записывается слева. На самом деле никакого противоречия нет, 3 действительно находится в стеке над 1, просто здесь показано то, что выдаст ЭВМ на экран. А первым на экране отображается "верхнее" число из стека. Оператор MOD определяет остаток от деления целых положительных чисел одинарной длины, например 10 3 MOD . <ВК> даст 1 OK, а 3 10 MOD даст 3.

Рассмотрим конкретную задачу. В массиве 1024 элемента сгруппированы в строки длиной по 64 элемента. Нужно определить, в какой строке и каком столбце находится элемент 724. Решение: 724 64 /MOD . . . Ответ 11 20 OK означает, что данный элемент расположен в 11-й строке и 20-м столбце. При этом предполагается, что строки и столбцы пронумерованы начиная с 0.

Особый интерес могут представлять операторы */ и */MOD. Уже не раз обращалось внимание на ограничения, связанные с разрядностью чисел (переполнение). Это обстоятельство весьма важно, так как во многих версиях Форта переполнение не вызывает прерывания и программа продолжает работать как ни в чем не бывало, выдавая непред-

сказуемые результаты. Указанные операторы отчасти снимают эти ограничения. Функционально */ и * /, а также */MOD и * /MOD в какой-то мере эквивалентны. Отличие заключается в том, что для хранения промежуточного результата этих операций используется 32 разряда. Например:

4096 128 64 * / . <ВК> 8192 ОК ([4096×128]/64).

Если попытаться выполнить эту операцию, используя * и /, то результат будет неверным из-за переполнения. Можно, разумеется, решить задачу и иначе:

4096 64 / 128 *

Но в общем случае это может дать менее точный ответ, так как результат деления – целое число (остаток отброшен), например

4097 128 64 */ . <ВК> 8194 ОК
4097 64 / 128 * . <ВК> 8192 ОК

Последний результат неточен. Конечно, можно, используя оператор /MOD получить правильный ответ и другим путем, но применение */ проще.

В качестве еще одного простого примера рассмотрим работу с числами с плавающей точкой, применяя только целочисленные операции. Пусть требуется вычислить длину гипотенузы равнобедренного треугольника с длиной стороны 5 см. Как известно, искомая длина равна $5\times\sqrt{2}$ ($\sqrt{2}$ = 1,4142). Требуемые вычисления с достаточной точностью можно проделать, и не прибегая к "плавающей" арифметике:

5 14142 100 */ . <ВК> 707 ОК

что соответствует 7,07. Понятно, что такие "фокусы" лучше оставить на крайний случай. При решении подобных задач правильнее воспользоваться библиотекой процедур для чисел с плавающей точкой.

Остановимся еще на двух операторах MIN и MAX, которые сравнивают два верхних числа в стеке и оставляют там только меньшее или большее соответственно:

3 4 MIN . <ВК> 3 ОК и 3 4 MAX . <ВК> 4 ОК

В первом случае 4, а во втором 3 будут удалены из стека. На самом деле благодаря операции "." в обоих случаях стек возвращается в состояние, которое он имел до ввода чисел 3 и 4.

В Форте скобки не используются, что может создать определенные трудности. Для их преодоления предусмотрен набор операций для манипулирования числами в стеке (см. §1.2). Однако следует заботить-

ся о том, в какой последовательности записываются числа в стек. Например, вычисление (3 + 9) (4 + 6) может быть выполнено без перестановки чисел в стеке:

```
3 9 + 4 6 + * . <ВК>  120 ОК.
```

Если же в стек записаны числа 3 9 4 6, то без перестановок в стеке поставленную задачу уже не решить. Однако не всегда удается "выложить" числа в стек так, чтобы можно было произвести над ними нужные арифметические действия, не меняя их порядка в стеке. Когда столкнетесь с такой ситуацией, вспомните об операциях со стеком.

Арифметические операции с числами двойной длины и с плавающей точкой будут рассмотрены позже (гл. 3 ч. II).

1.2. ОПЕРАЦИИ СО СТЕКОМ

Вернемся к задаче вычисления (a–b)c, когда в исходном состоянии стек содержит a b c. Чтобы получить требуемый результат, надо иметь в стеке c a b. Но как привести стек к нужному состоянию? Загляните в табл. 2. Воспользуемся командой SWAP, которая меняет местами два верхних числа, и получим в стеке a c b. Если теперь применить команду ROT (rotate – вращать), в стеке будет c b a. Повторное применение команды SWAP даст искомый результат c a b. Таким образом, полный текст программы будет иметь вид: SWAP ROT SWAP – *. Последовательность SWAP SWAP оставляет стек неизменным, таким же свойством обладают команды DUP DROP, OVER DROP, ROT ROT ROT и некоторые другие. Заметим, что один и тот же результат можно получить различными способами. Например, указанную выше задачу можно решить с помощью ROT ROT – *, что, конечно, лучше, или >R – R> *, но примененные в последнем варианте операторы будут описаны

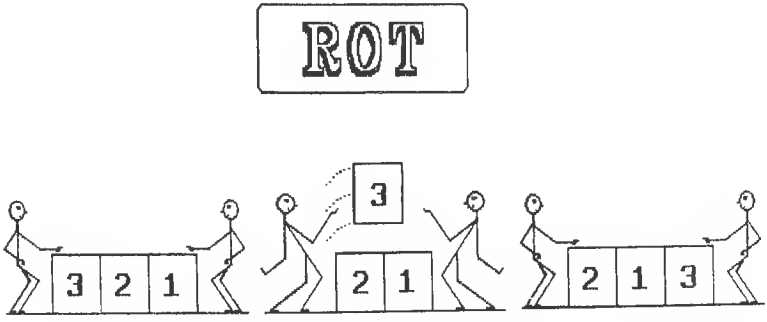


Таблица 2. Операции над кодами в стеке параметров

Имя	Состояние стека	Версия	Функция
SWAP	n1 n2 → n2 n1	9, 3, F	Меняет местами два верхних элемента стека
DUP	n → n n	9, 3, F	Дублирует верхний элемент стека
OVER	n1 n2 → n1 n2 n1	9, 3, F	Копирует 2-й элемент стека и заносит его наверх
ROT	n1 n2 n3 → n2 n3 n1	9, 3, F	Переносит 3-й элемент стека наверх
DROP	n → —	9, 3, F	Удаляет верхний элемент стека
SP!	— → —	F	Устанавливает указатель стека в исходное состояние
SP@	— → SP	9, 3, F, M	Записывает в стек значение указателя, которое он имел до исполнения команды SP@
2DROP	d → —	9, 3	Удаляет из стека число двойной длины
2OVER	d1 d2 → d1 d2 d1	9, 3	Копирует 2-ю пару чисел в стеке и заносит ее наверх
2SWAP	d1 d2 → d2 d1	9, 3	Меняет местами верхние две пары чисел в стеке
2DUP	d → d d	9, 3	Дублирует верхнюю пару чисел в стеке
DEPTH	— → n	9, 3	Выдает в стек число кодов одиночной длины, которые хранились в стеке до исполнения команды DEPTH
PICK	n1 → n2	9, 3	Копирует элемент стека с номером n1 и записывает его на верх стека. Отсчет начинается сверху, верхний элемент имеет номер 1
ROLL	n1 → —	9, 3	Удаляет элемент n1 из стека (n1 не считается) и записывает его на верх стека. 2 ROLL эквивалентно SWAP
—ROT	a b c → c a b	M	Операция эквивалентна ROT ROT

Примечание. См. примечание к табл. 1; M — MMSFORTH.

позже (см. табл. 12). Многовариантность открывает широкие возможности для оптимизации.

Выше промелькнуло загадочное слово DUP. Для прояснения его функции рассмотрим пример:

: КУБ DUP DUP * * ;

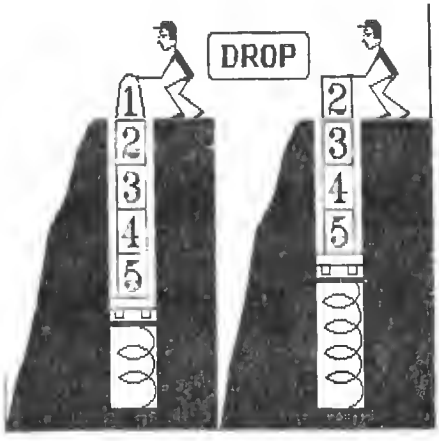
При обращении 4 КУБ . <ВК> получим 64 ОК. В исходном состоянии в стеке 4. DUP дублирует верхнюю ячейку стека. И мы сначала получаем в стеке 4 4, а затем, после повторного использования DUP, 4 4 4. Не будет лишним напомнить об ограничениях применения слова КУБ. Куб числа 64 уже не будет вычислен правильно, так как произойдет переполнение.

Процедура DROP удаляет из стека одно (верхнее) число одинарной длины. Например, в стеке 5 2 5. Нужно вычислить произведение 5×5, а 2 вообще не нужна для дальнейших вычислений. Задача решается путем выполнения последовательности команд

SWAP DROP * . <ВК> 25 ОК

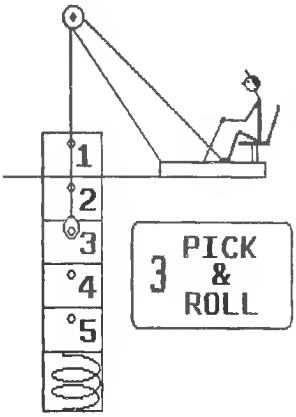
Примером использования DROP и SWAP могут служить описания арифметических операторов / и MOD:

: / /MOD SWAP DROP ;
: MOD /MOD DROP ;



OVER преобразует стек следующим образом: a b c→a b c b. В некоторых версиях Форта имеются операторы PICK и ROLL. Обращение: N PICK и N ROLL. По команде PICK количество чисел в стеке увеличивается на 1. При исполнении ROLL число элементов в стеке не изменяется.

Например, в стеке 3 4 5 6, по команде 3 PICK стек преобразуется к виду 3 4 5 6 4. Если повторить команду, в стеке окажется 3 4 5 6 4 5. По команде 3 ROLL при том же начальном состоянии стека (3 4 5 6) получим 3 5 6 4. Нетрудно заметить, что команды 1 PICK и 2 PICK эквивалентны DUP и OVER соответственно (последние, конечно, выполняются заметно быстрее). Команды же 2 ROLL и 3 ROLL функционально тождественны SWAP и ROT соответственно. По команде 1 ROLL никакой работы не производится. Команда ROLL из числа самых "тихоходных" (перемещается много данных) и поэтому должна применяться осмотрительно. Вообще искусство работы со стеком в Форте – одни из главных критериев, характеризующих программиста.



Вы уже знакомы с оператором DUP, создающим копию верхнего числа в стеке (DUP a → a a). Существует также его двойник –DUP (в некоторых версиях ?DUP), который обычно работает так же, как и DUP, и только в случае a = 0 содержимое стека не изменяется. Например, опишем слово, которое делает пропуск в N строк при выводе данных на экран или печать, присвоим ему имя CR#:

```
: CR# –DUP IF 0 DO CR LOOP THEN ;
```

Оставим без комментариев не описанные пока слова-процедуры, заметим лишь, что то же слово с использованием оператора DUP имело бы вид

```
: CR# DUP IF 0 DO CR LOOP ELSE DROP THEN ;
```

Ведь слово CR# не должно изменять состояния стека.

Не требует какого-либо глубокого проникновения в тайны Форта и понимание операторов, работающих с числами двойной длины: 2SWAP,

2OVER, 2DROP и 2DUP.Их функции идентичны SWAP,OVER, DROP и DUP; только они оперируют парами чисел в стеке. Например, последовательность чисел 1 2 3 4 в стеке (верх стека справа) преобразуется по командам следующим образом:

2SWAP	3 4 1 2
2OVER	1 2 3 4 1 2
2DROP	1 2
2DUP	1 2 3 4 3 4

Принцип работы других команд из табл. 2 легко понять по состоянию стека. Для начинающих работать с Фортом полезно следить за состоянием стека. Для этого можно вопользоваться оператором "." (вывести). Существуют и другие более эффективные операторы, но с ними познакомимся позднее.

В случае необходимости вернуть стек в исходное состояние (это делается путем присвоения указателю стека начального значения) достаточно выполнить команду SP! (записать значение указателя стека). Эта же команда используется и для инициализации словаря и некоторых других процедур. Другой командой, имеющей отношение к указателю стека, является SP@ (загрузить в стек указатель стека). По этой команде в стек записывается значение указателя, которое было до выполнения данной команды. Команда применяется также для контроля состояния стека в ходе выполнения трансляции и при некоторых других процедурах. Пожалуй, самым простым способом восстановления исходного состояния стека при работе с клавиатурой является выдача не существующей в словаре команды, например WWWW. ЭВМ удивится и откликнется WWWW? MSG# O OK, попутно очистит стек. Этот прием годится только при работе в пультовом режиме. В результате данной операции стек преобразуется следующим образом:

```
777 <-- SP (указатель стека)        777
  1                                   1
 12                                  12
128                                 128
--- "дно стека"                     --- <-- SP
```

Дальнейшее освоение Форта полезнее вести за терминалом ЭВМ. Для этого на одном из дисков должны быть записаны файлы FORTH. SAV(RT-11) или FORTH.EXE (для персональной ЭВМ IBM PC), а кроме того, потребуется файл FORTH.DAT или какой-то другой с системной библиотекой, где вы будете хранить свои программы. Этот диск должен быть объявлен устройством по умолчанию (DK: для RT-11).

Для загрузки Форта напечатайте команду RU FORTH (или RUN FORTH), если вы работаете с операционной системой RT-11, или просто FORTH для MS-DOS, если же вы владеете автономной версией, Форт будет загружен при включении ЭВМ. На экране появится FORTH IS HERE или FORTH-PC IS HERE или нечто иное в этом же духе. Теперь вы хозяин (хозяйка) машины и можете приступать к работе.

Описание любого слова содержит список имен других слов. Это могут быть слова базового словаря, или загруженные ранее из библиотеки, или процедуры, описанные самим программистом. Например, описания

```
: EMPTY ; ( пустое описание)
: A 65 EMIT ;                          : B 66 EMIT ;
: NUMBERS 10 0 DO '60 I + EMIT SPACE LOOP ;
```

включают только слова базового словаря Форта (апостроф перед числом указывает, что число представлено в восьмеричной системе счисления, что впрочем, справедливо лишь для FIG-FORTH). Описание

```
: COMPARE DUP 65 = IF A THEN 66 = IF B THEN ;
```

печатает на экране букву A, если в стеке число 65, букву B, если 66, и ничего во всех остальных случаях. Важно то, что слово COMPARE использует слова A и B, которых нет в базовом словаре, но которые были описаны и загружены ранее. Если это правило не выполнено, интерпретация слова COMPARE будет прервана. Некоторые относят это свойство Форта к его недостаткам [32], на мой взгляд, этот тип регламентации лишь дисциплинирует программиста и упрощает поиск ошибок в тексте. Более сложный случай представляет собой пример косвенной рекурсии:

```
: AA BB 2+ ;
: BB AA 4+ ;
```

Здесь как ни меняй местами слова, сообщение об ошибке неизбежно. (Желающих найти выход из этого тупика отсылаю к § 4.4, где рассмотрены рекурсивные процедуры. На практике такие ситуации встречаются крайне редко.)

1.3. РЕЖИМЫ РАБОТЫ ФОРТ

Как и многие интерпретаторы, Форт может имитировать в той или иной мере работу на ручном *калькуляторе:*

```
6 6 * . <ВК>  36 OK
```

Если читатель располагает ЭВМ, он без труда сможет проверить оставшуюся часть таблицы умножения самостоятельно. Возможны и более сложные операции:

```
1 2 + 3 + 4 * . <ВК>  24 OK
```

Нужно только помнить, что базовый словарь Форта содержит только целочисленные операции. Не удивляйтесь, если в результате операции `3 5 / 10 * . <ВК>` вы получите `0 OK`. Правильный ответ получится, если операцию переписать иначе: `3 10 * 5 / . <ВК>  6 OK`. Разные варианты решения в некоторых случаях дадут различные ответы (для целочисленных операций умножение и деление некоммутативны). При этом не нужно забывать и об ограничениях, связанных с разрядностью чисел.

Следующий уровень Форта предполагает описание процедур и их исполнение. Допустим, вы хотите объективно оценить достоинства двух ораторов и оперативно подсчитать, сколько раз за время выступления они скажут "так сказать" или какое-либо другое слово или выражение. Тогда напишите программу

```
0 VARIABLE AA
0 VARIABLE BB
: A 1 AA +! ;        : B 1 BB +! ;
: ITOG AA @ ." A=" . BB @ ." B=" . ;
```

Теперь всякий раз, когда ораторы А и В произносят выбранное слово или выражение, вы даете команду `A <ВК>` или `B <ВК>`. По завершении соревнования напечатайте команду `ITOG <ВК>` и ЭВМ откликнется, например, `A = 135 B = 144`, если результат оратора А равен 135 и 144 у оратора В. Эта программа несовершенна: требуется нажатие клавиши <ВК>, что снижает быстродействие и может привести к просчетам. Но главная цель программы — показать возможности Форта в *пультовом режиме*. Основная особенность программ, написанных в этом режиме, то, что они будут безвозвратно утрачены при выключении ЭВМ (ведь они хранятся в оперативной памяти).

Чтобы запечатлеть свое имя в веках, следует хранить программы в более надежном месте, например на магнитном диске или ленте. Для этой цели можно использовать программу-редактор. Так, с помощью редактора запишем на экране 70 описание оператора FF (см. гл.3):

```
: FF ." FFFFF  OOO  RRR  TTTTT H   H"
  CR ." F     O   O R  R   T   H   H"
  CR ." FFF   O   O RRR    T   HHH H"
  CR ." F     O   O R R    T   H   H"
  CR ." F      OOO  R  R   T   H   H" ; ( см. гл. 3)
```

Чтобы им воспользоваться, нужно загрузить этот экран с помощью команды 70 LOAD, которая обеспечит интерпретацию и загрузку в словарь этого описания. Теперь выдайте команду FF <ВК>, и на экране большими немного неуклюжими буквами будет написано слово FORTH.

Упражнение 1. Сдвинуть код, лежащий в стеке на 9 разрядов влево.

Решения. а) 2* 2* 2* 2* 2* 2* 2* 2* 2*

б) 512 *

в) '177 AND SWAB 2* (если описана операция смены местами байтов числа в стеке SWAB)

Самое быстродействующее решение в).

Упражнение 2. В стеке числа 6 7 8 9 10. Что будет выдано на экран в результате выполнения команды 3 PICK . . . . . <ВК>?

Ответ. 8 10 9 8 7 OK

То же, но при команде 3 ROLL . . . . . <ВК>

Ответ. 8 10 9 7 6 OK

Упражнение 3. Пусть в стеке коды A B C D E F. Вычислить выражение {(F − A)C/(B − − D)}+E.

Решения. а) SWAP >R >R ROT SWAP − R> SWAP >R ROT − * R> / R> +

б) 6 ROLL − 4 ROLL * 4 ROLL 4 ROLL − / +

в) SWAP >R >R ROT SWAP − / R> ROT − * R> +

Самое медленное решение б) (много команд ROLL).

Упражнение 4. В стеке коды A B C D. Вычислить (B + D)C + B + A.

Решение. 3 PICK + * + +

Упражнение 5. Исходные данные те же, что и в упражнении 3, но вычислить (A + + B)C − D + C.

Решения. а) MINUS OVER + SWAP ROT 4 ROLL + * +

б) OVER − SWAP ROT 4 ROLL + * SWAP −

Возможны и другие варианты решения.

Глава 2. Операции с памятью

2.1. ОПИСАНИЕ КОНСТАНТ И ПЕРЕМЕННЫХ

В Форте обычно числа хранятся в стеке, но иногда возникает потребность иметь константы или переменные, что традиционно для других языков. Для этого в Форте имеются два слова VARIABLE (переменная) и CONSTANT (константа) (см. табл.23). Обращение к ним: m VARIABLE <NAME> и n CONSTANT <name>, где m и n − целочисленные начальные значения переменной и константы (начальное значение переменной при ее описании задается не во всех версиях

Форта). Введем переменную с именем REGOUT, указывающую адрес модуля КАМАК [42]:

```
171 VARIABLE REGOUT
```

Теперь в словаре появилось слово с именем REGOUT, и при обращении к нему в стек будет записан адрес числа 171, если, конечно, до этого обращения значение переменной не было изменено. В некоторых версиях Форта (например, в Форт-83) начальное значение переменной (в данном случае 171) не вводится.

В Форте можно работать с символьными и абсолютными адресами. Примером символьного адреса может служить REGOUT. Если выполнить команду REGOUT @ . <ВК> (оператор @ замещает адрес в стеке его содержимым), то ЭВМ выдаст 171 ОК. Команда типа OCTAL 1000 @ . <ВК> 132 ОК представляет собой случай обращения по абсолютному адресу. Опишем константу BASCAM, которая указывает на начало поля адресов КАМАК:

```
OCTAL 164000 CONSTANT BASCAM
```

При обращении к BASCAM в стек будет записано восьмеричное число 164000. Теперь напишем программу вычисления адреса КАМАК по известному субадресу и номеру станции:

```
: CAMADR 32 * SWAP 2 * + BASCAM + ;
```

Предполагается, что КАМАК-адрес равен BASCAM + 32 * NST + 2 * SA. Здесь NST – номер станции в каркасе КАМАК, где размещен блок, адрес которого вычисляется; SA – субадрес объекта внутри этого блока. Такой способ адресации пригоден только для ЭВМ, где для внешних устройств выделено определенное поле адресов. При работе с персональными ЭВМ эта схема адресации неприемлема. Примером обращения к CAMADR может служить команда вычисления КАМАК- адреса модуля, размещенного в станции 10 и имеющего субадрес 1:

```
1 10 CAMADR
```

Результат вычисления будет помещен в стек. Присвоим этот адрес переменной REGOUT. Для этого достаточно выдать команды REGOUT !. Оператор ! запишет число, которое находится в стеке по адресу, также расположенному в стеке. В нашем случае адрес в стек записал оператор REGOUT. Если мы хотим посмотреть, чему равно значение переменной REGOUT, надо выдать команду REGOUT @ . <ВК> или REGOUT ? <ВК>. Из этих записей видно, что оператор ? эквивалентен @ . и может быть описан как ? @ .;. По команде "?"считывается содержимое ячейки, адрес которой находится в стеке, и печатается на

экране ее значение. Таким образом, до выполнения команды в стеке хранится адрес, после – ничего (в таблицах "ничего" обозначается прочерком).

Другой пример. Для управления некоторым внешним по отношению к КАМАК устройством надо записать в выходной регистр станции 2 с субадресом 0 восьмеричный код 15. Решением может служить следующая программа:

```
OCTAL
0 2 CAMADR  REGOUT !   15 REGOUT @ !
```

Может возникнуть вопрос: почему в первом случае мы пишем REGOUT !, а во втором – REGOUT @ ! ? (большие пробелы перед знаками препинания местами вводятся для того, чтобы их нельзя было спутать с операторами Форта ".", "?", "!" или ","). В первом случае переменной REGOUT присваивается значение КАМАК-адреса, во втором – производится запись по этому адресу. Если запись производится многократно, команда 0 2 CAMADR REGOUT ! должна быть выполнена только раз при запуске управляющей программы.

Иногда абсолютная адресация используется при работе с внешними устройствами или при обращении к системным переменным типа JSW в RT-11 (0 44 !). При работе с абсолютными адресами следует проявлять определенную осторожность: в некоторых версиях Форта команда 40 1001 ! вызовет прерывание (обращение по нечетному адресу).

Операторы C@ (извлечь байт) и C! (записать байт) – аналоги операторов @ и ! (табл. 3), но выполняются для байтов, а не для 16-разрядных чисел, т. е. извлекаются из ячейки и записываются в нее не числа, а байты.

Таблица 3. Операторы для работы с памятью

Имя	Состояние стека	Версия	Функция
@	адр → n	9, 3, F	Замещает адрес в стеке его содержимым
C@	адр → –	9, 3, F	Извлекает байт информации из ячейки, адрес которой находится в стеке
!	n адр → –	9, 3, F	Записывает число n в ячейку с адресом "адр"
C!	b адр → –	9, 3, F	Записывает байт b по адресу, указанному в стеке.

Окончание табл. 3

Имя	Состояние стека	Версия	Функция
+!	n адр → –	9, 3, F	Добавляет к содержимому ячейки с адресом "адр" число n
, (запятая)	n → –	9, 3, F	Компилирует число n в первую свободную ячейку словаря
C,	b → –	9, 3	Компилирует байт b в очередной свободный байт словаря
ALLOT	n → –	9, 3, F	Добавляет n байт к полю параметров слова, описанного последним
FILL	адр n b → –	9, 3, F	Записывает n байт b в память начиная с адреса "адр"
ERASE	адр n → –	9, 3, F	Записывает n 0-байтов в память начиная с адреса "адр".
BLANKS	адр n → –	F	Записывает n кодов "пробел" (32) в память, начиная с адреса "адр"
BLANK		9, 3, M	

Примечание. 9 – стандарт Форт-79; 3 – стандарт Форт-83; F – FIGFORTH; M – MMSFORTH.

Оператор +! (плюс-присвоить) добавляет 16-разрядное число к содержимому ячейки, адрес которой (ADR) хранится в стеке. Формат обращения N ADR +!. До исполнения команды в стеке находятся N и ADR, после ничего (табл. 3).

Приращение содержимого ячейки может быть как положительным, так и отрицательным, например по команде –5 ADR +! к значению переменной ADR добавится –5. Хотя оператор +! относится к числу примитивов базового словаря, тем не менее даже его можно описать, используя уже известные слова:

```
: +!                                ( в стеке   n ADR          )
    DUP                             (           n ADR ADR      )
    @       ( запись в стек содержимого адреса  n ADR (ADR)    )
    ROT                             (           ADR (ADR) n    )
    +                               ( приращение ADR (ADR)+n   )
    SWAP                            (           (ADR)+n ADR    )
    ! ;    ( запись результата по адресу, хранящемуся стеке)
```

где n – константа, которая добавляется к содержимому ячейки по адресу ADR. ADR может быть записан в стек при обращении к переменной, например REGOUT.

Следует подчеркнуть различие между переменной и константой. При появлении в списке исполняемых слов имени переменной в стек записывается ее адрес, а в случае константы – значение. Существует

несколько способов описания переменных и констант. Это продемонстрировано в таблице на примере числа 8:

Описание	Текст в словаре					Состояние стека
: 8 8 ;	201	270	LINK	CALL	LIT, 8*	8
8 CONSTANT 8	201	270	LINK	CONST	8*	8
8 VARIABLE 8	201	270	LINK	VAR	8*	PFA

* Записано в PFA.

Малопонятные на данном этапе письмена в центральной части таблицы оставьте пока без внимания. Числа во второй и третьей колонках восьмеричные. Все три описания в первой колонке запишут число 8 в одну из ячеек памяти. Но доступ к этой информации и израсходованные ресурсы (память и время) будут разными. Первый способ самый неэкономный как по памяти, так и по скорости извлечения нужного значения в стек. Самый экономный – второй (число 8 описано как константа). В случае переменной (третий вариант) при обращении к 8 в стек будет записан адрес 8, а не само значение. Для получения требуемого значения в стеке следует написать и исполнить 8 @. Первое описание может показаться несколько сюрреалистическим, но парадокс заключается в том, что именно так чаще всего описываются константы в программе.

Пусть слово INCH преобразует длину в дюймах (число дюймов предполагается целым) в длину в миллиметрах и выдает результат на экран:

```
: INCH 254 10 */ . ;
```

Обращение: 10 INCH даст 254 OK. Здесь числа записаны в стек и извлекаются по первому способу. Именно это послужило причиной того, что числа 1, 2, 3 (иногда и некоторые другие) описываются как константы в базовом словаре. Таким образом, если в вашей программе какое-либо число встречается часто, а вам небезразлична скорость исполнения программы, опишите это число как константу. Например, 1024 CONSTANT 1K или 1024 CONSTANT 1024. В последнем случае имя константы 1024.

Среди переменных, определенных в системе Форт, особое место занимает BASE, задающая основание системы счисления. В Форте определены три оператора, управляющие системой счисления и задающие десятичную, шестнадцатеричную и восьмеричную системы: DECIMAL, HEX и OCTAL. Описания этих слов весьма просты (табл. 8):

```
: DECIMAL 10 BASE ! ;
: HEX     16 BASE ! ;
: OCTAL    8 BASE ! ;
```

Переменная BASE используется при вводе и выводе чисел на экран или печать. В процессе работы система счисления может многократно изменяться, поэтому в какой-то момент времени вы можете с трудом интерпретировать результат из-за неуверенности относительно текущей системы счисления. Как же можно оперативно определить основание системы счисления? Первое, что приходит в голову, это BASE @ . Ответ очевиден: 10 OK. Но что означает 10? Ведь такой результат мы получим при любой системе счисления. Правильный ответ может дать процедура:

```
: BAS? BASE @ DUP DECIMAL . BASE ! ;
```

которая отпечатает основание системы счисления в десятичном виде и при этом оставит переменную BASE в том состоянии, которое она имела до проверки. (Читателю, желающему испытать себя, предлагаем написать программу, которая будет поименно называть действующую систему счисления (DECIMAL, OCTAL или HEX). Для этого, правда, нужно познакомиться с главами, где описаны операторы условия и операторы для работы с текстовыми строками.)

Рассмотрим простейший способ описания массивов. Для этого используется оператор VARIABLE в сочетании с ALLOT. Например, по команде 0 VARIABLE AA 126 ALLOT будет описан массив с именем AA длиной 128 байт. Причем первые два байта на стадии описания нулевые, содержимое остальных пока не определено. Оператор N ALLOT резервирует в словаре N байт, N для ЭВМ типа ДВК или СМ должно быть четным, а для персональных ЭВМ ЕС1841 – любым. Возможна и комбинация, когда число байт в массиве вычисляется в процессе выполнения программы:

```
0 VARIABLE LL
.............
16 16 * LL !
```

```
.....
0 VARIABLE BB LL @ ALLOT
```

В данном примере длина массива BB равна 258 (LL = 256), так как команда 0 VARIABLE BB уже зарезервировала два байта для значения BB. Если попытаетесь переопределить длину массива BB с помощью 0VARIABLE BB LL @ ALLOT, в словаре появится новый массив с тем же именем BB, а доступ к старому станет невозможным (во всяком случае через обращение BB). После того как массив описан, с ним можно работать. Например, записать число 5 в ячейку 12 массива BB (нумерация начинается с 0) можно с помощью команды 5 BB 12 2* + ! (использовано умножение на 2, так как запись производится в ячейку 12, а не в байт.)

Для переноса массивов из одной области памяти в другую используются операторы CMOVE, MOVE (перенести) и <CMOVE (в Форт-83 CMOVE>). Первый есть во всех версиях, служит для переноса последовательности байтов, второй – слов, и поэтому он быстрее, а третий начинает перенос с конца массива. Форма обращения для всех трех операторов одна и та же: ADR1 ADR2 N CMOVE, где ADR1 – адрес первого байта массива-источника; ADR2 – адрес первого байта массива-адресата; N – число пересылаемых байтов (или слов в случае MOVE). Все эти три числа должны быть в стеке, после выполнения команды они удаляются из стека. Все три параметра переноса могут быть заданы символьно или абсолютно:

```
0 VARIABLE A 12 ALLOT
0 VARIABLE B 120 ALLOT
.....
A B 12 CMOVE
A B 12 + 12 CMOVE
```

В обоих вариантах перенос байтов происходит из массива A в массив B. Последний перенос записывает массив A, начиная с 12-го байта массива B. В обоих случаях N = 12. Для данных массивов $1 \leqslant N \leqslant 14$. Но если N>14, предсказать, что будет передано в массив B после 14-го байта, весьма трудно. Никаких сообщений вы не получите, последствия могут быть катастрофическими. Возможна команда типа 1500 2500 1000 CMOVE, которая перешлет 1000 байт из области памяти, начинающуюся с адреса 1500, в область, начинающуюся с адреса 2500. Но лучше не злоупотреблять абсолютной адресацией: это слишком опасное упражнение, тем более, что, пользуясь абсолютной адресацией, нужно знать, с какой страницей или сегментом памяти вы имеете дело.

Оператор MOVE перемещает всегда четное число байт, но делает это почти в 2 раза быстрее, чем CMOVE. Оператор <CMOVE служит главным образом для переноса перекрывающихся массивов или для раздвижки массива и освобождения места для нового элемента. При переносе байт между перекрывающимися массивами надо тщательно выбирать тот или иной оператор пересылки (осмотрительность – важная добродетель программиста).

Имеется ряд операторов для заполнения массивов памяти идентичными символами. Универсальным оператором такого типа является FILL. Обращение к нему: ADR N sym FILL, где ADR – адрес первого байта памяти, куда будет засылаться код символа sym; N – число заполняемых байтов, например

```
DECIMAL 0 VARIABLE AAA 128 ALLOT
AAA 130 65 FILL или AAA 130 ASCII A FILL
```

Слово ASCII выводит в стек код ASCII (см. ч.II, гл.7) следующего за оператором символа, но этот оператор имеется не во всех версиях Форта. Описанная процедура заполняет все 130 байт массива AAA кодами символа A.

Здесь уместно сказать о кодировании буквенных символов. Латинские буквы обычно кодируются в соответствии с американским стандартом ASCII, буквы русского алфавита по-разному. Отчасти это связано с многообразием периферийной аппаратуры, отчасти с особенностями операционных систем. В версиях Форта, с которыми пришлось работать автору, использовалось два способа. В первом переключение с латинского на русский алфавит выполнялось кодом ^N, а обратно кодом ^O(дисплей "Электроника ИЭ-15", ОС RT-11, см. приложение 5), во втором (он более распространен) – коды русских букв отличаются от кодов латинских наличием 1 в старшем бите байта (ЭВМ IBM PC, ОС MS- DOS). Этим вариации не исчерпываются, так как существует несколько разных таблиц соответствия латинского и русского алфавитов для клавиатур, многовариантен и вывод на печатающее устройство, но это уже другая тема. Пример таблицы кодов смотри в гл. 5. Важно, что имеется возможность создать командный язык, состоящий только из русских слов.

Для обнуления массивов предусмотрен оператор ERASE (стереть), который является, по существу, частным случаем слова FILL (заполнить). Обращение: ADR N ERASE. Значения параметров те же, что и для FILL. Описание ERASE имеет вид:

```
: ERASE 0 FILL ;
```

Напишем AAA 130 ERASE, и описанный выше массив будет обнулен. При работе с массивами символов (строками) аналогичную функцию выполняет команда ADR N BLANKS (заполнить пробелами), которая выполняет массив байтов кодами пробела (32 в десятичной системе счисления). Описание BLANKS имеет вид:

```
: BLANKS BL FILL;
```

где BL – системная константа, равная 32.

Существует два близких родственника оператора ALLOT – это "," (записать в словарь код) и "C," (записать в словарь байт). Запишем:

```
›40530 VARIABLE HA ›55 C, ›130 C, ›101 C
```

где "C," записывает число в очередной свободный байт словаря. Апостроф перед числом указывает интерпретатору, что его следует считать восьмеричным вне зависимости от действующей системы счисления; "+" или " – " перед числом означает, что оно записано в десятичной системе счисления (FIG). Если теперь выдать команду HA 5 TYPE <ВК>, то получим XA ≠ XA OK.

Функция оператора ",", та же, что и "C,", но для чисел, активно используется в процессе интерпретации (например, при реализации слова COMPILE) для записи последовательности кодов в словарь Форта.

Упражнение 1. Опишите оператор с именем KBDB для вычисления квадратного трехчлена $Ax^2 + Bx + C$, где A, B и C – константы, равные 7, 8 и 1, а x – число в стеке. Результат должен появиться на экране сразу после нажатия клавиши <ВК>.

Решение.

```
7 CONSTANT A   8 CONSTANT B   1 CONSTANT C
: KBDB DUP A * B + * C + . ;
```

Упражнение 2. Напишите программу для расчета определителя второго порядка с элементами a, a+1, a+2 и a+3, a=2.

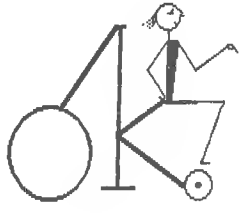
Решение.

```
2 CONSTANT CONST                        ( значение a)
CONST DUP VARIABLE D2 DUP 1+ , DUP 2+ , 3 + ,

D2 – имя массива матрицы

: DET2 D2 @ D2 6 + @ *      ( перемножение элементов основной
                              диагонали)
       D2 2+ @ D2 4 + @ * – ;
```

Эта задача ие из числа практических: результат равен –2 при любых значениях CONST. Оператор, пригодный для произвольного массива, имеет вид

```
: DE2 DUP @ OVER 6 + @ * OVER 2+ @ ROT 4 + @ * – ;
```

При обращении D2 DE2 . <ВК> будет –2 OK. Разумеется, при другом содержимом матрицы D2 результат будет иным.



Глава 3. Ввод-вывод данных

3.1. ОПЕРАТОРЫ ВВОДА-ВЫВОДА

Нажатие клавиши терминала вызывает прерывание (в персональных ЭВМ IBM PC даже два). Коды вводимых символов поступают во входной кольцевой буфер, откуда их забирает оператор KEY (клавиша), входящий в процедуру EXPECT. После некоторой предварительной селекции они заносятся во входной буфер Форта (TIB). Схема обработки входных кодов представлена на рис.1 (подробнее см. гл.7).

С некоторыми операторами ввода-вывода ("." и др.) вы уже познакомились. Мы использовали и операторы "@." или "?" для вывода на экран содержимого ячейки, адрес которой хранится в стеке. Но этим набор операторов ввода-вывода не ограничивается. Для упрощения восприятия выводимых данных желательно сопровождать их теми или иными комментариями и сообщениями. Такая возможность реализуется в рамках структуры ." XXXXXX", где ." – оператор начала сообщения, после чего необходим пробел; XXXXXX – последовательность любых кодов ASCII, включая расширенный набор с кодами более 127 (недопустим только символ " и символы с восьмеричными кодами <40); " – указатель конца сообщения, до него или после него пробелы необязательны. Например, на обращение

```
: DOBROE_UTRO ." Good morning," ;
DOBROE_UTRO ." I'll call you later" <BK>
[в Форт-83 .( I'll call you later) ] ЭВМ выдаст
Good morning, I'll call you later OK
```

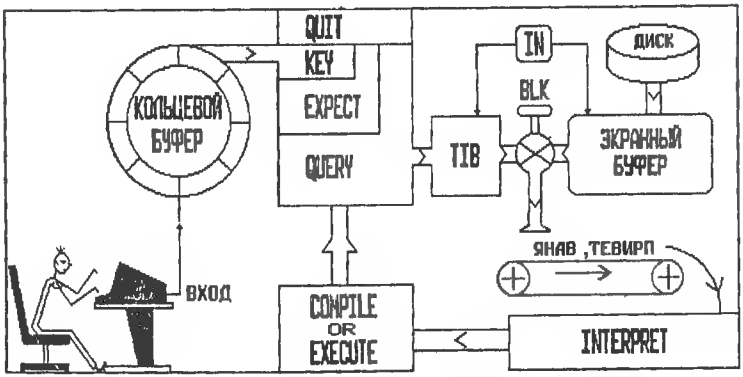


Рис. 1. Схема Форт-интерпретатора

Или еще пример. Введем описание

```
: MY_AUNT ." 100-00-00" ;
```

где 100-00-00 – номер телефона вашей тети. Теперь всякий раз, когда вы напечатаете команду MY_AUNT <ВК>, ЭВМ напомнит вам: 100-00-00 OK.
Другой пример:

```
: SIGNUM 0< IF ." NEGATIVE " ELSE ." POSITIVE" THEN ;
```

Теперь можно не трудиться, определяя знак числа, это сделает за вас ЭВМ. Итак, –10 SIGNUM <ВК> NEGATIVE OK. Рассмотрим пример, несколько более близкий к практике. Пусть ранее была описана переменная A, введем слово с именем A_VALUE:

```
: A_VALUE ." A = "A ? ;
```

В последующей программе обращение A_VALUE будет информировать вас о текущем значении переменной A, например A_VALUE <ВК> A = = 77.

Для вывода на экран символа, соответствующего коду в стеке, служит оператор EMIT (отобразить символ). Ниже приводятся примеры его использования:

```
47 EMIT <ВК> / OK

43 EMIT <ВК> + OK

65 EMIT <ВК> A OK

 7 EMIT <ВК>   OK
```

Для тех, кто не сидит за пультом ЭВМ, поясняю, что в последнем примере ЭВМ выдает звуковой сигнал. Оператором EMIT можно пользоваться также, если вы забыли, какому из кодов ASCII соответствует тот или иной код в интересующей вас системе счисления.

Еще одним весьма употребимым оператором является KEY, который ожидает ввода, берет из входного буфера код ASCII и посылает его в стек. Если вы забыли код * или A, не надо искать таблицу соответствия, просто выполните команду

```
KEY <ВК>                    или    KEY <ВК>
* . <ВК>                           A . <ВК>
42 OK                              65 OK
```

и узнаете, что код ASCII для * равен 42, а A равен 65. Здесь предполагается, что система счисления десятичная.

В версии FIG-FORTH имеется оператор ?TERM (ввод без ожидания), который идентичен KEY, но он ни при каких обстоятельствах не ждет ввода символа с терминала. Если к моменту исполнения ?TERM во входном буфере что-то есть, то код одного символа будет передан в стек, если же нет, управление будет передано следующему за ?TERM оператору. Этот оператор можно с успехом использовать и для выхода из бесконечного цикла:

```
BEGIN XXX ?TERM ASCII Z = UNTIL
```

Выход из цикла произойдет, если нажать клавишу Z.

Набор операторов вывода чисел на экран (печать) можно пополнить словами U., D., .R и U.R. Оператор U. служит для распечатки чисел одинарной длины без знака (знаковый бит рассматривается как обычный двоичный разряд числа). Оператор D. выводит на экран числа двойной длины, например

```
13 0 D. <ВК>  13 OK
0 3 D. <ВК>  196608 OK
```

Операторы .R и U.R служат для табличного представления чисел. Обращение: N k .R и N k U.R, где N – число, которое мы хотим отпечатать, может быть записано в стек и в результате предшествующих вычислений; k – число знакомест, выделенных на экране для печати числа. Младшая цифра числа помещается в самое правое из выделенных знакомест. Например, опишем слово SMPL:

```
: SMPL 12 9 DO 1 5 .R CR LOOP ;
```

исполнив его: SMPL <ВК>, получим:

```
 9
10
11 OK
```

Здесь младшая цифра занимает всегда 5-ю позицию (k=5). Назначение оператора U.R то же, что и .R, но для целых чисел без знака.

Оператор EXPECT (ожидание ввода) служит для ввода последовательности кодов с терминала. Например:

```
0 VARIABLE XX 24 ALLOT
XX 16 EXPECT
или
PAD 120 EXPECT
```

Такая процедура воспримет 16 или 120 символов с клавиатуры и запишет их, начиная с адреса XX или PAD. Если среди вводимых символов встретится код <ВК>, ввод будет немедленно прерван.

Для вывода последовательностей символов на экран используется оператор TYPE (распечатать). Стек при обращении к TYPE должен содержать адрес, начиная с которого надо начать вывод, и число выводимых символов (последнее лежит наверху стека). Например:

```
0 VARIABLE HI 12 ALLOT
: GREAT HI 12 BLANKS       ( заполнение массива HI пробелами)
        CR ." ENTER>"              ( приглашение к вводу)
        HI 12 EXPECT ;          ( запись введенной строки в
                                  массив HI)
```

Предположим, что произошел диалог:

```
GREAT <ВК>
ENTER> HALLO <ВК>
```

Исполнение позднее команды HI 8 TYPE <ВК> вызовет отклик HALLO OK.

Форт имеет свой входной буфер, адрес которого хранится в системной переменной по имени TIB (Terminal Input Buffer). Ввод в этот буфер осуществляется оператором QUERY (запрос) базового словаря:

```
: QUERY TIB @ 120 EXPECT 0 IN ! CR ;
```

где IN – системная переменная, которая является указателем в пределах входного буфера и отмечает байт, подлежащий обработке следующим; CR (возврат каретки) – команда базового словаря, которая выполняет переход на начало следующей строки для терминала или печатающего устройства.

Все операции обмена (при редактировании или загрузке) осуществляются через буферные зоны памяти, каждая из которых содержит 1024 байт. Число этих зон может варьироваться от версии к версии и в зависимости от емкости памяти и требований к скорости обмена может быть от 1 до 6. Эти зоны в памяти и на диске называются в Форте *экранами.*

Рассмотрим трехбуферный вариант системы (рис.2). Ячейка с именем "Номер" содержит номер экрана, записанный в данный буфер (или для которого буфер приготовлен). Знаковый бит этого кода – флаг "спасения" – устанавливается в единичное состояние оператором UPDATE (изменить), сигнализируя о необходимости его записи на диск. Нулевой код в конце буфера служит для прерывания интерпретации, если в процессе его выполнения не встретилось других команд, выполняющих аналогичную функцию.

При обращении к тому или иному экрану система проверяет, есть ли он уже в буфере. Если его там нет, то один из буферов освобождается и туда считывается требуемый экран. Освобождается всегда тот буфер, который был занят раньше остальных. В процессе освобождения контролируется флаг "спасения" в ячейке номера экрана. Если флаг единичный, этот буфер сначала записывается на диск (в FIG-FORTH в виртуальной файл FORTH.DAT, в пределах которого экраны упорядочены в соответствии с номерами),и только затем на его место в буфер производится запись.

Для чтения экрана с диска в буфер используется оператор BLOCK (блок). При обращении к BLOCK необходимо указать номер читаемого экрана (блока). В результате выполнения команды соответствующий экран (если он еще не в буфере) будет занесен в буфер, а в стеке окажется адрес первого байта этого буфера (здесь имеется в виду байт, расположенный сразу за номером экрана). Так, чтобы в буфере оказался экран 1 нужно выдать команду 1 BLOCK.

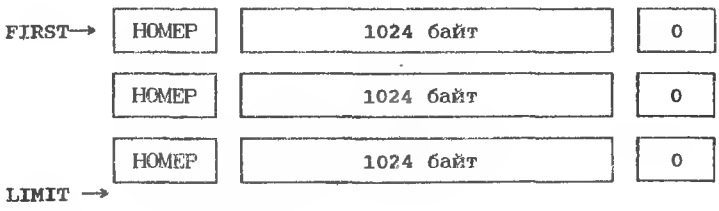


Рис. 2. Схема размещения экранных буферов

Запись на диск всех буферов, имеющих флаг "спасения", происходит по команде FLUSH (сохранить). Примером ее использования может служить оператор n m COPY, который переносит экран n на место экрана m:

```
: COPY SWAP    ( в стеке m n )
       BLOCK   (          m и адрес первого байта буфера,
                          куда считан экран с номером n)
       2 -     (          m и адрес ячейки, где лежит номер
                          экрана, хранящегося в буфере)
       !        ( очистка стека, номер экрана n заменен на m)
       UPDATE                    (установка флага "спасения")
       FLUSH ;                                ( запись на диск)
```

Из комментариев видно, что никакого переноса данных в оперативной памяти не происходит, а происходит только смена номера экрана, установка флага "спасения" и собственно запись буфера на диск на место, предназначенное для экрана m. Например, по команде 7 10 COPY <ВК> ОК содержимое экрана 7 будет перенесено на экран 10. Текст экрана 10 будет утрачен, а на диске будут две копии экрана 7 (в 7-м и 10-м блоках).

Буферные зоны памяти можно использовать и для других целей. Резервирование буферов производит оператор BUFFER (буфер), при обращении к которому нужно указать номер экрана, для которого выделяется буфер (например, 10 BUFFER). При исполнении BUFFER в стек записывается адрес первого байта зарезервированного буфера. Если свободного буфера нет, занимается очередной, при необходимости автоматически производится запись занимаемого буфера на диск. Никакого чтения информации с диска в процессе выполнения BUFFER не производится. Если хранение буфера на диске не планируется, номер экрана, присваиваемый ему, может быть любым. Главное – не установить по ошибке флаг "спасения": это может привести к замене на диске текста соответствующего экрана. Пример такого использования оператора BUFFER можно найти в тексте экранного редактора EDT (см. приложение 1).

Полная очистка всех буферов без записи на диск происходит по команде EMPTY-BUFFER (очистить буферы).

Одна из часто употребимых команд Форта n LIST (листинг):

```
: LIST DECIMAL CR DUP                              ( в стеке n n )
       SCR !          ( запись в SCR номера выводимого экрана;
                        в стеке n)
       ." S# " .       ( отображение номера выводимого экрана)
       16 0 DO          ( начало цикла отображения 16 строк)
       CR I 3 .R SPACE                  ( печать номера строки )
       I SCR @ .LINE                 ( распечатка текста строки)
       LOOP CR ;
```

Эта команда читает содержимое экрана n в буфер и отображает его на экране. При попытке исполнить команду LIST для экрана, где хранится числовая (а не текстовая) информация, ничего хорошего не произойдет. Мало того, что на экран будет выведена непонятная череда символов, сам терминал может переключиться в другой режим, если в выводимом тексте окажутся коды, имитирующие соответствующую управляющую последовательность. Из текста видно, что LIST устанавливает десятичную систему представления чисел, что исключает неоднозначность интерпретации текста. Но "старая" система счисления не восстанавливается, если ранее работала другая система, и это может создать трудности.

Так же как и другие операции ввода-вывода, обмен с диском весьма специфичен для каждого типа ЭВМ. Но операторы LIST, LOAD, BLOCK, .LINE, INDEX и другие являются стандартными для многих версий Форта и ЭВМ. Но есть операторы, без которых нельзя реализовать ни одну из названных процедур. Потребность в них ощущается при написании программ для нестандартного ввода-вывода со стандартного или нестандартного внешнего устройства (например, магнитофона).

Существующие Форт-стандарты не касаются этой деликатной сферы. Один оператор, не являясь стандартным, встречается в различных версиях Форта – это R/W. В случае работы с диском считывает или записывает 1 Кбайт. Обращение: ADR BL# FLG R/W, где ADR – адрес памяти, куда (откуда) производится чтение (запись); BL# – номер экрана (блока); FLG – флаг "чтение-запись" (FLG = 1 для чтения, FLG = = 0 для записи). Возможны модификации, где в качестве параметра указывается код (номер) внешнего устройства, участвующего в операции обмена. Стандартизация операторов такого типа – актуальнейшая проблема, особенно в связи с развитием средств работы с файлами.

При оформлении выдачи текстов на экран необходимо позаботиться о размещении отдельных его частей. Для этого полезным инструментом могут служить операторы SPACE (пробел) и SPACES, являющиеся резидентными для базового словаря в большинстве версий Форта. Первый перемещает курсор терминала на одну позицию вправо, а второй – на n позиций. Эти операторы можно описать следующим образом:

```
: SPACE 32 (код пробела) EMIT ;
: SPACES –DUP IF 0 DO SPACE LOOP THEN ;
```

где операторы –DUP IF...THEN введены на случай нулевого аргумента (если гарантируется неравенство аргумента нулю, их можно опустить).

Например,по команде CR 50 SPACES." FORTH"<ВК> будет отпечатано слово FORTH на правом краю экрана. Принципиальное отличие операторов BLANKS (BLANK в Форт-83) и SPACES заключается в том, что BLANKS засылает коды пробелов в память, а SPACES – на вход терминала или печатающего устройства.

Программист сам при необходимости может описать слово TAB (команда табуляции):

```
: TAB 9 EMIT ;
```

может ввести свою схему табуляции и позиционирования курсора, но для этого уже надо познакомиться с описанием терминала.

Текст из экранного или входного буферов берется интерпретатором порциями (словами). Разрешенным разделителем слов в Форте является пробел (иногда и код <ВК> или TAB). Словом считается последовательность символов, начинающаяся после пробела и завершающаяся пробелом или другим разрешенным разделителем. Выделение слов производится оператором WORD (слово). Обращение к WORD требует наличия в стеке символа, который используется оператором WORD в качестве разграничителя. Обычно это код пробела. В начале входной последовательности может быть любое число пробелов, все они будут проигнорированы. Оператор WORD заносит последовательность символов в память, начиная с адреса, указанного оператором HERE (здесь). Системный оператор HERE выдает в стек адрес первой свободной ячейки памяти, выделенной под словарь к моменту исполнения этой команды. В первом байте будет записано число байт в слове. В разных версиях Форта оператор WORD работает по-разному: в одних выдает в стек адрес памяти, куда записано слово, в других – нет. Поэтому, прежде чем пользоваться им, прочтите соответствующее описание.

Если использовать обращение 1 WORD, то, начиная с адреса HERE, будет записано содержимое всего входного или экранного буфера. Это будет вызвано тем, что код 1 не может быть введен с клавиатуры и, следовательно, не может встретиться среди вводимых символов. Команда '51 WORD ('51 – восьмеричный код символа ")") используется для "обхода" комментариев, а '42 WORD – для выделения строки в структуре ." XXXXX" ('42 – код двойных кавычек). Тип буфера, из которого извлечет оператор WORD очередное слово, определяется системной переменной BLK, равной 0 для входного буфера и номеру экрана в противном случае. Смещение относительно начала буфера задается системной переменной IN, которая корректируется при каждом обращении к WORD.

Если мы хотим распечатать слово, выделенное оператором WORD, с помощью оператора TYPE, нам необходимо поместить в стек адрес HERE+1 (это адрес первого байта слова) и число символов в слове (HERE C@). Запись этих чисел в стек производит оператор COUNT (счет), также содержащийся в базовом словаре:

```
: COUNT DUP 1+ SWAP C@ ;
```

Обращение: HERE COUNT TYPE.

Многие операции выдачи результатов и сообщений на экран производятся через специальный буфер, адрес которого смещен относительно HERE. В базовом словаре имеется оператор PAD, который выдает в стек адрес начала этого буфера:

```
: PAD HERE 68 + ;
```

Другим важным буфером является входной. Адрес его начала хранится в системной переменной TIB (см. описание QUERY).

Сводные данные по процедурам, рассмотренным в этой главе, содержатся в табл.4 – 8.

Таблица 4. Операторы печати чисел

Имя	Состояние стека	Версия	Функция
U.R	u ширина → —	9, 3, M	Печатает число одинарной длины без знака так, что младшая цифра занимает самое правое положение в выделенном поле, заданном числом "ширина"
. R	n ширина → —	9, 3, F	То же, что и U .R, но для чисел со знаком
S.	— → —		Печатает верхнее число в стеке, не меняя значения указателя стека
O.	— → —		То же что и S., но в восьмеричной системе счисления (значение BASE восстанавливается)
D.R	d ширина → —	9, 3, F	Печатает числа двойной длины со знаком. Младшая цифра занимает самое правое положение в поле, размер которого определен числом "ширина"
PAGE	d ширина → —	9, 3, M	Очищает экран и устанавливает курсор в верхнее левое положение

Окончание табл. 4

Имя	Состояние стека	Версия	Функция
HOME	— → —		Устанавливает курсор в верхнее левое положение

Примечание. 9 и 3 — стандарты Форт-79 и Форт-83, M — MMFORTH, F — FIG — FORTH.

Таблица 5. Операторы отображения информации на экране

Имя	Состояние стека	Версия	Функция
.	n → —	9, 3, F, G	Удаляет число из стека, преобразует и отображает на экране
?	n → —	9, F	Удаляет число из стека и отображает содержимое ячейки, адрес которой равен этому числу, на экране
U.	u → —	9, 3, F	То же что и ".", но код в стеке рассматривается как число без знака
." XXXX" PRINT "XX"	— → —	9, 3, F, G	Печатает строку XXXX на экране. Код " завершает строку
EMIT	c → —	9, 3, F	Отображает на экране символ, код которого находится в стеке
CR	— → —	9, 3, F	Посылает коды <ВК> или <ПС> на выходное устройство
SPACE	— → —	9, 3, F	Выдает код пробела на экран
SPACES	n → —	9, 3, F	Выдает n кодов пробела на экран
HTAB		G	

Примечание. См. примечание к табл.1 и 4.

Таблица 6. Операторы для работы с экранами

Имя	Состояние стека	Версия	Функция
BLOCK	u → адр	9, 3, F	Записывает в стек адрес первого байта в блоке с номером u.

Окончание табл. 6

Имя	Состояние стека	Версия	Функция
			Если блок не находится в памяти, он переносится с носителя в буфер, который был занят блоком, вызванным в память раньше других. Если блок, занимавший буфер, был ранее изменен (UPDATE), то этот блок сначала записывается на диск, и только затем на его место будет занесен новый блок
EMPTY-BUFFERS	— → —	9, 3, F	Помечает все блоки как пустые, в результате даже измененные блоки не будут сохранены
UPDATE	— → —	F	Устанавливает в буфере флаг "спасения"
FLUSH	— → —	F	Записывает все измененные экраны на диск
SAVE-BUFFERS		9, 3	
BUFFER	u → адр	9, 3, F	Резервирует блок в памяти, приписывает ему номер u, но никакого чтения с носителя не производится
–TRAILING	адр n1 → адр n2	9, 3, F	Преобразует число символов n1 в строке, начинающейся с адреса "адр", в число n2, не включающее число пробелов, которые имеются в конце строки. Адрес "адр" остается неизменным.

Примечание. См. примечание к табл.4.

Таблица 7. Операторы ввода-вывода

Имя	Состояние стека	Версия	Функция
KEY	— → s	9, 3, F	Ожидает ввода символа с клавиатуры, при его вводе посы-

Окончание табл. 7

Имя	Состояние стека	Версия	Функция
			лает соответствующий код в стек
?KEY ?TERM	— → c	M F	Если во входном буфере есть хотя бы один символ, его код будет записан в стек, если же нет, исполнение программы продолжается без ожидания нажатия клавиши
EXPECT	адр u → —	9, 3, F	Ожидает ввода символов или кода <ВК> с терминала и запоминает их, начиная с адреса "адр"
QUERY	— → —	9, 3, F	Осуществляет ввод строки символов с клавиатуры. Ввод прекращается, если нажата клавиша <ВК> или заполнен входной буфер
TYPE	адр u → —	9, 3, F	Передает u символов начиная с адреса "адр" на выходное внешнее устройство
Y/N	— → —	M	Отображает на экране (Y/N)?, ждет ввода Y или N. Когда соответствующая буква введена, в стек заносится флаг 0 для Y и 1 для N
MOVE	адр1 адр 2 u —	9, 3, F	Копирует область памяти длиной u ячейка за ячейкой начиная с адр1 и записывает ее начиная с адр2

Примечание. См. примечание к табл.4.

Таблица 8. Управляющие операторы

Имя	Состояние стека	Версия	Функция
<CMOVE CMOVE>	адр1 адр2 u → —	9, 3 F	Копирует область памяти размером u байт начиная с адр1 и записывает ее начиная с адр2. Копирование производится с конца последовательности к началу

Окончание табл. 8

Имя	Состояние стека	Версия	Функция
CMOVE	адр1 адр2 u → —	9, 3, F	Копирует побайтно область памяти размером u байт начиная с адр1 и записывает ее начиная с адр2
DECIMAL	— → —	9, 3, F	Устанавливает десятичную систему счисления
OCTAL	— → —	9, 3, F, M	Устанавливает восьмеричную систему счисления
HEX	— →	9, 3, F, M	Устанавливает шестнадцатеричную систему счисления
LIST	n → —	9, 3, F	Распечатывает экран с номером n
LOAD	n → —	9, 3, F	Загружает экран с номером n (компилирует или исполняет)
—–>	— → —	9, 3, F	Дает команду немедленно приступить к интерпретации следующего по порядку экрана
COPY	n m → —	M, F	Копирует экран n на экран m
WIPE	— → —		Заполняет экран кодами пробелов. Используется в контексте EDITOR

Примечание. См. примечание к табл.4.

Упражнение 1. Опишите оператор * LINE, который отображает на экране (или печати) любую строку L экрана S. Обращение: S L * LINE.

Решение.

```
: *LINE 64 * SWAP BLOCK + 64 TYPE ;
```

Упражнение 2. Опишите слово, которое сформирует на экране рамку размером 25×80. Тип символов, образующих рамку, должен задаваться в качестве параметра.

Упражнение 3. Заставьте ЭВМ воспроизвести ваше имя азбукой Морзе, используя звуковой сигнал (7 EMIT).

Тексты программ, как правило, хранятся в виртуальных файлах (например, FORTH.DAT), которые разбиты на секции по 1024 байт – экраны. В экране 16 строк, в каждой по 64 символа (включая пробелы). Чистый экран заполнен пробелами. Размещение текста на экране произвольное, но для облегчения читаемости рекомендуется начинать описание нового оператора с новой строки. Перенос описания с экрана на экран нежелателен (это ограничение можно обойти, например с помощью оператора – –>). Слова, входящие в это описание, могут быть описаны на других экранах. К моменту начала загрузки данного экрана эти экраны должны быть уже загружены.

Разбиение экрана на 16 строк условно. Фактически экран – это одна строка длиной в 1024 символа. Поэтому никаких символов перехода на новую строку или возврата каретки в текст экрана не вносится. В некоторых версиях Форта допустимы коды переноса и возврата каретки. В этом случае редактирование выполняется обычными системными редакторами К 52 NE, ME или EDT.

Если вы записали на строке только один символ, остальные 63, заполненные пробелами, не используются. Это подталкивает программиста на более полное заполнение строк. Но при этом текст программы становится плохо читаемым (трудно, например, визуально найти начало описания конкретного слова). Целесообразнее свободную часть строки заполнять комментарием. Полезно также иметь в виду, что, если имя слова в описании кончается на 64-м байте строки, новое имя должно начинаться со 2-го байта следующей, иначе эти имена просто сольются.

Отмеченные ограничения в силу структурных особенностей Форта несущественны – редко описание слова занимает более 3–6 строк. Если же программисту надо описать более сложное слово, ему следует или выделить в нем определенные функциональные части и описать их в виде отдельных слов, или разбить данное "длинное" описание на части. Это обычно упрощает и отладку программы. Позднее, по завершении отладки, можно, чтобы сэкономить место на диске и память, снова их объединить. Длинные описания противоречат стилю Форта, и их следует избегать.

4.1. СТРОЧНЫЙ РЕДАКТОР EDIT

Простейшим Форт-редактором является строчный редактор EDIT, который не зависит от особенностей операционной системы и типа терминала.

Чтобы начать редактирование, необходимо сначала загрузить редактор. Редактор размещается на последовательности экранов (например, начиная с 50-го), предшествующий экран загружает последующий. Если вы помните это, то можете дать команду 50 LOAD, остальное сделает за вас ЭВМ. Если же вы не хотите держать в голове номер стартового экрана редактора, тем более что обстоятельства могут заставить вас поместить его в другое место, можно использовать некоторый экран, например экран 3, для записи программы загрузки редактора. Тогда загрузка экрана будет всегда осуществляться по команде 3 LOAD вне зависимости, где реально размещен редактор. Для большей наглядности можно ввести константу 3 CONSTANT РЕДАКТОР, описание которой разместить на стартовом экране, а для вызова редактора воспользоваться командой РЕДАКТОР LOAD.

Чтобы приступить к редактированию экрана N, надо выдать команду N EDIT <ВК>. Текст редактора занимает пять экранов. Редактор создает свой словарь процедур, некоторые из них могут иметь имена, совпадающие с именами базовых или загруженных пользователем слов. Команды редактора выполняются только после нажатия клавиши <ВК>. Выход из редактора EDIT осуществляется командой EX, которая производит также запись на диск отредактированных экранов. Запись происходит автоматически при переходе от редактирования одного к редактированию другого экрана.

Выход из системы Форт через <CTRL C> (там, где он разрешен) не сохранит последний редактируемый экран (или группу экранов).

Полный список команд редактора EDIT приведен в табл.9.

Таблица 9. Команды строчного редактора EDIT

Имя	Состояние стека	Функция
T	n → –	Печатает текущую строку, помечая положение курсора
I или I XXX	– → –	Копирует во входной буфер последовательность XXX, если таковая есть, затем вводит его содержимое в текст сразу после курсора
P или P XXX P _ _	– → –	Копирует во входной буфер последовательность XXX, если таковая есть, затем вводит содержимое входного буфера в текущую строку. Если число символов в XXX больше 64, лишние отбрасываются

Продолжение табл. 9

Имя	Состояние стека	Функция
U U _ _ U XXX	— → —	То же, что и P, но входной буфер загружается в строку, следующую за текущей
F F XXX	— → —	Копирует последовательность XXX, если она есть, в буфер поиска, затем ищет такую последовательность в пределах экрана. Курсор ставится сразу после нее. На экране появится строка, содержащая такую последовательность
S S XXX	n → —	То же, что и F, но поиск происходит вплоть до экрана с номером n включительно
N	— → —	Вызывает следующий экран для редактирования, текущий экран сохраняется
B	— → —	Вызывает предшествующий экран для редактирования, текущий экран сохраняется
E	n → —	Стирает символы, указанные в буфере поиска (слева от курсора). Используется после F.
D D XXX	— → —	Копирует последовательность XXX, если она есть, в буфер поиска, ищет ее в пределах текущей строки и стирает
TILL TILL XXX	— → —	Копирует последовательность XXX, если она есть, в буфер поиска, затем стирает все символы вплоть до XXX включительно
X	n → —	Копирует текущую строку или строку n во входной буфер, удаляет ее из текста, остальные строки сдвигает
M	блок, стр → —	Копирует текущую строку во входной буфер, затем вводит содержимое входного буфера в строку, следующую за указанной "стр" в блоке с номером "блок"
R R XXX	— → —	Комбинирует команды E и I и замещает найденную последовательность символов последовательностью XXX, если она есть, или содержимым входного буфера
L	— → —	Выводит на терминал содержимое редактируемого экрана
SAVE	— → —	Копирует содержимое текущей строки во входной буфер

Окончание табл. 9

Имя	Состояние стека	Версия	Функция
EX	– → –		Записывает отредактированный экран на диск, осуществляет выход из редактора
J	n → –		Сдвигает курсор на n символов. Направление сдвига задается знаком числа n (плюс означает сдвиг вправо)
NEWLI	– → –		Раздвигает строки, освобождая строку, следующую за текущей. Входной и поисковый буферы остаются без изменений
TRADE	i k → –		Меняет местами строки с номерами i и k

Команды редактора

L – при обращении <L> <ВК> выдает на терминал содержимое всего экрана вне зависимости от положения курсора (в редакторе EDIT курсор "невидимый", локализовать его можно с помощью команды T); эквивалентна N LIST, где N – номер редактируемого экрана.

T – выводит на терминал строку, на которую указывает курсор, по команде n T курсор перемещается в начало строки n, и она распечатывается на терминале.

I – вводит текст, начиная с места, указанного курсором. Если число введенных символов (M) больше 64–N, где N определяет положение курсора в строке, то M–64+N введенных символов будет потеряно. Таким образом, I производит запись только в одной строке.Текст, введенный после команды I, запоминается во входном буфере (INBUF). Если последовательность символов XXX отсутствует, то, начиная с места, указанного курсором, вводится содержимое входного буфера. Старый текст в строке правее курсора будет следовать за введенным текстом, если для него в строке есть место.

P – также служит для ввода текста. Если за P следует последовательность символов XXX, она копируется во входной буфер, затем его содержимое вводится в текущую строку. Если в строке уже был текст, то он будет стерт. Команда P _ _ (_ _ означает два пробела) очищает текущую строку, заполняя ее пробелами. По команде P содержимое буфера вводится в текущую строку, старый текст, если он был, стирается.

U – аналог команды P, но содержимое входного буфера вводится в строку, следующую за текущей. Если номер текущей строки K, то все

строки начиная с К+1 сдвигаются на одну, содержимое 15-й строки при этом теряется.

X – копирует текущую строку во входном буфере, удаляет ее из текста, а остальные сдвигает, убирая образовавшийся "зазор".

F – копирует последовательность XXX, если она введена в буфер поиска, затем ищет такую последовательность в тексте экрана. Если последовательность найдена, курсор ставится сразу после нее, а строка, содержащая искомую последовательность, отображается на терминале. Если поиск неуспешен, курсор ставится в начало текста экрана. По команде F без последующего текста XXX производится поиск последовательности символов в буфере поиска.

S – аналог команды F, но поиск проводит, начиная с точки, отмеченной курсором в текущем экране, вплоть до экрана с номером n–1.

E – стирает символы, указанные в буфере поиска (слева от курсора), используется после команды F или S.

D – копирует последовательность XXX, если она введена, в буфер, ищет ее в пределах строки и стирает; если последовательность XXX не введена, из буфера поиска берется образец для сравнения.

TILL или TILL XXX – копирует последовательность XXX, если она введена, в буфер поиска, затем стирает все символы вплоть до XXX включительно; если XXX не введена, за образец берется содержимое буфера поиска, загруженное ранее. Если поиск неудачен, стирание не производится.

R или R XXX – комбинация команд E и I, замещает найденную последовательность строкой символов XXX, если она введена, в противном случае для этого используется содержимое входного буфера.

J – при обращении n J курсор смещается на n позиций вправо или влево в зависимости от знака n; если n не указано, по умолчанию считается n=1.

NEWLI – раздвигает строки, освобождая строку, следующую за текущей, входной и поисковые буферы остаются без изменений.

M – при обращении SCR LINE M копирует текущую строку во входной буфер, после чего вводит содержимое входного буфера в текст экрана с номером SCR после строки с номером LINE, содержимое 15-й строки на этом экране теряется.

N – вызывает следующий экран для редактирования, текст текущего экрана записывается на диск.

B – вызывает для редактирования текст предшествующего экрана, текущий экран сохраняется.

TRADE – при обращении i k TRADE меняет местами строки текущего экрана с номерами i и k.

EX – записывает отредактированный экран на диск, осуществляет выход из редактора.

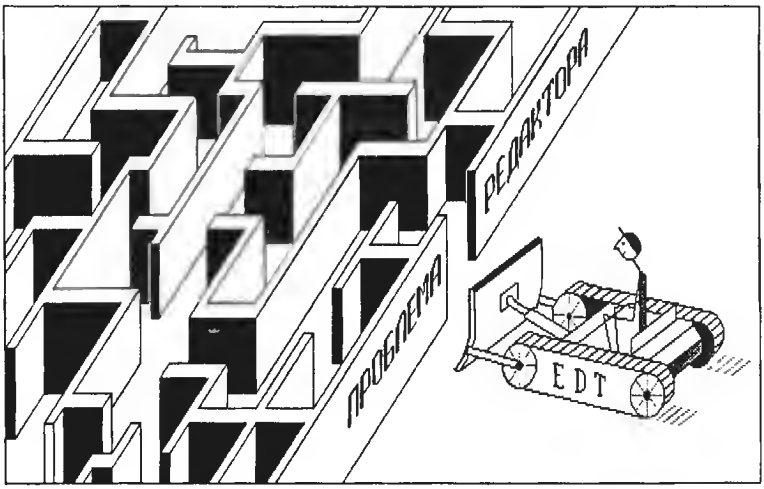
WIPE – очищает текущий экран, заполняя его пробелами.

FLUSH – сохраняет изменения, внесенные в текст экрана, без выхода из редактора EDIT.

Трудно представить себе программиста, который пользуется строчным редактором, когда в его распоряжении имеется экранный редактор. Поэтому и перейдем к его описанию. К сожалению, экранный редактор по своей природе зависим от типа операционной системы (BIOS) и типа терминала.

4.2.ЭКРАННЫЙ РЕДАКТОР EDT

Текст экранного редактора при компактной записи полностью помещается на четырех экранах [30]. Такая запись допустима только для системных библиотек, внесение изменений в которые не планируется. Программа экранного редактора для ЭВМ СМ1420 или ДВК приведена в приложении 1. Предполагается, что клавиатура терминала снабжена соответствующим набором вспомогательных клавиш (например, СМ7209). Для облегчения чтения программа записана менее компактно и снабжена небольшими комментариями и пояснениями. Когда редактор загружен, для начала редактирования экрана достаточно дать команду N EDT. На экране появится текст с указанием номера экрана, на левом поле – символы \, отмечающие край строки. Курсор в стартовом состоянии находится в верхней левой позиции (в этом можно убедиться визуально). Теперь вы в редакторе и можете вносить любые изменения в текст экрана.



Перемещение курсора. Для перемещения курсора по экрану используется вспомогательная клавиатура терминала. Так, для перемещения курсора вверх на одну строку служит клавиша <↑> или <F5> (процедура ^), вниз на одну строку – клавиша <↓> или <F9> (процедура V), соответственно вправо или влево – клавиши <←>(<F7>) и <→> (<F8>) (процедуры << и >>). (Имена клавиш, начинающиеся с буквы F, относятся к персональной ЭВМ IBM PC.)

Если курсор находится где-то на верхней строке и вы нажали клавишу <↑>, то он переместится в начальное положение (вверх слева). Аналогичное замечание справедливо для клавиши <↓>, если курсор находился на последней (15-й) строке, только в этом случае он окажется в крайнем правом положении на 15-й строке. После чего клавиша <↓> какого-либо воздействия на положение курсора оказывать не будет.

Перемещение курсора в начало следующей строки выполняется нажатием клавиши <0> на дополнительной клавиатуре. Причем курсор окажется в начале строки n+1 или n–1 в зависимости от того, какая из клавиш <4> или <5> на вспомогательной клавиатуре были нажаты до этого. Здесь предполагалось, что до нажатия клавиши <0> курсор находился на строке n. Сразу после входа в редактор по умолчанию нажатой считается клавиша <4>, и нажатие на <0> вызовет переход курсора в начало строки n+1.

Аналогичную функцию только для концов строк выполняет клавиша <2> вспомогательной клавиатуры (не путать с цифровыми клавишами основной клавиатуры!).

Перемещение курсора в конец или в начало экрана выполняется нажатием последовательно комбинаций двух клавиш $\overline{\uparrow}$ и <4> (<GOLD> и <4>)или $\overline{\uparrow}$ и <5> соответственно.(Для персональной ЭВМ клавише $\overline{\uparrow}$ соответствует клавиша <ESC> (в описаниях фирмы DEC (США) это клавиша <GOLD>).)

Перемещение курсора в начало следующего слова с учетом направления, заданного клавишами <4> или <5> выполняется при нажатии клавиши <1> вспомогательной клавиатуры.

Введение нового текста. Ввод символов в текст экрана производится нажатием клавиш основной клавиатуры, причем на место, указываемое курсором. Текст строки справа от курсора смещается вправо, а 64-й символ в строке, если он был, безвозвратно теряется (перехода на следующую строку не происходит).

Стирание элементов текста. Символ, на который указывает курсор, стирается при нажатии клавиши <6>(<F6>) вспомогательной клавиатуры (процедура DS).

Символ слева от курсора стирается при нажатии клавиши <ЗБ> основной клавиатуры (или <DEL>, <RUBOUT>, <←>). Стертый символ записывается в специальный буфер емкостью 1 байт.

Слово справа или слева от курсора в зависимости от направления, которое задается клавишами <4> или <5>, стирается при нажатии клавиши <9> (или <F4>).

Часть или вся строка справа от курсора стирается при нажатии клавиши <DEL L> (или <F2>). Если стирается вся строка, текст экрана смещается так, что освободившееся место заполняется последующими строками, при этом 15-я строка заполняется пробелами.

При ошибочном стирании слова, строки или символа их можно восстановить, нажав последовательно клавиши <GOLD> <9>, <GOLD> <DEL L> или <GOLD> <6> соответственно. Стертые слова или строки, так же как и последний из стертых символов, запоминаются в буферах. Это может быть использовано для копирования слова или строки (в версии для персональной ЭВМ содержимое этих буферов постоянно отображается на экране). Для этого курсор перемещается в нужное место, нажимаются клавиши <GOLD> и <9> (или <GOLD> и <DEL L>), и слово (или строка) будет введено в текст экрана. При стирании слова или перемещении курсора в начало следующего слова пробелы перед словом игнорируются.

Здесь уместно напомнить, что текст экрана не содержит кодов <ВК> (коды 15, 12), что послужило причиной введения ряда ограничений. Так, ввод символов осуществляется только в пределах заданной строки, исключение делается лишь для процедуры <GOLD> <3>. Переход на следующую строку возможен при нажатии клавиши <0> (в некоторых реализациях и <ВК>), при этом часть слова может быть на предшествующей строке, а часть на следующей. Текст в буфере не имеет разграничителей строк, после 64-го байта следует текстовый символ 65-го. При переносе слова надо следить, чтобы в нем не возникло разрыва (пробела). Но при вводе строки по команде <GOLD> <DEL L> или <GOLD> <0> (последняя комбинация служит для открытия новой строки) последующие строки сдвигаются, и 15-я строка безвозвратно теряется (нумерация строк начинается с 0).

Операции с частями текста на экране. Строка справа от курсора стирается при нажатии клавиши <ВК>. Стертый текст с предшествующей строки вводится в новую строку. Последующие строки сдвигаются, последняя строка экрана теряется. (Надо сказать, что нажатие <ВК> не вносит в текст никаких символов, видимых или невидимых (это всего лишь команда редактору).) Обратная процедура осуществляется нажатием клавиш <GOLD> <ВК> (<CR>). Строка n+1 стирается (на ее

место вводится строка n+2 и т. д.), и ее содержимое записывается в строку n справа от курсора. Часть, не поместившаяся в строке n, будет потеряна.

Удаление и введение определенной части текста может быть осуществлено следующим образом. Сначала курсор ставится в начало выделяемой части, при этом он указывает на первый элемент строки, который будет удален. Нажимается клавиша <,> (курсор 1 на рис.3). Затем курсор перемещается к концу выделяемого текста и ставится вслед за последним элементом. Нажимается клавиша <3> (курсор 2 на рис.3), выделенный текст стирается с экрана и записывается в буфер. Если стирание произошло ошибочно или сама операция была нужна только для записи в буфер, нажимаются клавиши <GOLD> <3>, и текст экрана восстанавливается в первоначальном виде. Теперь можно переместить курсор в нужное место, нажать <GOLD> <3>, и текст, записанный в буфере, будет внесен в указанное место на экране.

Варианты, приведенные на рис.3, отражают специфику записи текста на экране, где нет символов-разграничителей строк.

При введении информации из буфера последовательность символов на строке, куда будет внесен заключительный сегмент текста, будет сдвинута вправо, и символы, не поместившиеся на этой строке, теряются.

Рис. 3. Работа экранного буфера с сегментами текста

Нажатие клавиш, функции которых не описаны, вызовет звуковой сигнал, но никаких воздействий на текст или положение курсора не произойдет.

Поиск комбинации символов на экране. Нажатие клавиш <GOLD> <8> (или <ESC> <8>) вызовет появление сообщения MODEL: в верхней левой части экрана. Теперь можно напечатать искомую комбинацию символов, после чего нажать клавишу <ВК>. Если поиск завершился успешно, курсор будет указывать на первый символ найденной комбинации. Если поиск неудачный, курсор останется на прежнем месте.

Если вы хотите повторить поиск в последующем тексте, нажмите клавишу <8>. Для управления направлением поиска используются, как и раньше, клавиши <4> и <5>. Но если курсор находится в верхнем левом положении, а направление задано клавишей <5> (реверсивное), то поиск проводиться не будет. Такая ситуация возникает при переходе к редактированию предшествующего экрана с помощью клавиши <7>. То же самое будет при положении курсора внизу справа и направлении <4> (прямое). Для выполнения поиска на данном экране в такой "тупиковой" ситуации смените направление поиска.

Выполнение команд. Для выхода из редактирования необходимо нажать клавиши <GOLD> <7>, в верхней левой части экрана появится сообщение COMMAND. Для записи на диск отредактированного экрана печатаем команду EXIT и нажимаем клавишу <ВК>. Выход из редактора без сохранения отредактированного производится по команде QUIT <ВК>. В обоих случаях текст на экране стирается. В качестве команд могут быть использованы: n EDT, которая заставляет редактор поставить для редактируемого экрана флаг UPDATE и перейти к редактированию экрана n; DECIMAL, если до этого использовалась восьмеричная система счисления, и т. д. К числу полезных команд можно отнести WIPE, очищающую весь экран (заполняющую его пробелами), N M COPY, копирующую экран N на экран M, при этом содержимое экрана M теряется, и FLUSH, сохраняющую содержимое экрана на диске без выхода из EDT. Сигналом начала исполнения команды является нажатие клавиши <ВК>.

Вспомогательные команды. При нажатии клавиши <7> начинается редактирование следующего экрана с учетом направления, заданного клавишами <4> или <5>, при этом содержимое текущего экрана сохраняется. При ошибочном нажатии клавиши <GOLD> для отмены его действия следует нажать клавишу <.> (<,>). При любых сбоях рекомендуется выполнить команду ^W (одновременное нажатие клавиш <CTRL> и <W>).

Редактор имеет буферы для хранения слова (64 байт), строки (64 байт) и выделенного текста (1024 байт). В некоторых версиях имеется буфер и для искомой строки символов. Содержимое их сохраняется при переходе к редактированию другого экрана. Это свойство можно использовать для введения в текст редактируемого экрана фрагментов предшествующего. В рассматриваемом редакторе в целях экономии модель для поиска записывается в область HERE. При переходе к следующему экрану с помощью клавиши <7> эта область сохраняется, что позволяет осуществлять поиск в смежных экранах. Однако если вы воспользовались COMMAND, которая "портит" область HERE, дальнейший поиск становится невозможным. Вам это не нравится? В приложении имеется текст редактора, исправьте его.

Теперь рассмотрим как устроен редактор.

Различные дисплеи имеют разные протоколы управления. Многие из них контролируются специальными управляющими последовательностями символов, которые начинаются с кода 27 (ESC), поэтому их иногда называют ESC-последовательностями. Далее предполагается, что терминал типа СМ7209 или совместимый с ним по системе команд. Позиционирование в нем курсора выполняется последовательностью ESC Y a b, где ESC и Y – восьмеричные коды 33 и 131, a a и b – два числа, равные ›40+x и ›40+y соответственно, x и y – номера колонки и ряда, где находится курсор, причем при x=y=0 курсор фиксирован в верхнем левом углу. Для установки курсора в нужное положение используется оператор FIX:

```
: FIX 13 EMIT            ( сброс счетчика позиций в терминале)
      '54433 PAD !                ( запись в PAD кодов ESC и Y)
      PAD 2+ !      ( запись в PAD будущих координат курсора)
      PAD 4 TYPE ;             ( выдача ESC-последовательности)
```

Обращение: N FIX, где N – число, старший байт которого является кодом колонки, а младший – строки. При передаче на терминал такой ESC-последовательности на экране не отображается никаких символов, а изменяется только положение курсора. Координата курсора на экране определяется переменной R#, которая может принимать значения от 0 до 1023. Причем для первой строки 0≤R#≤63, для второй – 64≤R#≤127 и т. д. Изменениями координат курсора управляет оператор +CUR:

```
: +CUR R# @ + 0 MAX 1023 MIN R# ! ;
```

обращение к которому происходит согласно схеме K + CVR, где K – приращение переменной R#. Номер строки определяет слово L#:

```
: L# R# @ 64 / ;
```

а номер столбца – слово

```
: C# R# @ 64 MOD ;
```

После того как координата курсора определена и присвоена переменной R#, курсор фиксируется в нужном положении оператором ON:

```
: ON L# 36 + C# 35 + SWAP + FIX ;
```

Теперь легко описать операторы перемещения курсора вправо, влево, вверх и вниз:

```
: >> 1 +CUR ;  ( вправо)          : << -1 +CUR ; ( влево)
: ^ -64 +CUR ; ( вверх)           : DWN 64 +CUR ; ( вверх)
```

Эта техника перемещения курсора пригодна не только для экранного редактора. Ввод символа в позицию, отмеченную курсором, производится согласно следующему алгоритму. Сначала все символы строки правее курсора записываются в буфер строки (процедура GAP). Затем в стек записывается адрес экранного буфера, соответствующий позиции курсора, а также код ASCII введенного символа. Курсор смещается на одну позицию вправо, и содержимое строчного буфера укладывается в строку, начиная с места, указанного курсором (процедура PUT). Последний символ в строке теряется. После завершения операции измененная часть строки отображается на экране (процедура SHOW). Описание слова записи вводимого символа:

```
: SS GAP CA          ( вычисление текущего адреса в экранном
                       буфере)
        C! LE                ( последний элемент строки ?)
        IF >> PUT THEN SHW ;
```

При компактной записи он занимает восемь блоков по 512 байт на диске против 55 для редактора К52, что демонстрирует компактность записи программ на Форте.

Запись текста без кодов <ВК>, <ПС> экономит два байта на каждые 64. Для дисков малой емкости это может быть и оправдано. В версиях Форта, где разрешены символы перехода на новую строку, для редактирования используются стандартные системные редакторы. При этом взаимодействие с файловой системой должно быть очень гибким. Такая версия Форта должна позволять интерпретацию текстов в файлах и загрузку в словарь так, как это производит FIG-FORTH для отдельных экранов или их групп.

Особенностью варианта экранного редактора для персональной ЭВМ является то, что процедуры очистки экрана (SCL) и фиксации положения курсора (FIX) содержатся в базовом словаре Форта. Кроме

того, функции управляющих клавиш соответствуют системному редактору EDT персональной ЭВМ.

В заключение отметим некоторые недостатки данной версии редактора:

отсутствие макроопределений типа LEARN (в К52),

сложности при работе с произвольными виртуальными файлами,

отсутствие возможности кратного исполнения команд.

При экранной структуре файлов эти недостатки несущественны. При желании вы без большого труда сможете их устранить.

4.3. ПОСЛЕ РЕДАКТИРОВАНИЯ

После того как текст отредактирован и записан на диск можно осуществить загрузку операторов в словарь Форта. Если требуемые операторы размещены на экране n, следует выдать команду n LOAD (загрузить). Часто программа состоит из большого числа слов и занимает несколько экранов. Тогда можно воспользоваться одной из следующих возможностей:

выдать команду n1 LOAD n2 LOAD и т. д., где n1, n2,... – номера нужных экранов;

если экраны расположены подряд (а к этому следует стремиться), можно в конце текста описаний на экране n поставить оператор – –>, который обеспечит загрузку экрана n+1;

если экраны размещены произвольно, в конце текста на экране ставится команда k LOAD, где k – номер следующего экрана.

Встречаются ситуации, когда нежелательно вводить часть экрана. Эту задачу также можно решить различными путями.

а) Поместить ненужную часть в скобки (XXXX), после левой скобки необходим пробел. Этот способ хорош при условии отсутствия скобок внутри "изолируемого" текста XXXX. Например:

```
: --> ?LOAD 0 IN ! 1 BLK +! ;
: U. 0 D. ;
( : S. DUP . ;
: TRIO 12 EMIT 3 OVER + SWAP DO I LIST LOOP ;)
```

Последние две строчки воспринимаются как комментарий и не загружаются.

б) Если нужно исключить нижнюю часть экрана, можно воспользоваться оператором – –> или k LOAD, помещенным перед нежелательным для загрузки текстом. Например:

```
: * M* DROP ;
: /MOD >R S->D R> M/ ;
```

– –> (или k LOAD) и последующие строки загружены не будут, хотя загрузка следующих экранов будет продолжена:

```
: EMPTY-BUFFERS FIRST @ 3084 ERASE ; (это слово загружено не будет)
```

в) Если нужно убрать ряд заключительных строк на последнем из загруженных экранов, можно перед ними ввести оператор ;S.

Когда необходимо модифицировать загружаемую программу, адаптируя ее, например, к используемому типу терминала, можно определенные части экрана загружать выборочно, но при выполнении определенного условия (условная интерпретация). Для реализации такой возможности опишем массив ENDC и слово IFF:

```
'47105 VARIABLE ENDC '41504 ,
: IFF                        ( оператор условной интерпретации)
        IF BLK @ BLOCK 1K IN @ - 1- 0      ( вычисление числа
                          символов до конца экранного буфера)
           DO 4 0                                  ( поиск ENDC)
              DO ENDC I + C@ OVER IN @ + I + C@ -
                 IF 0 LEAVE     ( ставим флаг FALSE, если это
                                  не ENDC)
                 THEN
              LOOP -DUP 0=
                 IF 1 IN +!    ( коррекция указателя входного
                                 буфера)
                 ELSE 10 IN +! LEAVE       ( если нашли ENDC)
                 THEN
           LOOP DROP
        THEN ;
```

Теперь, если на каком-то экране записать:

```
0 CONSTANT CONDI
CONDI IFF
." START" CR
ENDC DROP
." FINISH"  ,
```

то загрузка такой программы даст отклик:

START
FINISH OK

Если же постоянной CONDI присвоить значение 1, то на экране будет выведена только вторая строка (FINISH). Таким образом, программа между IFF и ENDC будет загружаться только при условии CONDI=0. Приведенный текст IFF – хороший пример того, как используется указатель входного буфера IN.

Если по вашему мнению редактор не имеет некоторых нужных вам процедур, например поиска образа в пределах нескольких экранов, копирование выделенных текстов (а не "вырезка"), вы можете устранить этот недостаток.

На отдельном экране с помощью редактора подготовьте описания нужных слов. Эти описания должны следовать после команды EDITOR DEFINITIONS, что обеспечит включение их в контекст редактора. В конце текста следует написать FORTH DEFINITIONS, чтобы вернуться к Форт-контексту. Загрузив эти описания при загруженном редакторе, вы сможете ими воспользоваться (попытаться отладить) в режиме COMMAND (нажатие клавиш GOLD (клавиша ↓ на дисплее "Электроника ИЭ-15") <7> вспомогательной клавиатуры). Если вы допустили ошибку в программе, можно отредактировать текст дополнительного экрана и повторно его загрузить. Перед загрузкой необходимо выдать команду FORGET YYYYY, где YYYYY – имя первого слова на вашем дополнительном экране. В противном случае в словаре появятся слова-дубликаты, а при загрузке будет выдана последовательность сообщений о повторных описаниях слов. С точки зрения работоспособности программы это не имеет значения, но память, выделенная под словарь, небеспредельна, и при многократном повторении процедуры загрузки этот лимит может быть исчерпан.

Кроме названных можно ввести команду, подсчитывающую число обращений к тому или иному оператору. Результат подсчета может выдаваться на свободное поле вне текста экрана. Такая процедура полезна для оптимизации программ.

При желании обращение к вновь описанным словам редактора может происходить путем нажатия резервных командных клавиш или одновременным нажатием клавиш <CTRL> <SY>, где SY – клавиша, соответствующая выбранному вами символу. Но для этого требуется редактирование базового текста EDT. Прежде чем начать это рискованное дело, запаситесь копией дискеты со "старым" текстом EDT; в противном случае при ошибке у вас не будет инструмента, чтобы ее исправить. Имея копию, вы можете загрузить с нее редактор, поместить на ее место дискету с вашим редактором и продолжить работу над вашим вариантом.

Не следует путать QUIT из редактора и из базового словаря Форта. Хотя их имена идентичны, функции абсолютно различны. Идентичность имен приводит к тому, что при интерпретации EDT выдается предупреждение о повторном описании слова. QUIT в редакторе EDT имеет вид

```
: QUIT EMPTY-BUFFERS SCL ABORT ;
```

SCL стирает текст на экране; ABORT – команда базового словаря Форта:

```
: ABORT SP! ( установка указателя стека параметров в начальное состояние)
        DECIMAL     ( установка десятичной системы счисления)
        CR ."FORTH-PC IS HERE"     ( сообщение о готовности)
        FORTH DEFINITIONS ( словарь Форт сделан контекстным)
        QUIT ;                    ( вход в  QUIT-петлю )
```

которая обеспечивает вход в QUIT, используется при инициализации системы, обработке всевозможных ошибок и выходе системы из нештатных ситуаций. Функция QUIT базового словаря фундаментальна – это основной оператор Форта:

```
: QUIT 0 BLK !                          ( переход в пультовой режим)
       [                               ( установка режима исполнения)
       BEGIN          ( начало  бесконечного  цикла ожидания)
       RP!          ( установка указателя возвратного стека в
                      начальное состояние)
       CR QUERY     ( приглашение ввести команду с терминала)
       INTERPRET        ( интерпретация и исполнение команды)
       STATE @ 0= IF ." OK"        ( сообщение о благополучном
                                выполнении в режиме "исполнение")
                  THEN 0   ( засылка 0 для поддержания цикла)
       UNTIL ;
```

В сущности, пока Форт работает, он исполняет команду QUIT. QUIT – это резидентный монитор системы Форт (подробнее см. в гл.7 ч.II).

Глава 5. Логические операции, циклы и работа со стеком возвратов

5.1. ЛОГИЧЕСКИЕ ОПЕРАЦИИ

Форт предлагает программисту обширный список логических операторов (табл.10). Основная логическая структура Форта

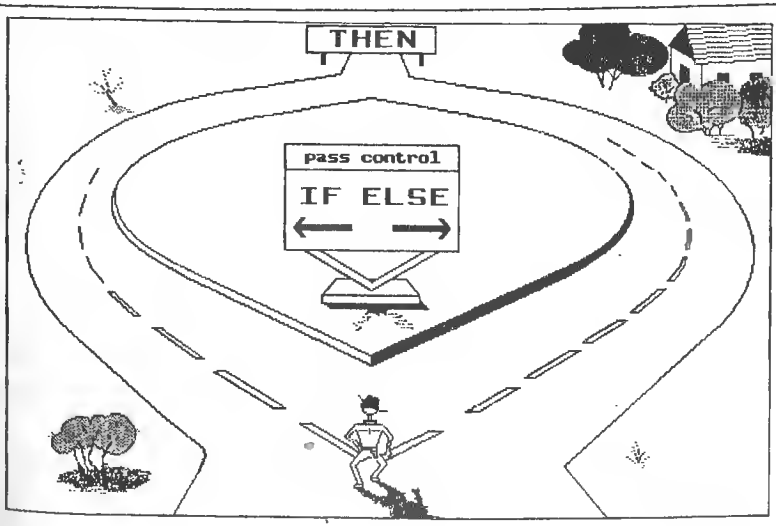
IF A ELSE B THEN C

где A, B и C – некоторые процедуры. Предполагается, что к моменту исполнения оператора IF (если) в стеке флаг f (FALSE/TRUE) : f =0 – FALSE (ложно), f ≠0 – TRUE (истинно), в некоторых версиях Форта TRUE=–1 [19]. Если f=TRUE, выполняется процедура A, в противном случае – B. Процедура C выполняется при любом f. Оператор ELSE (в противном случае) может отсутствовать, тогда f определяет, будет или нет исполняться программа между IF и THEN. Оператор IF требует обязательного наличия слова THEN после него, вместо THEN в некоторых версиях используется оператор ENDIF, который тождествен THEN.

Таблица 10. Условные операторы

Имя	Состояние стека	Версия	Функция
IF XXX ELSE YYY THEN ZZZ	f → —	9, 3, F	Если f = TRUE (≠0), исполняется XXX, в противном случае YYY, ZZZ в любом случае
=	n1 n2 → f	9, 3, F	f = TRUE, если n1 = n2
—	n1 n2 → n1—n2	9, 3, F	f = TRUE, если n1—n2≠0
<	n1 n2 → f	9, 3, F	f=TRUE, если n1<n2
>	n1 n2 →f	9, 3, F	f =TRUE, если n1>n2
0=	n → f	9, 3, F	f=TRUE, если n = 0
0<	n → f	9, 3, F	f=TRUE, если n<0
0>	n → f	9, 3	f=TRUE, если n>0
NOT	f → f	9, 3	Результат предшествующего текста меняется на противоположный. Эквивалентно 0=
AND	n1 n2 → n1&n2	9, 3, F, G	Выполняется логическое И
OR	n1 n2 → OR	9, 3, F, G	Выполняется логическое ИЛИ
XOR	n1 n2 → n3		Выполняется Исключающее ИЛИ
—DUP	n → n n	F	Осуществляет операцию DUP,
?DUP	или 0 → 0	9, 3	если n≠0
?STACK	— → f	F	f=TRUE, если значение указателя стека в пределах допусков
U<	u1 u2 → f	9, 3, F	f=TRUE, если u1<u2, оба числа не имеют знака

Примечание. 9 — стандарт Форт-79; 3 — стандарт Форт-83, F-FIG-FORTH, G-GraFORTH.



Флаг f может формироваться в стеке с помощью арифметических или логических (булевых) операций. К числу логических операций относятся "<", ">" и "=". Эти операции производятся над числами в стеке. Так, в результате операции 10 7 < или 10 7 = в стек будет записано число 0, что означает f=FALSE, а в результате операции 10 7 > число 1 (f=TRUE). Исходные числа из стека удаляются.

Операции 0=, 0> и 0< выполняются над одиночными числами в стеке. Если это число равно 0, больше или меньше нуля, то вместо него записывается флаг истинности (TRUE). Оператор 0=, тождественный оператору NOT, присутствующему в некоторых версиях Форта, преобразует флаг FALSE в TRUE, а TRUE в FALSE. Рассмотрим пример:

```
: SORT DUP 0> IF ." POSITIVE" DROP
              ELSE 0=
                   IF ." ZERO"
                   ELSE ." NEGATIVE"
                   THEN
              THEN ;
```

Если число в стеке равно 0, при обращении к SORT будет выдано сообщение ZERO, а при положительном или отрицательном числе – POSITIVE или NEGATIVE соответственно. Слово SORT при всей непритязательности – хороший пример вложения операторов IF...ELSE... THEN. Два THEN в конце описания – не причуда, а суровая необходимость. Только при их наличии интерпретатор сможет разобраться, что же вы от него хотите. Последовательности типа IF...IF...ELSE...ELSE... THEN...THEN не разрешены, так как непонятно, какому IF какое ELSE соответствует. Ограничений на число вложений операторов IF...ELSE... THEN практически не существует, надо только следить, чтобы каждому IF соответствовал THEN.

Операторы AND, OR и XOR производят логические операции И, ИЛИ и Исключающее ИЛИ над парами кодов в стеке.

Рассмотрим несколько примеров использования логических операций. Например программу работы холодильника (данная программа – полезный пример управления объектом в реальном масштабе времени):

```
0 VARIABLE SWITCH        ( переменная, определяющая состояние
                         компрессора. Запись в нее 1 включает
                         компрессор, 0 - выключает)
-1 CONSTANT LIMIT  ( температура, поддерживаемая в холодиль-
                   нике)

ADDRESS CONSTANT TEMPERATURE ( ADDRESS - адрес, при обраще-
к которому, в стек записывается значение температуры в холо-
дильнике)

: ON 1 SWAP ! ;  ( включение)

: OFF 0 SWAP ! ; ( выключение)
```

```
: FREEZER BEGIN TEMPERATURE @ LIMIT < IF   SWITCH OFF
                                     ELSE SWITCH ON
                                     THEN 0
        UNTIL ;
```

Оператор FREEZER содержит бесконечный цикл BEGIN...UNTIL, внутри которого определяется текущее значение температуры в холодильнике, сравнивается с эталонным значением (LIMIT) и, если температура выше эталонной, включается компрессор, в противном случае он выключается.

Рассмотрим алгоритм работы операторов IF и THEN (начинающие программисты эту часть текста могут пропустить). Описания IF и THEN в самом интерпретаторе написаны на Форте:

```
: IF COMPILE ?BRANCH HERE 0 , 2;IMMEDIATE
```

COMPILE (скомпилировать) вводит в описание компилируемого слова ссылку на оператор, имя которого следует за ним. В приведенном примере это ?BRANCH (ветвление?) – оператор условного перехода, спрятанный внутри интерпретатора, в обычных программах использовать его не рекомендуется. Слово "," вводит в описание интерпретируемого оператора код, хранящийся в стеке (здесь 0).

```
: THEN ?COMP 2 ?PAIR HERE OVER – SWAP!;IMMEDIATE
```

?COMP проверяет, не находимся ли мы в пультовом режиме, и, если это так, выдает сообщение об ошибке; ?PAIR (операторы парные?) контролирует парность операторов IF и THEN. Последующая цепочка операторов вычисляет и записывает в ячейку, зарезервированную оператором IF, адрес ветвления для оператора ?BPANCH. Слово IMMEDIATE (немедленно) указывает, что оператор, описание которого находится непосредственно перед ним, выполняется только при интерпретации. В пультовом режиме слова IF, ELSE и THEN неприменимы, так как они являются операторами немедленного исполнения (IMMEDIATE !).

Пусть требуется оператор, который проверяет число в стеке и формирует флаг TRUE, если оно лежит в заданном интервале значений (например, между 7 и 10), и FALSE в противном случае:

```
 7 CONSTANT L1
10 CONSTANT L2 (L1 и L2 – пределы отбора)
: SELECT DUP L1 > OVER L2 < AND ;
```

При обращении a SELECT число в стеке (a) будет заменено флагом TRUE, если 7<a<10.

Если записать 35000 10 <, в стек будет сформирован флаг TRUE, так как 35000 будет воспринято как отрицательное число (здесь предполагается десятичная система счисления). При работе с кодами

бывает необходимо сравнивать числа, рассматривая бит знака как обычный двоичный разряд (например, сравнение 16-разрядных адресов). Для этой цели используется оператор U<. Команда 35000 10 U < сформирует уже флаг FALSE (см. также описание слова FORGET).

Кроме названных в некоторых версиях Форта имеются операторы:

```
a b <=   f=TRUE, если a ≤ b
a b <>   f=TRUE, если a ≠ b
a b >=   f=TRUE, если a ≥ b
```

Часто вместо логических операторов используются арифметические. Например, нужно проверить условие M≠64 & M>0 (предполагается, что оператора <> (не равно) в системе нет). Проверку первого условия можно организовать как 64 M = 0= или 64 M −. Вторая реализация короче и быстрее, но ее применение без должного внимания может привести к досадным ошибкам. Так, если записать указанное выше условие как M 64 − M 0 > AND, то ошибка неизбежна. Пусть 64 M− дает результат 14, что эквивалентно флагу TRUE (ведь 14≠0). Что же нам даст оператор AND? По логике он должен выдать флаг TRUE. Но AND − операция поразрядная, а мы имеем в стеке 1110 и 1 (двоичное представление) и, естественно, AND даст 0 (т. е. FALSE). Думаю, именно этим соображением руководствовались авторы версии, где TRUE=−1 (ведь это дает единицы во всех разрядах числа).

Если организовать проверку числа M иначе: 64 M − IF M 0> ELSE O THEN, ошибки бы не произошло, а оптимальность по памяти и скорости исполнения сохранилась. Безусловную корректность в таких случаях гарантирует использование при проверках только логических операторов.

Рассмотрим еще один пример. Пусть положение курсора при редактировании лежит в пределах 0−1023, а число, задающее его координату, после очередного приращения записано в стек. Необходимо сохранить это число неизменным, если оно лежит в указанных пределах, сделать его равным нулю, если приращение сделало его отрицательным, и приравнять 1023, если координата превысила верхний предел. Опишем для этого слово LIMI:

```
: LIMI DUP 1023 >                    ( больше 1023 ? )
       IF ( если да) DROP 1023
       ELSE DUP 0<                   ( меньше 0 ? )
            IF ( если да) DROP 0
            THEN
       THEN ;
```

Но нужно помнить и о других путях. Эту же задачу можно решить проще:

```
: LIMI 1023 MIN O MAX ;
```

5.2. ОРГАНИЗАЦИЯ ЦИКЛОВ

В Форте предусмотрено несколько способов организации программных циклов (табл.11). Учитывая, что в Форте нет оператора GO TO (оператор BRANCH не является его аналогом) и, как правило, не применяются метки, операторы цикла наряду с условными операторами приобретают основополагающее значение.

Таблица 11. Операторы циклов

Имя	Состояние стека	Версия	Функция
DO XXX LOOP	DO: lim ind — LOOP: — → —	9, 3, F	Организует конечный цикл, фиксирует индекс (ind) и предел (lim), приращение индекса на 1
DO XXX +LOOP	DO: lim ind — +LOOP: n → —	9, 3, F	То же, что и DO...LOOP, но к индексу добавляется n, а не 1
LEAVE	— → —	9, 3, F	Прерывает цикл по исполнении очередного LOOP или +LOOP
BEGIN XXX UNTIL	UNTIL: f → —	9, 3, F	Организует бесконечный цикл, который завершается, если f=TRUE. Один раз XXX выполняется при любых условиях
BEGIN XXX WHILE YYY REPEAT	WHILE: f → —	9, 3, F	Организует бесконечный цикл. При этом XXX исполняется всегда, а YYY только при f=TRUE. Выход из цикла при f=FALSE
DO XXX /LOOP	DO:u-lim u-ind → — /LOOP: u → —	9, 3, F	То же, что и DO +LOOP, но индекс, предел и приращение являются числами без знака

Примечание. См. примечание к табл.10.

При обращении М К DO...LOOP, где М – число, определяющее предельное значение индекса цикла, а К – начальное значение индекса цикла, после каждого цикла индекс получает приращение 1, процедура между DO и LOOP исполняется М – К раз. Опишем, например, слово ROW, которое печатает числа натурального ряда от 0 до 9:

```
: ROW 10 0 DO I . SPACE LOOP;
```

где слово I выдает в стек параметров текущее значение индекса цикла, не изменяя состояния стека возвратов. Исполнение ROW даст

```
0 1 2 3 4 5 6 7 8 9 OK
```

Оператор DO...LOOP использует два верхних числа стека возвратов – стека для хранения предельного и текущего значений индекса цикла. В вышеприведенном примере перед началом исполнения цикла в стеке возвратов 10 и 0 (последнее число на верху стека).

Предельное и начальное значения индекса необязательно должны быть записаны в стек непосредственно перед оператором DO (исполнить), они могут быть определены в ходе предшествующих вычислений. Имеется возможность вложения одного цикла в другой, например

```
: TAB 10 0 DO 10 0 DO I . LOOP CR LOOP;
```

при исполнении будет выдано 10 рядов чисел:

```
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
. . . . . .
0 1 2 3 4 5 6 7 8 9 OK
```

Оператор цикла М К DO...n +LOOP во многом аналогичен DO... ...LOOP, но приращение индексу цикла (n) задается программой и, вообще говоря, может меняться от цикла к циклу. Например, опишем слово EVEN, которое пропечатывает первые пять четных чисел:

```
: EVEN 10 0 DO I . 2 +LOOP ;
```

Откликом на обращение EVEN будет 0 2 4 6 8 OK. Другой пример:

```
: BACKCOUNT 0 10 DO I . –1 +LOOP ;
```

При обращении BACKOUNT <ВК> даст: 10 9 8 7 6 5 4 3 2 1 OK. Предел и индекс могут быть и отрицательными. Процедура между DO и LOOP будет исполнена по крайней мере один раз. Если это нежелательно, можно записать –DUP IF 0 DO...LOOP THEN и т. д.

Полезно иметь в виду, что все без исключения операторы циклов могут использоваться только внутри описаний слов (пультовой режим для них запрещен).

Цикл BEGIN XXXX f UNTIL относится к бесконечным. Это означает, что при определенных условиях они могут вызвать бесконечное повторение процедуры XXXX. f – число в стеке, воспринимаемое оператором UNTIL (пока не) как логическая переменная TRUE/FALSE. Цикл XXXX будет повторяться до тех пор, пока f =0 (FALSE). Отсюда видно, что последовательность XXXX будет исполнена один раз при любых обстоятельствах. Если оператор описать так:

: BBB BEGIN XXX 0 UNTIL; ,

он будет "циклить" бесконечно. На первый взгляд, истинно бесконечный цикл малопривлекателен. На самом деле это полезно при отладке аппаратуры, но выйти из такого цикла можно только с помощью <CTRL C> (RT-11), при этом управление будет передано пультовому монитору операционной системы. Этого же можно достичь, перезагрузив операционную систему. Поэтому лучше заранее предусмотреть программные средства выхода из цикла. Например: BEGIN XXXX ?TERM ›105 = UNTIL, здесь выход из цикла произойдет сразу, как только вы нажмете клавишу <E>. Примером бесконечного цикла может служить оператор QUIT (гл.4), а также оператор EDT в экранном редакторе:

```
FORTH DEFINITIONS
: EDT ED EDITOR LI ON ESC 61 EMIT OF CB
      BEGIN ' T1 KEY L1 5 TT ON 0
      UNTIL ;
```

(За разъяснениями, как это работает, отсылаю читателей к гл.4 и приложению 1, где имеется прокомментированный текст экранного редактора.) Здесь перед циклом BEGIN...UNTIL производится подготовка режима редактирования, а внутри – ожидание и исполнение редактирующих команд (перемещение курсора, ввод и удаление символов, слов или кусков текста). Выход из этого бесконечного цикла осуществляется с помощью оператора QUIT(уйти) базового словаря Форта. Таким образом, если позаботиться о возможности выхода программы при вводе некоторой последовательности символов (команды) на исполнение процедуры QUIT или ABORT, бесконечный цикл станет не таким уж бесконечным.

Рассмотрим несколько примеров программ, где используются операторы цикла:

: LINE 0 DO ASCII * EMIT LOOP CR ;

При обращении k LINE на экране (или печатающем устройстве) будет отображено k символов *. ASCII – слово, которое указывает интерпретатору, что за ним следует код ASCII (см. гл.7). Используя это описа-

ние, напишем программу, которая напечатает на экране квадратную рамку из символов *:

```
: FRAME DUP LINE DUP 2 - DUP 0 DO DUP ASCII * EMIT SPACES
        ASCII * EMIT CR LOOP DROP LINE ;
```

при обращении 5 FRAME <ВК> на экране будет напечатано

```
******
*    *
*    *
*    *
****** OK.
```

Число k может лежать в пределах 3<k≤24. Последнее ограничение связано с числом строк на экране. Последовательность ASCII * можно при желании заменить числом 42 (десятичный код символа *). Бывает так, что до исполнения программы нельзя определить число необходимых циклов, что связано с характером обрабатываемой входной информации. Чтобы решить эту задачу, можно использовать оператор LEAVE (выйти), обращение к которому происходит только при выполнении определенного условия (см., например, описание слова EXPECT, приведенное в конце параграфа).

Еще одним бесконечным циклом является BEGIN XXX f WHILE YYY REPEAT. Процедура выполняется по крайней мере один раз, так как проверка условия выхода из цикла помещена после нее. Последовательность YYY выполняется только при условии f=TRUE. Выход из цикла происходит при f=FALSE. Флаг f может быть вычислен тем или иным способом, но его наличие в стеке перед исполнением WHILE (в то время как) или UNTIL совершенно необходимо, в противном случае в качестве флага будет рассматриваться код, который оказался к этому моменту в стеке, а это в свою очередь может привести к разрушению программы.

Как устроены операторы цикла?

```
: BEGIN ?COMP HERE 1 ; IMMEDIATE
```

?COMP проверяет, находится ли система в режиме компиляции Если это не так, выдается сообщение об ошибке. Главное назначение оператора BEGIN (начать) – запись в стек текущего значения указателя HERE, которое использует компилятор, чтобы сформировать правильный адрес ветвления при обработке оператора UNTIL или REPEAT (повторить).

Сходную структуру имеет оператор DO:

```
: DO COMPILE XDO HERE 3 ; IMMEDIATE
```

COMPILE XDO компилирует в описание нового слова ссылку на оператор XDO, который при исполнении записывает в стек возвратов исходные параметры цикла (предельное и начальное значения индекса). Флаги 1 и 3 служат для контроля парности операторов цикла. Ниже описан алгоритм UNTIL:

```
: BACK HERE - , ;    (компилирует в описание слова адрес воз-
                      врата)
: UNTIL 1 ?PAIR COMPILE ?BRANCH BACK ; IMMEDIATE
```

Здесь 1 образует пару с флагом =1 в операторе BEGIN, а ?PAIR контролирует эту парность. Далее в новое описание слова вводится ссылка на оператор ?BRANCH, который в процессе исполнения при определенных условиях передаст управление по адресу, описанному в слове BACK.

Структура оператора WHILE еще проще:

```
: WHILE IF ; IMMEDIATE
```

а операторов REPEAT и LOOP чуть сложнее:

```
: REPEAT ROT 1 ?PAIR ROT COMPILE BRANCH BACK
         THEN ; IMMEDIATE
: LOOP 3 ?PAIR COMPILE XLOOP BACK ; IMMEDIATE
```

COMPILE BRANCH компилирует в описание слова ссылку на команду безусловной передачи управления по адресу, сформированному оператором BACK. Сходное назначение команды COMPILE XLOOP, но в недрах XLOOP "спрятан" контроль условия выхода из цикла, т. е. организован условный переход на начало цикла. Все это операторы базового словаря и приведены здесь лишь для того, чтобы показать, насколько они просты. Любой программист после небольшой практики сам сможет описать сходную процедуру, если в этом возникнет необходимость.

Оператор IMMEDIATE всегда (когда требуется) следует непосредственно за только что описанным словом, преобразует его так, что оно работает только на этапе компиляции. Он выполняет очень простую работу – изменяет первый байт описания, устанавливая второй по старшинству (6-й) бит в единичное состояние. Этот флаг используется интерпретатором и помеченное таким образом слово будет работать только при интерпретации.

В заключение рассмотрим алгоритм системного оператора EXPECT, который также использует циклы:

```
: EXPECT          ( в стеке адрес ввода и максимальное число
                    вводимых символов)
    0 DO          ( начало цикла, в стеке только адрес ввода)
```

```
KEY DUP 13 =          ( введенный символ <ВК> ? )
IF ( если да, то) LEAVE DROP 0 (уход из цикла,
     если нажата клавиша <ВК>)
THEN OVER C!          ( запись очередного символа в
                        буфер)
1+              ( в стеке адрес, куда будет введен
                  следующий байт)
LOOP                                  ( конец цикла)
2 ERASE ;   ( запись нулей в два последних байта)
```

Причины последней операции объясняются в описании процедуры прерывания интерпретации.

5.3. СТЕК ВОЗВРАТОВ

По структуре и способу функционирования стек возвратов не отличается от стека параметров (табл.12).

Таблица 12. Операции над кодами в стека возвратов

Имя	Состояние стека	Версия	Функция
>R	n → —	9, 3, F	Записывает число из стека параметров в стек возвратов
R>	— → n	9, 3, F	Записывает число из стека возвратов в стек параметров
I	— → n	F	Копирует верхнее число из стека возвратов и записывает в стек параметров
R		9, 3	
R@			
I'	— → n	9, 3, M	Копирует второе сверху число из стека возвратов и записывает его в стек параметров
J	— → n	9, 3	Копирует третье сверху число из стека возвратов и записывает его в стек параметров
Rp!	— → —	F	Ставит указатель стека возвратов в исходное состояние

Определены процедуры записи числа из стека параметров в стек возвратов >R и обратная процедура R> (из стека возвратов). Оба эти оператора изменяют указатели как стека параметров (SP), так и стека возвратов (RP).

Имеются операторы, не изменяющие значение указателя стека возвратов (I, J, I'). Оператор I (в некоторых версиях R) записывает в стек параметров верхнее число из стека возвратов. Он уже использовался во многих примерах работы с циклами DO...LOOP. Во многих

версиях Форта имеются операторы, которые записывают в стек параметров второе (I') и третье (J) (в Форт-83 нумерация кодов в стеке начинается с 0, а не с 1) числа из стека возвратов. В случае цикла DO...LOOP оператор I' позволяет считывать значение предела цикла, например

```
: T 6 0 DO I' . SPACE LOOP ;
```

При исполнении оператор T отпечатает 6 6 6 6 6 6 OK. Здесь SPACE необходим, чтобы цифры не слились в одно 6-значное число.

В некоторых версиях Форта имеются операторы эквивалентные R> DROP (LEV). Примером использования операторов, работающих со стеком возвратов, является LEAVE (см. описание DO...LOOP):

```
: LEAVE R>     ( занесение адреса возврата в стек параметров)
          R> DROP I >R                    ( индекс – предел)
        >R ;                    ( восстановление адреса возврата)
```

Таким образом, LEAVE приравнивает значения индекса и предела. Эта процедура в базовом словаре реализована на Ассемблере и выглядит проще, так как там не требуется ни первого R>, ни последнего >R операторов. Эти два слова связаны с тем, что стек возвратов играет определяющую роль при переходе от одной процедуры к другой и обратно (отсюда и название "стек возвратов"). При переходе к очередной процедуре адрес возврата запоминается в стеке возвратов, а по завершении операции выполняется процедура ;S, которая восстанавливает программный счетчик, занося в него код из стека возвратов.

При наличии вложенных процедур в стек возвратов оказывается записана целая цепочка адресов возврата. Если выполнить команду LEV (R> DROP), то возврат будет осуществлен не в процедуру, откуда произошел вызов, а в процедуру, более раннего уровня или в систему Форт. Это свойство широко используется в Форте для управления порядком выполнения процедур.

Существует аналог SP! для стека возвратов – это RP!. Оператор RP! присваивает указателю стека возвратов исходное (базовое) значение. Помнить об этом полезно, но пользоваться нужно осторожно. Используя этот оператор внутри процедуры, мы поставим систему в затруднительное состояние – она не будет знать, куда ей передать управление по завершении исполнения, и передаст куда придется. Можно считать, что применение RP! есть привилегия операторов типа QUIT или ABORT. В пультовом режиме эта команда выполнима, но от нее мало проку: в этом случае стек возврата и без того находится в исходном состоянии. Именно поэтому, работая в пультовом режиме, нельзя выполнять операторы R> или LEV. Попытка их исполнить может привести к

непредсказуемому результату. Надо сказать, что, предоставляя программисту большие возможности, Форт дает ему доступ к немалому разрушительному потенциалу.

Важной функцией стека возвратов является возможность временного запоминания кодов из стека параметров. Это делает работу с основным стеком более гибкой. Нужно только не забывать так или иначе восстанавливать стек возврата до выхода из процедуры. Например:

```
: /MOD                ( в стеке параметров делимое и делитель)
      >R                    ( запись делителя в стек возвратов)
      S->D   (преобразование делимого в число двойной длины)
      R>                 ( возврат делителя в стек параметров)
      M/ ;                                ( выполнение деления)
```

Немало аналогичных образцов использования стека возврата для аналогичных целей вы найдете и в других разделах.

Упражнение 1. Пусть массив из 64 элементов 0 VARIABLE MAS 126 ALLOT. В результате работы некоторой программы этот массив заполняется числами. Напишите программу MAXI, которая выбирает максимальный элемент массива MAS и записывает его в стек. Решение см. в§ 8.1 (оператор CMAX).

Упражнение 2. Опишите слово с именем N!, которое вычисляет значение N!, где N — число в стеке. Результат вычисления также записывается в стек.

Решение.

```
: N! 1 SWAP 0 DO I 1+ * LOOP ;
```

Упражнение 3. Опишите оператор, который распечатывает таблицу кодов ASCII в 8-, 16-ричном и десятичном представлении. Имя оператора ASCTAB.

Решение.

```
: ASCTAB ." OCT HEX DECI CHAR   OCT HEX DECI CHAR    "
         ." OCT HEX DECI CHAR"    CR      ( печать заголовка)
         128 32 DO    ( начало цикла, первый символ - пробел)
         I OCTAL 3 .R          ( распечатка 8-ричного значения)
         SPACE I HEX 3 .R   ( распечатка 16-ричного значения)
         SPACE I DECIMAL 3 .R          ( распечатка десятичного
                                         значения кода)
         3 SPACES I EMIT             ( распечатка ASCII-символа)
         5 SPACES I 3 MOD 0= IF CR THEN     ( укладка таблицы
                                              в три колонки)
         LOOP ;
```

При обращении ASCTAB <ВК> будет напечатана таблица в виде:

OCT	HEX	DECI	CHAR	OCT	HEX	DECI	CHAR	OCT	HEX	DECI	CHAR
40	20	32		41	21	33	!				
42	22	34	"	43	23	35	#	44	24	36	$
45	25	37	%	46	26	38	&	47	27	39	'
50	28	40	(	51	29	41	)	52	2A	42	*
53	2B	43	+	54	2C	44	,	55	2D	45	-
56	2E	46	.	57	2F	47	/	60	30	48	0
61	31	49	1	62	32	50	2	63	33	51	3
64	34	52	4	65	35	53	5	66	36	54	6

67	37	55	7	70	38	56	8	71	39	57	9
72	3A	58	:	73	3B	59	;	74	3C	60	<
75	3D	61	=	76	3E	62	>	77	3F	63	?
100	40	64	@	101	41	65	A	102	42	66	B
103	43	67	C	104	44	68	D	105	45	69	E
106	46	70	F	107	47	71	G	110	48	72	H
111	49	73	I	112	4A	74	J	113	4B	75	K
114	4C	76	L	115	4D	77	M	116	4E	78	N
117	4F	79	O	120	50	80	P	121	51	81	Q
122	52	82	R	123	53	83	S	124	54	84	T
125	55	85	U	126	56	86	V	127	57	87	W
130	58	88	X	131	59	89	Y	132	5A	90	Z
133	5B	91	[	134	5C	92	\	135	5D	93	]
136	5E	94	^	137	5F	95	_	140	60	96	Ю
141	61	97	А	142	62	98	Б	143	63	99	Ц
144	64	100	Д	145	65	101	Е	146	66	102	Ф
147	67	103	Г	150	68	104	Х	151	69	105	И
152	6A	106	Й	153	6B	107	К	154	6C	108	Л
155	6D	109	М	156	6E	110	Н	157	6F	111	О
160	70	112	П	161	71	113	Я	162	72	114	Р
163	73	115	С	164	74	116	Т	165	75	117	У
166	76	118	Ж	167	77	119	В	170	78	120	Ь
171	79	121	Ы	172	7A	122	З	173	7B	123	Ш
174	7C	124	Э	175	7D	125	Щ	176	7E	126	Ч

Объясните, почему начало таблицы выглядит именно так и как этого можно избежать? Русская часть таблицы лишь один из вариантов кодировки.)

Упражнение 4. Опишите слово NLOAD, которое при обращении K N NLOAD загружает N экранов, начиная с K.

Упражнение 5. Опишите оператор, который при обращении K N LISTING выдает на экран (или печать) N экранов, начиная с K.

Упражнение 6. Опишите оператор, который при обращении N WORDS печатает число описаний вида : NNN ...; на экране с номером N.

Решение.

```
: WORDS 0 (счетчик описаний) SWAP
        BLK @ >R IN @ >R       ( сохранение в стеке возвратов
                                 значений BLK и IN)
        BLK ! 0 IN !     ( установка новых значений BLK и IN)
        BEGIN '72 WORD ( выделение последовательности симво-
                         лов, завершающейся кодом ":")
             1+           ( инкрементация счетчика описаний)
             IN @ 1023 >        ( проверка значения указателя
                              буфера - условие выхода из цикла)
        UNTIL R> IN ! R> BLK !     ( восстановление IN и BLK)
        1- ." WORD#=" . ;            ( распечатка результата)
```

Упражнение 7. Опишите слово, которое при обращении S1 SN WORD_LIST пропечатает в четыре колонки имена операторов, описания которых содержатся на экранах S1,...,SN.

Решение.

```
: WORD_LIST 1+ SWAP BLK @ >R IN @ >R DO    ( цикл по экранам)
            ." S#=" I . CR          ( печать заголовка экрана)
            I BLK ! 0 IN !        ( начало работы с очередным
                                    экраном)
            0 BEGIN '72 WORD              ( поиск символа ":")
            32 WORD HERE COUNT TYPE              ( печать имени
                                          очередного оператора)
```

```
14 HERE C@ - 1 MAX SPACES     ( организация табу-
                                ляции)
1+ DUP 4 = IF DROP 0 CR THEN      ( формирование
                  переноса на новую строку после
                  распечатки 4 имен)
IN @ 1023 > UNTIL  ( проверка условия завершения
                     работы с текущим экраном)
DROP CR CR         ( переход к следующему экрану)
LOOP R> IN ! R> BLK ! ;
```

Глава 6. Словарь Форта

6.1. СТРУКТУРА СЛОВАРЯ

Словарь Форта состоит из некоторого числа первичных определений-примитивов, написанных на Ассемблере, системного набора слов-определений, имеющих структуру Форта, загружаемых пользователем системных библиотек (редактор, Форт-ассемблер, библиотека для работы с числами с плавающей точкой и т. д.) и собственно определений пользователя. Все эти слова образуют единый словарь.

Описание слова DE в словаре представлено на рис.4. Первый байт описания содержит число символов в имени слова. Старший бит (7-й) этого байта равен 1, а 6-й бит может указывать, что данное слово должно исполняться на этапе компиляции. Ограничение максимального числа байтов в имени в разных версиях Форта различно. Теоретически оно равно 32 (5 бит). В FIG-FORTH это число можно задавать, присваивая определенное значение системной переменной WIDTH. Практически 12≤WIDTH≤32. Старший (7-й) бит последнего байта имени всегда равен 1. Установка старшего бита первого и последнего байтов имени в единичное состояние носит вспомогательный характер и служит для облегчения поиска первого и последнего байтов имени (см. описания операторов NFA и PFA).

Имя используется интерпретатором для распознавания, имеется ли требуемое слово в словаре. В начале формирования имени нового слова первый его байт имеет вид '240 N XOR, где N – число букв в имени. Это препятствует распознаванию слова, описание которого еще

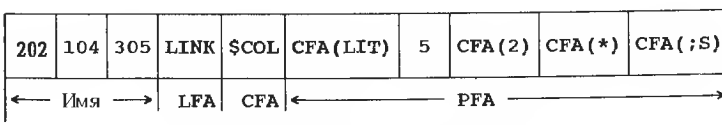


Рис. 4. Структура описания слова DE

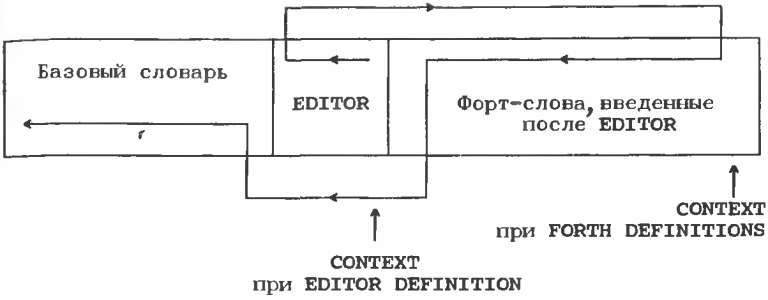


Рис. 5. Маршрут поиска слова при наличии контекстного словаря EDITOR

не завершено. Оператор ";" сбрасывает 5-й бит первого байта описания и тем самым делает данное слово полноправным членом текущего словаря. Для выполнения этой операции используется оператор SMUDGE (SMUG):

```
: SWUDGE LATEST BL TOGGLE ;
```

(Функция оператора TOGGLE описана в гл.7.)

После имени следует поле связи LFA, которое содержит адрес начала описания предшествующего слова (адрес первого байта имени). Схема адресной связи (рис.5) (в описании первого слова поле содержит 0) определяет инаправлениепоиска слов при компиляции и исполнении от конца к началу. Сразу после поля связи в описании слова следует адрес программы, которой должно быть передано управление при исполнении данного слова, – поле CFA. Так, если это слово – константа, управление будет передано программе, которая перешлет число, расположенное сразу после поля CFA, в стек параметров. Вслед за адресом программы идет поле параметров PFA. В описаниях константы или переменной это поле состоит только из одной ячейки. Для массивов размер поля ограничен только наличием свободной оперативной памяти. В случае же стандартного определения типа двоеточия длина этого поля зависит от числа операторов, составляющих программу. При исполнении слова адрес поля CFA (в некоторых версиях PFA) передается оператору EXECUTE (исполнить). Поле параметров определения типа двоеточия содержит адреса полей CFA слов, которые определены ранее и входят в состав данного определения.

Пусть мы имеем описание слова с именем DE:

```
: DE 5 2 * ;
```

(рис.4). Оператор ";" в конце описания преобразуется в адрес CFA оператора ;S, который осуществляет возврат к исполнению программы,

откуда произошло обращение к данному слову; 5 будет интерпретировано как последовательность кодов: CFA слова LIT и собственно числа 5.

При передаче управления какому-либо слову адрес следующего за ним в списке (PFA) слова записывается в стек возвратов. По завершении исполнения этот адрес извлекается оттуда и засылается в программный счетчик. Так начинается исполнение следующего слова. Как поступать с кодами в поле параметров, определяется адресом программы, записанным в поле CFA. В приведенном выше примере при обращении к слову 2 управление передается системной программе DOCON ($CON), которая запишет в стек параметров число первого слова поля параметров описания константы 2.

Аналогичную работу выполняет оператор LIT, только число извлекается из ячейки, следующей за кодом ссылки на LIT (т. е. из текущего описания), содержимое программного счетчика изменяется так, чтобы управление передавалось слову, поле CFA которого находится через одно по отношению к LIT. В приведенном примере управление должно передаваться не 5, а программе "константа 2". Так, образ слова DE в словаре будёт иметь вид, приведенный на рис.4. Все числа восьмеричные. CFA(2) означает, что в ячейке лежит CFA описания системной константы 2. CFA(...) – это CFA оператора, имя которого указано в скобках. При исполнении очередного слова его CFA записывается в стек, откуда оно извлекается оператором EXECUTE, который и осуществляет исполнение программы.

Не все слова при начале и завершении своей работы используют стек возвратов. Примитивы Форта его не используют. Это важно, если вы начнете использовать стек возвратов для управления процессом исполнения программы.

Примитивы (а также операторы Форт-ассемблера) содержат в поле CFA адрес следующего за ним слова (PFA). На месте поля PFA в таких словах расположена программа, написанная на Ассемблере (т. е. в машинных кодах). Описание слова Форта может содержать ссылки на примитивы или на другие слова Форта. При исполнении программы фактически работают только примитивы, остальное нужно лишь для управления последовательностью их выполнения.

При обращении к слову интерпретатор (версия FIG-FORTH) записывает в стек адрес поля PFA этого слова. Создавая описания слов, работающих с различными полями других слов, бывает нужно вычислить адреса любого из полей описания (NFA, LFA, CFA и PFA). Проще определить поле команды. Здесь и далее, если не оговорено обратное, подразумевается, что в исходном состоянии в стеке находится адрес

поля PFA. Поле CFA расположено перед PFA, и для вычисления его адреса достаточно выполнить команду 2– . Для этой цели в базовом словаре имеется оператор CFA. Весьма схож с ним оператор LFA, который выдает в стек адрес поля связи. Это поле, как видно из рис.4, находится перед полем CFA, поэтому оператор LFA можно описать как : LFA 4 – ;. В других версиях Форта для этих же целей используются операторы LINK>, >BODY, >LINK, >NAME, BODY> (табл.13) соответственно.

Таблица 13. Операции над полями описаний в словаре

Имя	Состояние стека	Версия	Функция
CFA	адр1 → адр2	F	адр1 — поле PFA, адр2 — поле CFA (адрес того же слова)
LFA	адр1 → адр2	F	адр1 — поле PFA, адр2 — поле LFA того же слова. Эквивалент 4–
LINK>	адр1 → адр2	3	адр1 — поле LFA, адр2 — поле CFA
NFA	адр1 → адр2	F	адр1 — поле PFA, адр2 — поле NFA того же слова
PFA	адр1 → адр2	F	адр1 — поле NFA, адр2 — поле PFA того же слова
>BODY	адр1 → адр2	3	адр1 — поле CFA, адр2 — поле PFA того же слова
>LINK	адр1 → адр2	3	адр1 — поле CFA, адр2 — поле LFA того же слова
>NAME	адр1 → адр2	3	адр1 — поле CFA, адр2 — поле NFA того же слова
BODY>	адр1 → адр2	3	адр1 — поле PFA, адр2 — поле CFA того же слова
TRAVERSE	адр1 n → адр2	F, MV	Ищет адрес противоположного края имени (NFA) в словаре начиная с адр1. Если n=1, поиск осуществляется в прямом направлении (т. е. в сторону больших адресов), если n=–1 (или 0), направление поиска меняется на противоположное. Адрес найденного байта адр2 записывается в стек

Примечание. F — FIG-FORTH, 3 — стандарт Форт-83, MV — MVPFORTH.

Переход от адреса поля PFA к адресу начала поля имени (NFA) того же слова выполняется оператором NFA: сначала вычисляет адрес последнего байта имени (PFA–5),а затем сканирует имя в направлении его начала и ищет байт с единицей в старшем бите. Обратное преобразование (NFA→PFA) производится в обратном порядке – имя просматривается от начала к концу и ищется байт с единицей в старшем бите (отрицательный байт), к адресу последнего байта имени добавляется 5 и получается адрес поля PFA. Теперь читателю ясно, с какой целью первый и последний байты имени имеют отрицательный знак (1 в старшем бите).

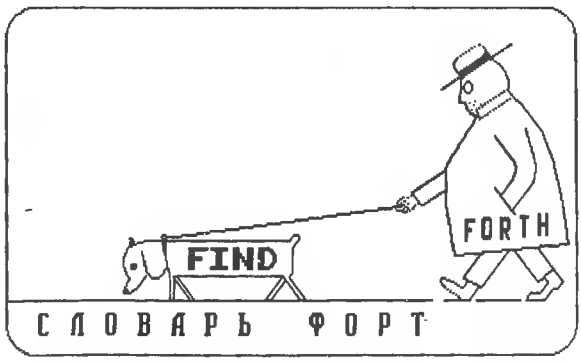
6.2. ПОИСК В СЛОВАРЕ

Команда ' XXXX немедленного исполнения. XXXX – имя слова, содержащегося в словаре, пробел между ' (апострофом) и XXXX обязателен.

```
: ' -FIND 0= 0 ?ERROR    ( если XXXX не найдено, то сообщение
                           об ошибке)
          DROP LITERAL ; IMMEDIATE
```

Оператор ' ищет слово с именем XXXX (из входного или экранного буфера) в словаре Форта. Если поиск увенчался успехом, адрес поля параметров XXXX (PFA(XXXX)) будет записан в стек. При неудачном поиске выдается сообщение об ошибке. Оператор ' удобен, если необходимо проверить, имеется то или иное слово в данный момент в словаре.

При наличии повторных описаний (два или более слов с идентичными именами) по значению PFA в стеке можно разобраться, какая из версий слов сейчас доступна. Если апостроф встречается в описании какого-либо слова, то в силу того, что он является оператором немедленного исполнения, при компиляции в стек будет записано поле PFA



слова, следующего за ним. Например, мы хотим выяснить, с каким словом в словаре связано слово с именем YYY. Для этого опишем слово NEXT-NAME:

```
: NEXT-NAME ' LFA @ DUP @ '77 AND SWAP 1+ SWAP TYPE ;
```

(Апостроф в комбинации '77 указывает на 8-ричную систему счисления (нет пробела). Если вы полагаете, что при обращении NEXT-NAME YYY будет напечатано имя слова, предшествующего YYY, то это заблуждение, так как не учтена немедленность исполнения апострофа. Поэтому при исполнении этого оператора в стек будет записано PFA слова LFA, а не YYY. Чтобы все работало так, как запланировано, необходимо переписать определение слова NEXT-NAME:

```
: NEXT-NAME[COMPILE] ' LFA @ DUP @ '77 AND SWAP 1+ SWAP TYPE ;
```

Еще одно применение апострофа – это изменение значения константы, например

```
200 CONSTANT SALARY
.....................
500 ' SALARY !
```

по команде ' SALARY выдает в стек адрес поля PFA этой константы. Последняя строка приведенного выше примера решает все проблемы. Новое значение константы будет равно 500.

Любопытно, что в случае VARIABLE VVV последовательность ' VVV и просто VVV дают идентичные результаты – записывают в стек PFA переменной VVV.

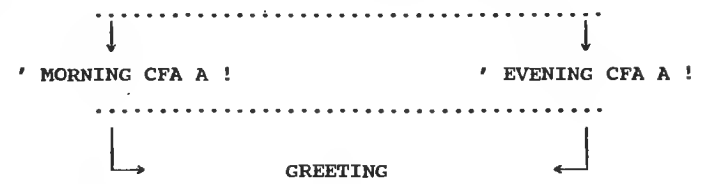
Есть и еще один вариант применения апострофа. Пусть имеется слово HURA, опишем также переменную POINTER:

```
: HURA ." HURA" ;
...................
0 VARIABLE POINTER
Последовательность команд:
' HURA CFA POINTER !
.....................
POINTER @ EXECUTE
```

позволяет исполнить оператор HURA. Конечно, столь замысловатая процедура может вызвать недоумение, ведь оператор HURA можно выполнить и непосредственно. Но в предложенном способе исполнения имеются скрытые возможности. Например, введем описания слов A, MORNING, EVENING и GREETING:

```
: MORNING ." ОЕ УТРО!" ;
: EVENING ." ЫЙ ВЕЧЕР!" ;
0 VARIABLE A
: GREETING ." ДОБР" A @ EXECUTE ;
```

и представим себе программу, которая имеет две ветви:



В результате оператор GREETING отпечатает "ДОБРОЕ УТРО", или "ДОБРЫЙ ВЕЧЕР" в зависимости от того, по левой или по правой ветви "прошла" программа.

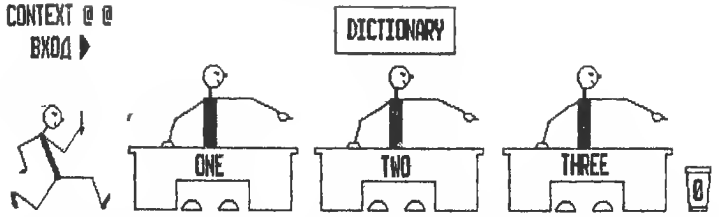
Как уже было показано, словарь Форт имеет цепочную структуру. Поле связи (LFA) слова n указывает на начало описания (NFA) слова n−1. Слово с n=1 имеет 0 в поле LFA. Если слово найдено, поиск прекращается. Именно это свойство позволяет иметь в словаре несколько слов с идентичными именами. Например:

```
: A ." AAA" ;
: B A A ;
: A  B B ;
```

После загрузки такой программы в словаре будет два слова с именами A. Из них доступными для непосредственного исполнения будет только последнее, которое печатает 12 букв A. Слово B печатает 6 букв A, т. е. использует первое из двух описаний слова A. Команда FORGET A уберет последнее слово A, и первое описание A снова будет активным.

Поиск слова в словаре выполняет оператор −FIND

```
: -FIND BL WORD      ( чтение слова из входного или экранного
                       буфера в зависимости от значения BLK и
                       запись его, начиная с адреса HERE)
        HERE COUNT   ( запись в стек адреса (HERE+1) - начала
                       выделенного слова, и HERE C@, равного
                       числу символов в этом слове)
        HERE CONTEXT @ @  (подготовка параметров для поиска.
                          Поиск начнется со слова, на которое
                       указывает системная переменная CONTEXT)
        (FIND)                              (оператор поиска)
        -DUP 0=      (поиск в контекстном словаре неудачен ?)
        IF       (если да, повторяем поиск в текущем словаре)
          HERE CURRENT @ @ (FIND)
        THEN ;
```



Из описания видно, что – FIND осуществляет поиск в контекстном и базовом (текущем) словарях. Для поиска в других словарях надо поменять контекст (системная переменная CONTEXT). О том, как это делать, смотри следующий раздел.

6.3. Контекстные словари Форта

До сих пор речь шла о едином словаре Форта, хотя уже упоминалось о редакторах, ассемблере и других библиотеках. Часто эти библиотеки образуют свои словари, связанные друг с другом и базовым словарем по определенным правилам. Базовый словарь имеет имя FORTH. Для создания нового словаря существует специальный оператор VOCABULARY (табл.14). Если вы хотите создать словарь, скажем с названием EDITOR, вы пишите VOCABULARY EDITOR, и он к вашим услугам.

Таблица 14. Управляющие операторы

Имя	Состояние стека	Версия	Функция
FORTH	– → – (I)	9, 3, F	Делает словарь Форт контекстным
EDITOR	– → – (I)	9, 3, F	Делает словарь EDITOR контекстным
ASSEMBLER	– → – (I)	9A, 3A, FA	Делает словарь ASSEMBLER контекстным
VOCABULARY	– → –	9, 3, F	Слово-описатель, которое создает новый словарь. Обращение: VOCABULARY <name>, <name> – имя нового словаря
DEFINITIONS	– → –	9, 3, F	Контекстный словарь становится текущим, все последующие описания связаны с этим словарем

Окончание табл. 14

Имя	Состояние стека	Версия	Функция
QUIT	— → —	9, 3, F	Очищает оба стека и возвращает управление терминалу. Не выдается никаких сообщений
ABORT	— → —	9, 3, F	Прерывает исполнение, делает словарь Форт контекстным, выполняет QUIT. Распечатывает версию интерпретатора
ABORT"	f → — (I, C)	3	Отображает сообщение, если f=TRUE. Обращение: : <name>.. ABORT" ссс" ; где ссс — текст сообщения; " — сепаратор сообщения
' (апостроф) XXX	— → адр (I)	F	Ищет слово XXX в словаре и записывает в стек его PFA (CFA)
(XXX)	— → — (I)	9, 3, F	Последовательность XXX игнорируется. ")" выполняет роль разграничителя. Используется для комментариев
.(	— → — (I)	3	Немедленно отображает на экране последовательность символов, расположенную между ".(" и ")". ")", является разграничителем
[,]	— → адр (I, C)	3	Определяет и компилирует CFA слова в описании, начинающемся с ":"
\	— → —	MV	Заставляет Форт-интерпретатор игнорировать при загрузке экрана оставшуюся часть строки, ее можно использовать для комментариев

Примечание. I — слово немедленного исполнения, 9 — стандарт Форт-79, 3A — ассемблер Форт-83, 9A — ассемблер Форт-79, C — используется только при компиляции.

Словарь создан, но он пока пуст. Далее вы пишите EDITOR DEFINITIONS и вслед за этим вводите или загружаете какое-то количество описаний. Все они войдут в созданный вами словарь. Это будет продолжать-

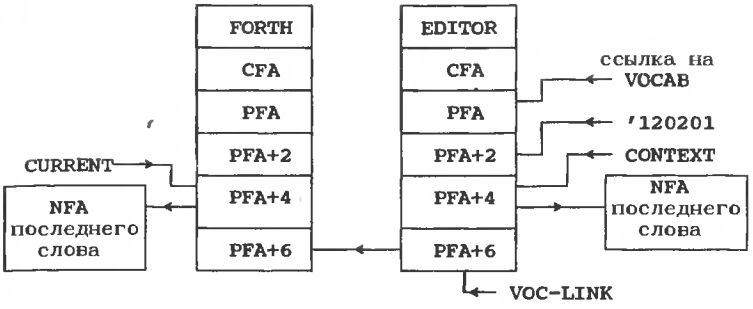


Рис. 6. Структура связей в словарях Форта

ся до тех пор, пока вы будете работать или пока не дадите команду FORTH DEFINITIONS (или ASSEMBLER DEFINITIONS и т. д.). Пребывание в том или ином словаре контролируется системной переменной CONTEXT, в которой хранится ссылка на адрес последнего слова, описанного в данном (контекстном) словаре.

При интерпретации или выдаче команд поиск слова начинается с контекстного словаря, а затем при неудаче повторяется в базовом словаре. Таким образом, если в контекстном словаре имеется слово с именем, идентичным имени в базовом словаре, то будет найдено и использовано слово из контекстного словаря. Понятно, что после команды FORTH DEFINITIONS "старый" контекстный словарь станет недоступным, так как поиск начнется с последнего Форт-слова и обойдет весь словарь EDITOR (рис.6). Причудливый маршрут поиска, отмеченный стрелками, задается адресами поля связи LFA (рис.5). Чтобы получить доступ к словарю EDITOR, надо дать команду EDITOR DEFINITIONS.

Оператор VOCABULARY имеет следующий алгоритм:

```
: VOCABULARY <BUILDS  ( формирование нового имени в словаре)
             '120201 ,          ( имитатор заголовка [PFA+2])
             CURRENT @ CFA , ( ссылка на последнее описанное
                               слово)
             HERE VOC-LINK @ ,          ( ссылка на словарь-
                                          "прародитель")
             VOC-LINK !      ( коррекция переменной VOC-LINK)
             DOES>            ( запись PFA словаря в стек при
                               исполнении)
             2+ CONTEXT ! ; ( CONTEXT указывает на выбранный
                              словарь)
```

CURRENT – системная переменная, хранящая ссылку на последнее слово, описанное в текущем словаре. Имитатор заголовка представляет собой заголовок слова, имя которого – код пробела. Код в поле LFA всегда указывает на начало заголовка предшествующего слова.

Первое слово вновь образованного словаря (например, EDITOR) всегда указывает на имитатор заголовка (’120201) в описании словаря-”прародителя” (в нашем примере FORTH). Такая схема и обеспечивает указанный выше порядок поиска, так как слово после имитатора будет рассматриваться как адрес поля связи.

Если выдадим команду EDITOR, то в стеке окажется PFA этого слова и CONTEXT @ будет указывать на последнее описание из словаря EDITOR. Слово DEFINITIONS (описания) просто присваивает значение CONTEXT переменной CURRENT и данный контекст становится текущим (резидентным). Если этого не сделать, то первый же оператор ”:” поменяет значение CONTEXT и сделает его равным CURRENT. В обычных условиях CURRENT @ равен CONTEXT @.

Упражнение 1. Определите с помощью оператора ’ (апостроф), имеется ли в вашей версии слово VLIST или WORDS?

Упражнение 2. Проанализируйте структуру вашего словаря. Для этого используйте оператор DUMP.

Глава 1. Форматы представления чисел

Код, введенный с пульта, сначала ищется в словаре, а затем при неудаче преобразуется в число с учетом действующей системы счисления и его формата. Если такое преобразование неосуществимо, выдается сообщение об ошибке (MSG# 0). При успешном преобразовании в стек поступает два кода N1 и N2.

На входе любое число представляется в виде последовательности XXXXX кодов ASCII, в одном из трех форматов:

±XXXXX – числа одинарной длины,
±XXXXX.XXXXX – числа двойной длины,
±XXXXX.XXXXXE±YY – числа с плавающей точкой.

Наиболее часто используемый формат первый – для чисел одинарной длины (16-разрядных). Второй формат ориентирован на работу с целыми числами двойной длины (32-разрядными). В третьем формате E – признак порядка ±YY числа. Наличие плюса или минуса перед XXXXX предполагает,что данное число представлено в десятичной системе счисления. По завершении преобразования основание системы счисления (значение системной переменной BASE) не изменяется. Апостроф перед первой цифрой (FIG) означает восьмеричное входное представление числа (например, '10=8). Между +, –, ' и первой цифрой числа не должно быть пробела. Это же касается E и порядка, иными словами входное представление числа не должно содержать пробелов. Библиотеки для работы с такими числами поставляются отдельно и в базовый словарь не входят. Интерпретация чисел выполняется оператором NUMBER (число) (табл.15).

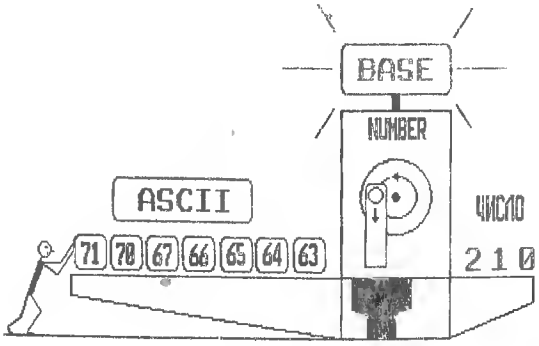


Таблица 15. Управляющие операторы

Имя	Состояние стека	Версия	Функция
NUMBER	адр → n или d	9, 3, F, M	Преобразует последовательность символов, начиная с адр+1 с учетом BASE, в двоичное число одинарной длины (или двойной, если текст имеет соответствующий формат); адр может содержать число символов в тексте
IMMEDIATE	— → —	9, 3, F	Преобразует слово, за описанием которого следует, в оператор, исполняемый при компиляции
COMPILE XXX	— → — (C)	9, 3, F	Используется при описании новых слов. CFA слова XXX вносится в соответствующую позицию PFA нового слова. В результате при исполнении нового слова будет исполнено и слово XXX
[COMPILE] XXX	— → — (I, C)	9, 3, F	Используется в описании типа двоеточия и служит для компиляции слова иемедленного действия XXX, как если бы оно не было таким. Слово XXX будет исполнено тогда, когда будет исполнено слово, в котором использована комбинация [COMPILE] XXX
FORGET XXX	— → —	9, 3, F	Удаляет из словаря, начиная с конца, все слова вплоть до XXX включительно

Примечание. I — слово немедленного исполнения, C — используется в режиме компиляции; 9 — стандарт Форт-79, 3 — стандарт Форт-83, F — FIG-FORT, M-MMSFORTH.

В процессе преобразования NUMBER присваивает значения двум переменным из области USER (FIG-FORTH [30]) DPL и EXP. Последнее содержит целочисленное значение порядка числа, который следует за признаком E, а DPL определяет положение десятичной запятой (во входном представлении это точка), для чисел одинарной длины DPL=−1.

Положение "." не определяет разделения кода на два числа. Пока 16 разрядов хватает, число размещается в одной ячейке. Если для

размещения кода требуется более 16 разрядов, старшая часть записывается в стек поверх первого.

```
DECIMAL
111.11 U. <BK> 0 OK
U. <BK> 11111 OK
DPL @ . <BK> 2 OK

OCTAL
2222.222 U. <BK> 11 OK
U. <BK> 22222
DPL @ . <BK> 3 OK
```

Оператор NUMBER использует только 10 вводимых цифр, что накладывает ограничение на точность. Если будет введено более 10 цифр, результат окажется неверным. Следует также помнить, что NUMBER не преобразует число в формат с плавающей точкой (во всяком случае это верно для некоторых версий Форта), а только подготавливает для этого исходные данные. Окончательное преобразование выполняется оператором из библиотеки для работы с числами с плавающей точкой (например, FL).

Обычно обращение к NUMBER происходит автоматически и не требует вмешательства программиста. Но встречаются ситуации, когда программисту требуется ввести число в диалоговом режиме, тогда он должен записать в стек адрес начала последовательности кодов ASCII, описывающей введенное число, и после этого обратиться к оператору NUMBER. Например, опишем слово TTT, которое запрашивает ввод числа и распечатывает число:

```
: TTT ." ENTER>"                    ( приглашение к вводу)
      QUERY                              ( ввод строки)
      BL WORD                  ( выделение последовательности,
                                 определяющей число)
      HERE NUMBER              ( преобразование числа и занесение
                                 результата в стек параметров)
      DROP . ;                 ( удаление из стека старшей части и
                                 распечатка результата)
```

(О представлении кодов на выходе оператора WORD см. гл.3 ч.I.) По своей функции TTT очень напоминает оператор IN# базовых словарей некоторых версий Форта.

Возможно и другое решение данной задачи с неявным обращением к оператору NUMBER:

```
13 CONSTANT DOZEN
: TTT ." ENTER>" QUERY     ( как   в предшествующем описании)
      INTERPRET .             ( преобразование и распечатка )
      0 TIB @ ! 0 IN ! ;          ( очистка входного буфера)
```

(Разумеется, распечатка не обязательна, она введена для наглядности, для тех, кто изучает Форт за терминалом.) Преимуществом этого варианта является возможность ввода не только значения, но и имени любой константы словаря, например:

```
TTT <BK>                TTT <BK>
ENTER> 66 <BK>          ENTER> DOZEN
66 OK                   13 OK
```

Здесь NUMBER работает внутри INTERPRET, но обращение к нему происходит только при вводе числа. Необходимость очистки входного буфера сопряжена с особенностями работы интерпретатора.

Для внесения в описание слов цифровых констант служат операторы LITERAL и DLITERAL, которые являются операторами немедленного исполнения и работают исключительно на этапе интерпретации. Работу оператора LITERAL поясним следующим описанием:

```
: LITERAL STATE @ IF                    ( если интерпретация)
          COMPILE LIT                ( введение в описание слова
                                   [CFA] системного оператора LIT)
          ,  ( введение в описание слова значения константы)
        THEN ; IMMEDIATE
```

Назначение DLITERAL то же, что и LITERAL, но для чисел двойной длины. Операторы LITERAL и DLITERAL в сочетании с [и] могут помочь сделать программу более читаемой. В программах часто встречаются арифметические выражения над константами, например:

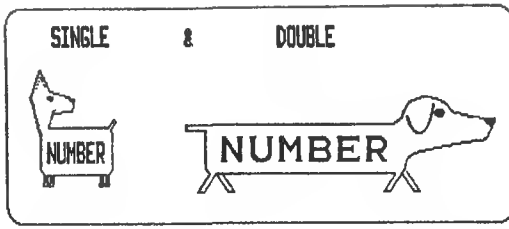
```
0 VARIABLE BB 1022 ALLOT
3 CONSTANT AA
: T*T 4 256 * 10 + AA / BB + + ;
```

Выражение 4 256 * 10 + AA / будет вычисляться каждый раз, когда происходит обращение к T*T. Разумеется, программист может вычислить это выражение с помощью калькулятора и подставить результат в описание T*T. Но тогда не будет видно, как получено это число. Работу может выполнить за вас ЭВМ, для этого достаточно записать:

```
: T*T [ 4 256 * 10 + AA / ] LITERAL BB + + ;
```

и значение этого выражения будет компилировано в наше описание. Причем по месту в оперативной памяти и времени исполнения такой оператор будет эквивалентен тому, где было вычислено и подставлено значение выражения вручную. При использовании таких приемов нужно помнить, что, если в процессе исполнения программы вы меняете значение константы AA, в T*T будет использоваться то значение AA, которое она имела в момент интерпретации. Данную задачу можно решить также, введя константу CC:

```
4 256 * 10 + AA / CONSTANT CC
```



которая описывается до T*T, а в самом T*T рассмотренное арифметическое выражение заменяется именем константы CC. Но, если данная константа используется только один раз, такое решение нельзя признать оптимальным, ведь описание константы занимает в словаре даже при таком коротком имени пять ячеек.

Для повышения точности целочисленных арифметических операций, а также для действий над числами с плавающей точкой в Форте используются числа двойной длины. Для работы с числами двойной длины предназначены прежде всего операторы 2DROP, 2DUP, 2OVER и 2SWAP, являющиеся аналогами операторов DROP, DUP, OVER и SWAP для чисел одинарной длины. Числа двойной длины занимают две следующие друг за другом ячейки стека или памяти. Причем "верхняя" содержит старшую часть числа и определяет знак числа. Оператор 2DROP удаляет два кода из стека, 2DUP копирует число двойной длины и записывает его в стек. Надо сказать, что эти операторы, да и 2SWAP и 2OVER часто используются и при работе с числами одинарной длины. Так, вместо DROP DROP лучше написать 2DROP, вместо OVER OVER – оператор 2DUP, это и экономней по памяти и быстрее при исполнении. Для некоторых арифметических операций требуется преобразование чисел из одного формата в другой. Если число положительное, для этого достаточно положить "поверх него" в стек 0, для отрицательного –1 (›177777). В некоторых версиях эту функцию выполняет оператор S→D (табл.16).

Таблица 16. Операции над числами двойной длины

Имя	Состояние стека	Версия	Функция
D+	d1 d2 → d1+d2	9, 3, F	Складывает два 32-разрядных числа
D−	d1 d2 → d1−d2	9, 3, F	Вычитает два 32-разрядных числа
DMINUS	d → −d	F	Меняет знак 32-разрядного числа

Окончание табл. 16

Имя	Состояние стека	Версия	Функция
(DNEGATE)		9, 3	
DABS	d → |d|	9, 3, F	Заменяет число в стеке его модулем
DMIN	d1 d2 → MIN	9, 3, F	Из двух 32-разрядных чисел в стеке оставляет только минимальное
DMAX	d1 d2 → MAX	9, 3, F	Из двух 32-разрядных чисел в стеке оставляет только максимальное
D=	d1 d2 → f	9, 3	f=TRUE, если d1=d2
D0=	d → f	9, 3	f=TRUE, если d=0
D<	d1 d2	9, 3	f=TRUE, если d1<d2
DU<	ud1 ud2 → f	9, 3	f=TRUE, если ud1<ud2. Оба числа 32-разрядные без знака
S→D	n → d	F	Преобразует 16-разрядное число со знаком в 32-разрядное число со знаком
2!	d адр → —	9, 3	Записывает число двойной длины по адресу "адр"
2@	адр → d	9, 3	Записывает в стек число двойной длины, находящееся по адресу "адр"
2ROT	d1 d2 d3 → d2 d3 d1	9, 3	Эквивалент ROT для чисел двойной длины

Примечание. См. примечание к табл.15.

Для работы с числами двойной длины без знака (U*, U/) предназначены и операторы M*, M/, M/MOD. Оператор M* перемножает числа одинарной длины, но результат выдает в виде числа двойной длины со знаком, оператор M/ делит число двойной длины на одинарное число, результатом является также число одинарной длины со знаком, M/MOD делит двойное число на одинарное, в результате получается остаток и частное двойной длины (последнее на верху стека). Для сложения и вычитания чисел двойной длины используются операторы D+ и D–. При этом стек в исходном состоянии должен содержать четыре кода – два числа двойной длины, после операции там остается два кода результата – одно число двойной длины.

Для печати чисел двойной длины предусмотрен оператор D.R, который отображает 32-разрядное число со знаком, помещая младшую

цифру на правый край выделенного для этого поля. Обращение к нему: d L D.R, d – число двойной длины в стеке; L – число знакомест на экране, выделенное для отображения числа.

Оператор 2! предполагает, что в стеке число двойной длины (d) и адрес (адр), записывает число d в две смежные ячейки с адресами адр и адр+2.

Оператор 2@ предполагает наличие в стеке адреса, в результате значение адреса удаляется, а в стек записывается число двойной длины, находящееся по этому адресу. Старшая его часть оказывается на верху стека.

Работу с числами двойной длины трудно организовать, не используя для их промежуточного хранения стек возвратов. Для реализации таких процедур введены операторы 2>R, 2R> и 2R, которые являются аналогами >R, R> и R, но оперируют с числами двойной длины.

Глава 2. Конструкция <BUILDS...DOES>

Одной из важнейших особенностей Форта является возможность создания новых слов-описателей. Главным отличием этих слов от традиционных является способ использования кодов поля параметров (PFA). С некоторыми словами-описателями вы уже знакомы – это операторы VARIABLE, CONSTANT, с определенными оговорками CREATE (создать) и безусловно : NNN XXX ; . Но создание новых слов-описателей с помощью структуры <BUILDS...DOES>, (в Форт-83 CREATE...DOES>) присуще только Форту (табл.17).

Таблица 17. Управляющие операторы

Имя	Состояние стека	Версия	Функция
<BUILDS XXX DOES> (CREATE XXX DOES>)	Исполнение: – → адр (I, C)	F 9 3	Формирование новых слов-описателей. XXX исполняется на этапе компиляции, текст программы, следующий за DOES>, – на этапе исполиения. При исполнении DOES> выдает в стек PFA вновь определенного слова

Окончание табл. 17

Имя	Состояние стека	Версия	Функция
LITERAL	Компиляция: n → — Исполнение: — → n (I, C)	9, 3, F	Используется только в конструкциях : NNN XXX ; При компиляции переносит код n из стека в описание нового слова, а при исполнении описанного слова записывает в стек указанное число n
DLITERAL (I, C)	Компиляция: n→ — Исполнение: — → n	F	То же что и LITERAL, но для чисел двойной длины
FIND —FIND	адр1 → адр2 n	9, 3 F	адрI — адрес начала счетной строки, содержащей имя слова,которое нужно найти в словаре

Примечание. I — слово немедлеиного исполнения, C — используется при компиляции; F — FIG-FORTH, 9 и 3 — стандарты Форт-79 и Форт-83.

Классическим примером использования такой структуры является формирование новых констант с помощью оператора NEWCONSTANT:

```
: NEWCONSTANT <BUILDS , DOES> @ ;
```

который по своим возможностям тождествен оператору CONSTANT. При обращении 1990 NEWCONSTANT YEAR слово <BUILDS сформирует константу с именем YEAR. В поле PFA будет 0. Оператор "," запишет число 1990 по адресу PFA+2. DOES> (выполнить) заменяет 0 в PFA на адрес оператора, следующего за DOES> в описании NEWCONSTANT, а в CFA YEAR будет записана ссылка на программу $DOE. Последняя при исполнении запишет YEAR в стек PFA+2 и передаст управление программе, которая следует за DOES>. <BUILDS имеет весьма простую структуру:

```
: <BUILDS 0 CONSTANT ;
```

т.е. это слово при обращении <BUILDS XXX, формирует в словаре описание константы равной 0 с именем XXX.

Другим примером использования <BUILDS...DOES> является описание констант с плавающей точкой:

```
: FCON <BUILDS , , DOES> DUP 2+ @ SWAP @ ;
```

Это слово пригодно и для формирования описаний обычных констант

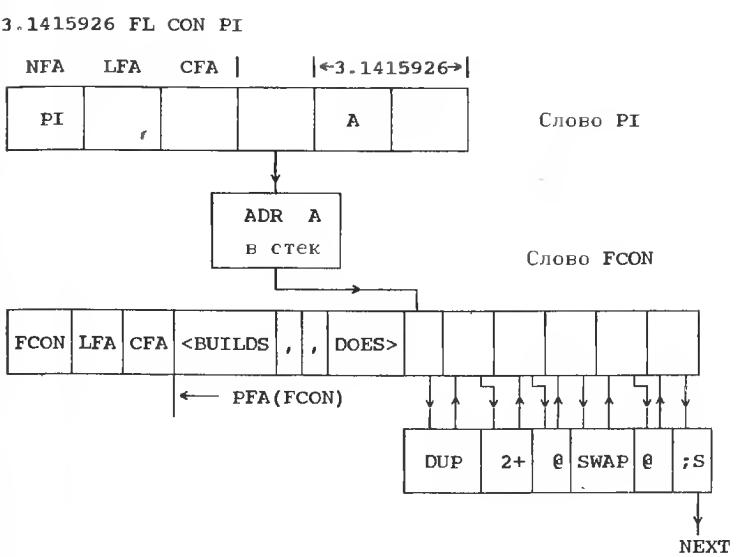


Рис. 7. Структура слова PI

двойной длины. Отличие здесь заключается не в работе FCON, а в содержимом стека при обращении к нему. Обращение в случае константы с плавающей точкой: +XXXXX.XXXXXE±XXXXX.XXXXXE±YY, а в случае целочисленной константы двойной длины: u1 u2 FCON <NAME>, где <NAME> – имя константы; u1 и u2 – любые целые константы одинарной длины, образующие число двойной длины со знаком. При обращении ко вновь описанному слову <NAME> в стек будет записано два кода u1 и u2.

Как работает эта конструкция? Часть программы между <BUILDS и DOES> выполняется только при компиляции, текст после DOES> при выполнении вновь описанного слова (рис.7). В случае констант с плавающей точкой при обращении, например, к константе PI в две верхние ячейки стека будет записано значение n. В библиотеке для работы с числами с плавающей точкой (табл.18) приведено несколько иное описание констант с плавающей точкой:

```
: 2@ DUP 2+ @ SWAP @ ;
: FC FL <BUILDS , , DOES> 2@ ;
```

Обратите внимание на слово FL. Оно выполняется так же, как и <BUILDS, но только при обращении к FC. В первой ячейке поля пара-

Таблица 18. Операции над числами с плавающей точкой

Имя	Состояние стека	Функция
FCON	D→—	Обращение: D FCON <NAME>. Формирует в словаре константу значением D с именем <NAME>
FVAR	D→—	Обращение: D FVAR <NAME>. То же, но для переменной. Для получения значения переменной в стеке надо написать <NAME> 2@
F0=	D→f	Аналог 0=(для чисел с плавающей точкой). Выдает в стек флаг f=TRUE, если D=0
FO>	D→f	Аналог 0>
FO<	D→f	Аналог 0<
F>	D→f	Аналог>
F<	D→f	Аналог<
FTY	D→—	Выдает на терминал в Е-формате значение числа с плавающей точкой
F−	D1 D2→D3	Вычитание чисел D3=D1−D2
F+	D1 D2→D3	Сложение чисел D3=D2+D1
F*	D1 D2→D3	Умножение чисел D3=D1*D2
F/	D1 D2→D3	Деление чисел D3=D1/D2
SIN	D1→D2	D2=Sin(D1), D1 в радианах
TAN	D1→D2	D2=tg(D1), D1 в радианах
LN	D1→D2	D2=ln(D1)
F**	D1N→D2	D2=$D1^N$, где N — целое число со знаком; обращение: D1 N F**
EXP	D1→D2	D2=e^{D1}, где e=2,718281828; обращение D1 EXP
INT	D1→N	Выделяет целую часть числа D1 и записывает ее в стек
1/X	D1 → D2	D2=1/D1, D1≠0
4DUP	D1 → D1 D1	Аналог 2DUP 2DUP
4DRO	D1 D2 → —	Аналог 2DROP 2DROP
SQRT	D1 → D2	D2=+$\sqrt{\text{|D1|}}$; обращение: D1 SQRT
FCHS	D1 → D2	Изменяет знак числа с плавающей точкой: D2=−D1
FABS	D1 → D2	Заменяет число с плавающей точкой в стеке его модулем
F+!	D адр → —	Аналог +! (для чисел с плавающей точкой)
F−!	D адр → —	Аналог SWAP MINUS SWAP +!
F.	D → D	Аналог 2DUP FTY. Пропечатывает число из стека, не меняя его указателя

метров PI указан адрес ячейки поля параметров слова FCON, где лежит адрес перехода в DUP. Такая последовательность переходов начнет осуществляться после того, как в стек будет записан адрес ячейки A поля параметров PI. Все константы с плавающей точкой будут использоваться программу, лежащую в FC после слова DOES, что обеспечивает компактность программы.

Фактически текст после DOES> – это программа, которая обрабатывает заданным образом содержимое ячейки (или ячеек), начиная с адреса PFA+2 вновь описанного слова. С этой точки зрения она аналогична программам, к которым происходит обращение через CFA в случае : NNN XXX ;, CONSTANT или VARIABLE. По своим возможностям структура <BUILDS...DOES> практически не уступает перечисленным выше операторам. Всякий раз, когда вы столкнетесь с необходимостью ввести новый вид данных (особые массивы, комплексные числа, виртуальные структуры данных и т.д.), вспомните о <BUILDS...DOES> и ваша проблема будет решена.

Интересный пример использования структуры <BUILDS...DOES> представляют операторы DOER и MAKE, предложенные Л.Броди [19]. Оператор DOER XXX вводит в словарь новое слово с именем XXX. Если затем написать MAKE XXX ." Что изволите?" ; и обратиться к слову XXX, то ЭВМ выдаст:

Что изволите? ОК

а затем

MAKE XXX . "А кто вас звал? ОК."

Откликом на XXX будет "А кто вас звал? ОК. Вместо оператора ." TTT" может быть записана любая последовательность операторов словаря. Обратите внимание на ";" в конце обращения. Это обязательный атрибут программы. Рассмотренный прием позволяет не только реализовать нехитрый диалог, приведенный выше.

Представьте себе программу со сложным логическим деревом. В некоторую точку при исполнении программы можно попасть различными путями и в зависимости от пути исполнить в этой точке различную работу. Традиционный способ решения предполагает введение переменной-флага, которая формируется в процессе исполнения и в нужном месте анализируется. Более наглядное решение предлагает DOER..MAKE – в XXX каждой из ветвей засылается соответствующая программа, которая и исполняется в оговоренной точке.

Теперь о структуре DOER и MAKE. Ниже приведены схемы их реализации.

```
: RRR ;                                    ("пустое" описание)
```

```
: DOER <BUILDS     ( введение в словарь нового слова с именем
                     из входного потока)
       ' RRR ,  ( определение PFA слова RRR и занесение его в
                  описание нового слова)
       DOES>          ( запись в стек PFA+2 при исполнении
                        нового слова)
       @ >R ;       ( извлечение содержимого этого адреса и
                      запись его в стек возвратов)

: (MAKE) R> DUP @ 4 + SWAP 2+ SWAP ! ;

: MAKE STATE @                                  ( компиляция ? )
       IF COMPILE (MAKE) ( компиляция ссылки (CFA) на слово
                           (MAKE) в новое описание )
       ELSE SP@ CSP !       ( сохранение указателя стека)
            HERE          ( запись в стек HERE, где лежит
                            текст программы-подмены)
          `[COMPILE] '  ( запись в стек PFA слова из вход-
                          ного потока [ XXX - как мы его
                          назвали выше])
            2+ !  ( запись в поле параметров значения HERE)
            SMUG     ( коррекция флага в имени нового слова)
            [COMPILE] ]       ( компиляция команды переход в
                                режим компиляции)
       THEN ; IMMEDIATE       ( установка флага немедленного
                                исполнения)

: UNDO ' RRR [COMPILE] ' 2+ ! ;
```

Опишем слово DRUCK:

```
: DRUCK MAKE XXX ." Кто там ?" MAKE XXX ." Это вы ?"
        MAKE XXX ." Да, это я" ;
```

дадим команду DRUCK <ВК>, а затем исполним XXX несколько раз:

```
XXX <ВК> Кто там ? OK                    (*)
XXX <ВК> Это вы ? OK
XXX <ВК> Да, это я. OK
XXX <ВК> Да, это я. OK
```

При последующих исполнениях XXX будет выполняться задание, описанное последним. Если исполнить DRUCK еще раз, последовательность (*) может быть повторена. Не правда ли, забавный результат? Но операторы DOER и MAKE способны и на серьезную работу по вариации и реконфигурации уже имеющихся программ. Оператор UNDO (отменить) при обращении UNDO XXX возвращает XXX в то состояние, которое он имел после выполнения команды DOER XXX.

Структура DOER...MAKE – прекрасный пример возможностей, которые скрывает в себе Форт. Как же работает оператор DRUCK? После интерпретации на месте MAKE оказывается ссылка на (MAKE). (MAKE) засылает в PFA+2 слова XXX адрес оператора ", стоящего в DRUCK перед "Кто там?". По завершении работы (MAKE) управление передается в систему Форт, так как оператор R> в (MAKE) убрал из

стека команду возврата в DRUCK. При выполнении XXX управление передается программе ." Кто там ?" в XXX, а после ее завершения выполняется очередной раз (MAKE) и ссылка в XXX на ." Кто там ?" будет заменена ссылкой на ." Это вы?" и т.д.

Если описать два слова с именами T и B, а также слово XX:

```
: T MAKE XXX ." TTT" ; : B MAKE XXX ." BBB" ;
: XX IF T ELSE B THEN XXX ;
```

то при 0 XX будет печататься BBB, а при 1 XX будет TTT. Так можно в широких пределах варьировать работу, которую выполняет одна и та же программа. На первый взгляд, аналогичного результата можно достичь, используя условные операторы IF...THEN. Приведенный пример реализуется таким путем довольно легко, но DOER...MAKE позволяет произвольно заменять функцию после того, как оператор уже описан, а IF...THEN на все случаи жизни не запастись.

При вводе текста с клавиатуры неизбежны ошибки (человек – не машина). В режиме редактирования ошибки легко устранить, да и в процессе отладки программы они больших проблем не создают. Но, если диалог встроен в сложную программу управления, где повторный запуск программы ведет к большим потерям, уход в систему Форт через оператор ERROR (ошибка) становится непозволительной роскошью. (В FIG-FORTH ERROR используется в операторах ?ERROR, ?COMP, ?EXEC, ?PAIR, ?LOAD, ?CSP, ?STACK и некоторых других.) Чтобы преодолеть эту трудность в интерпретатор Форта введена системная переменная ERB, значение которой контролируется оператором ERROR и в норме равно 0.

Если ERB=0, ERROR работает обычным образом, в противном случае переменная ERB обнуляется, а уход из программы в Форт через QUIT блокируется. Блокировка осуществляется как при "неузнанном" имени, так и при неправильном вводе чисел, включая ошибки, связанные с конфликтами по системе счисления. Ниже описан алгоритм оператора ERROR:

```
: ERROR HERE COUNT TYPE ." ? "        ( распечатка содержимого
                                        буфера слов)
        ERB @           ( проверка состояния флага блокировки)
        IF 0 ERB ! DROP       ( обнуление ERB и удаление кода
                                ошибки из стека)
        ELSE MESSAGE                    ( сообщение об ошибке)
        SP! QUIT THEN ;     ( восстановление указателя стека)
```

Рассмотрим пример, как можно воспользоваться ERB в программе ввода чисел. С целью блокировки ухода в QUIT при ошибке перед оператором INTERPRET (или NUMBER) надо записать команду 1 ERB !.

Сразу вслед за оператором-преобразователем вводим контроль равенства ERB нулю, например:

```
: TTT BEGIN ." ENTER>" QUERY BL WORD HERE 1 ERB ! NUMBER
          ERB @ DUP 0=
          IF 1 ERB !          ( Установка флага блокировки)
          THEN
      UNTIL ;          ( если после NUMBER   ERB @ = 0, ввод
                         повторяется)
```

Возможны и другие варианты.

Глава 3. Арифметические операции над числами с плавающей точкой

Эффективность Форта особенно заметна при работе с целыми числами (задачи управления, логические, поисковые и др.), но время от времени возникают ситуации, когда без операций с плавающей точкой обойтись нельзя. Тогда на помощь программисту приходят библиотеки, специально созданные для этих целей. Эти библиотеки предназначены для конкретной ЭВМ (это может быть связано, например, с наличием сопроцессора для работы с плавающей точкой). Можно создать и универсальную библиотеку, в которой не используются специальные команды, но ее быстродействие будет не слишком высоким. Практически в любом случае такая библиотека должна использовать Форт-ассемблер, а интерпретатор должен быть способен воспринимать числа, записанные в соответствующем формате. В некоторых вариантах это требует переделки оператора NUMBER. Поэтому, если вы берете библиотеку для работы с плавающей точкой, убедитесь, что она совместима с возможностями вашей версии интерпретатора.

Ниже описана библиотека, которая работает с числами с плавающей точкой. Перед работой с этой библиотекой необходимо загрузить Форт-ассемблер. Преобразование осуществляется с учетом действующей системы счисления, а результат укладывается в две верхние ячейки стека. После преобразования в "плавающую" форму на верху стека окажется код, соответствующий знаку, порядку и старшей части мантиссы. Оператор NUMBER использует не более 10 введенных цифр, ограничения на порядок идентичны существующим для ЭВМ серии СМ, МС1212 и т.д. Преобразование из формата NUMBER в "плавающий" осуществляется оператором FL. Формат обращения: ±XXXXX.XXXXE± YY FL, в результате в двух верхних ячейках стека окажутся два кода, представляющие собой число с плавающей точкой.

Для представления чисел с плавающей точкой на экране (или печати) можно использовать оператор FTY (F.) (см. приложение 2), который печатает число в виде (−). XXXXXXXE±YY, где (−) означает, что вместо знака "+" вводится пробел, например: .3141592E1, −.1250000E0 .3000000E−7. Число цифр после "." постоянно и равно 7. Как само число, так и его порядок являются десятичными вне зависимости от действующей системы счисления. Данная версия работает с форматом чисел, где 15-й разряд первого числа представляет собой знак, 7− 14 − порядок числа, а 0−6 −старшую часть мантиссы, второе число − младшая часть мантиссы. Этот формат накладывает ограничения на диапазон чисел и их точность.

Для реализации "плавающих" операций в интерпретатор введены операторы MUL и DIV, которые работают с целыми делимыми и множителями двойной длины. Делитель и второй множитель − целые числа одинарной длины, а результат всегда целое число двойной длины. Оператор DIV производит округление и уточняет младший бит частного. MUL и DIV не являются примитивами Форта, они описаны как стандартные слова:

```
: DIV >R R M/MOD ROT R> 2/ >          ( остаток больше половины
                                        делителя ? )
      IF 1 0 D+ ( округление) THEN ;
: MUL DUP ROT * >R U* R> + ;
```

На примере этих описаний видно, что, если потребуются действия с числами учетверенной длины, их можно будет реализовать исключительно средствами Форта.

3.1. ПРЕОБРАЗОВАНИЯ ЧИСЕЛ С ПЛАВАЮЩЕЙ ТОЧКОЙ

Преобразование чисел из исходного (входного) формата в "плавающий" производится следующим образом:

```
                 ┌─n─┐
      A0      A1     EX
   XXXXX.XXXXXE±YY
```

Точка может отсутствовать, но при этом не должно вводиться более пяти цифр. Знак после E (знак порядка) также может отсутствовать, тогда порядок считается положительным. Оператор NUMBER преобразует входное представление в число двойной длины. При этом десятичное значение порядка YY будет присвоено системной переменной $EX, а положение десятичной точки задается переменной DPL (см. приложение 8). Так, число OCTAL 11111. 22222E6 будет преобразовано в 4444 и 122222, а DECIMAL 11111. 22222E6 − в восьмеричные 41072 и

60456, первое из чисел – на верху стека. В обоих примерах значения переменных $EX=6 и DPL=5. Сначала производится преобразование к виду .A0A1 *BASE^{EX}, где EX – порядок числа, а A0A1 – целое число двойной длины (во втором примере это .1111122222×10^{11}).

Если BASE=10 (десятичная система счисления), .A0A1 последовательно умножается на 5 и делится на 4, после чего производится нормализация – изменение значения порядка и сдвиг числа, если это требуется. Число таких операций равно целому от деления (EX–n)/3, где n равно числу знаков после запятой. Для восьмеричной и шестнадцатиричной систем алгоритм преобразования существенно проще. Такая методика позволяет обойтись только целочисленными операциями с числами двойной длины. В результате получаем число с плавающей точкой в представлении ЭВМ типа СМ. Длина мантиссы 23 разряда. Сходная методика применима и для обратного преобразования, необходимого для печати результатов расчетов, которые представляются обычно в десятичной системе.

Исходное представление числа с плавающей точкой $f \cdot 2^e$ приводится к виду $F \cdot 10^E$. Так как выходное представление чисел всегда десятичное, то $E \simeq e/3$. При этом $F=f\ (2^{INT(e/E)}/10)^E\ 2^{R(e/E)}$, где INT(e/E) – целое от деления e/E; R(e/E) – остаток от деления e/E. Для осуществления преобразования умножаем сначала f на $2^{R(e/E)}$, затем полученный результат E раз умножаем на $[2^{INT(e/E)}]/10$. Данная схема позволяет работать с числами двойной длины, исключая возможность переполнения или потери точности на промежуточных операциях.

Обращение к оператору представления "плавающих" чисел на экране: A FTY, где A – имя переменной, подлежащей печати: FTY – оператор печати. Число печатается в виде ±XXXXXXXE±YY., знаки числа и порядка могут отсутствовать. Число выводимых цифр не более 7, порядок содержит две цифры. Возможно, вам более по душе формат, аналогичный F-формату Фортрана (операция DECIMAL 11111.22222E6 FL FTY выдаст .1111122E11). Используя программу FTY приложения 2 в качестве исходной, вы без труда напишите соответствующий оператор печати. Операторы для работы с числами двойной длины с плавающей точкой (64 двоичных разряда) в данной версии отсутствуют.

3.2. БИБЛИОТЕКА ОПЕРАТОРОВ ДЛЯ РАБОТЫ С ЧИСЛАМИ С ПЛАВАЮЩЕЙ ТОЧКОЙ

Библиотека работает на ЭВМ, снабженной процессором с расширенным набором команд и с возможностью выполнения операций с плавающей точкой. Для работы с этой библиотекой необходимо загрузить Форт-ассемблер. Полный текст библиотеки представлен в приложении

2. Эти программы могут послужить также примером операторов, написанных на Форт-ассемблере. При необходимости библиотека может быть легко расширена в соответствии со стоящими задачами.

Полный список операторов для работы с числами с плавающей точкой приведен в табл.18.

Для описания константы с плавающей точкой используется оператор FCON, а переменной – FVAR. Все операторы для работы с числами двойной длины в стеке пригодны и для чисел с плавающей точкой. В библиотеку включены и операторы вычисления элементарных функций. В программе SQRT (извлечение квадратного корня) используется алгоритм Герона [38]. Отличительной особенностью данной реализации SQRT является то, что в случае отрицательного аргумента, корень извлекается из его модуля и никаких сообщений об этом "безобразии" программист не получает. При попытке же вычислить 1/X для X = 0 система выдаст сообщение WSC≠26 (или "0 DIVISION", если WARNING=1). TAN (тангенс) реализован с помощью цепных дробей, а LN (натуральный логарифм), SIN (синус), EXP (показательная функция) написаны по рекуррентным алгоритмам. Форма обращения к этим операторам представлена в табл.18. В приложении 2 вы найдете текст оператора INT, который выделяет целую часть числа с плавающей точкой, и F**, который возводит "плавающее" число в целую степень; 2! и 2@, которые являются операторами-аналогами ! и @, но для чисел двойной длины и с плавающей точкой, также описаны в рамках этой библиотеки. Здесь же определены и некоторые константы, например: 1EO, 2EO, EE=2,71828183 (основание натурального логарифма), PI=3,1415926 и некоторые другие.

Упражнение 1. Напишите программу для расчета Arcsin и Arctg.

Упражнение 2. Опишите оператор для расчета своей любимой функции, например функции Бесселя.

Глава 4. Вспомогательные операторы и программы

4.1. ВЫХОДНОЕ ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИИ

Все операции в любой цифровой ЭВМ производятся над двоичными кодами – сегодня это технически проще. Но представьте себе, на экране вместо FORTH IS HERE появится

```
01001111 01000110 01010100 01010010 00100000 01001000
01010011 01001001 01010010 01000101 00100000 01000101
```

или вместо числа 17 – код 10001? Вы, конечно, предпочтете, чтобы текст был написан буквами, а цифры в десятичном (за редким исклю-

чением восьмеричном или шестнадцатеричном) представлении. И в любом случае вы хотите иметь возможность управлять процессом представления результатов на экране (или печати).

Некоторые операторы представления символов и текстов уже описаны (EMIT, TYPE, ."..." и др.), в прикладном плане рассмотрены и некоторые операторы представления чисел (".", "?", "S.", "U.R" и пр.). Для написания многих программ этого вполне достаточно. Но программисту приходится решать и неординарные задачи. Допустим, надо написать программу для базы данных. Банк данных должен пополняться, поэтому на магнитном диске следует хранить указатели свободных зон оперативной и внешней (дисковой) памяти, предназначенных для записи новых данных. Можно, конечно, хранить эти данные в двоичной форме и по запросу выводить в удобной форме на экран, но возможно и более наглядное решение.

На одном из экранов в начале текста программы, например 1-м, описываются переменные или массив переменных, определяющие начальные адреса свободных ячеек для записи информации. При загрузке программы эти переменные оказываются включенными в словарь и в процессе работы корректируются, отражая реальное состояние системы на текущий момент. По завершении работы с базой данных эти переменные заносятся в буквенно-цифровом виде на выделенный для этого экран (1-й) и при желании могут быть прочитаны с помощью стандартной команды LIST.

В качестве примера рассмотрим тексты программы-справочника по интегральным схемам (ИС) [34]. Программа позволяет получать данные о наличии ИС и зарубежных аналогов, динамике расхода, ожидаемых поступлениях, об основных параметрах, цоколевке ИС и т.д., имеет встроенный аппарат пополнения списка ИС, редактирования данных и внесения изменений (расход-приход). Эту программу нельзя рассматривать как базу данных, так как она не имеет аппарата настройки на тип, характер и структуру хранимой информации, узко специализирована для цифровых ИС и рассчитана на малые объемы памяти. Но с другой стороны, текст программы занимает шесть экранов (около 90 строк). А для информации о 500 типах ИС требуется на диске всего 200 блоков по 512 байт. Современные базы данных представляют значительно больше услуг, но указанная программа рассчитана на использование на стенде инженера-наладчика с микроЭВМ и мини-диском.

Прежде чем записать в текст на диске какое-либо число нужно его преобразовать в последовательность кодов ASCII, ведь только они доступны для непосредственного воспроизведения на экране. Аналогичная задача возникает при реализации, например, оператора ".".

Такое преобразование осуществляется посредством процедур <#, #S, #, #>, SIGN и HOLD (табл.19). Все они используют временный буфер с именем PAD, положение которого жестко связано с верхней границей словаря, определяемой оператором HERE.

Таблица 19. Операторы преобразования чисел для выходного представления

Имя	Состояние стека	Версия	Функция
<#	u → u	9, 3, F	Начинает процесс преобразования числа в последовательность кодов ASCII. Исходное число в стеке должно быть двойной длины без знака
#		9, 3, F	Преобразует одну цифру и записывает ее в выходной буфер (PAD), выдает цифру всегда, если преобразовывать нечего, записывается 0
#S		9, 3, F	Преобразует число до тех пор, пока не будет получен 0. Одна цифра выдается в любом случае (0)
#>	→ адр n	9, 3, F	Завершает преобразование числа. В стеке остается число полученных символов и адрес, как это требуется для оператора TYPE
SIGN		9, 3, F	Вводит знак "минус" в выходной буфер, если третье число в стеке <0. Обычно используется непосредственно перед #S
HOLD	S → –	9, 3, F	Вводит в текущую ячейку выходного буфера символ, код которого в стеке. Должен использоваться между <# и #>

Примечание. 9 и 3 – стандарты Форт-79 и Форт-83, F – FIG-FORTH.

Оператор HOLD, используя системную переменную HLD, которая является указателем позиции в выходном буфере (обычно PAD), вводит в выходной буфер байт из стека и уменьшает на 1 значение указателя HLD.

Оператор <# присваивает начальный адрес PAD-буфера переменной HLD:

```
: <# PAD HLD ! ;
```

подготавливая все для преобразования. При этом предполагается, что преобразуемое число занимает две верхние ячейки в стеке параметров и не имеет знака.

Оператор # преобразует одну цифру и помещает результат в выходной буфер, выдает 0, если преобразовывать нечего (или встретится

код, которому нельзя поставить в соответствие никакую цифру). Преобразование осуществляется с учетом основания действующей системы счисления BASE. Представление о работе # можно получить из описания:

```
: # BASE @ M/MOD ROT'11 OVER < IF 7 + THEN'60 + HOLD ;
```

Оператор #S преобразует число до тех пор, пока в результате очередного преобразования не будет получен 0:

```
: #S BEGIN # 2DUP OR 0= UNTIL ;
```

Таким образом один 0 будет передан на выход в любом случае.

Оператор SIGN посылает в выходной буфер код знака "минус", если третье сверху число в стеке отрицательное. Обычно SIGN стоит непосредственно перед #S.

Оператор #> завершает преобразование, в результате в стеке оказывается адрес начала строки символов, отображающих преобразованное число, а также их количество. Таким образом содержимое стека может быть непосредственно использовано оператором TYPE для отображения результата преобразования на экране (или печати). Поясним работу операторов #> и SIGN следующим описанием:

```
: #> 2DROP HLD @ PAD OVER - ;
: SIGN ROT 0< IF '55 HOLD THEN ;
```

Теперь вспомним о задаче, с которой начался разговор, – о преобразовании числовых кодов в последовательность кодов ASCII. Предположим, в стеке указан адрес, начиная с которого нужно записать эту последовательность. Это может быть адрес в пределах буфера экрана, что позволяет записать новый образ числа на диск. Пусть эта информация должна быть занесена на строку 2 экрана 12, первые две строки которого приведены ниже:

```
: $ VARIABLE ;  0 $ BF 152 ALLOT  0 $ .E 0 $ NSC : R@ R# @ ;
 27 $ BLP 813 $ PNT  88 $ PBL 948 $ OFS 42 $ PIB 775 $ PIO
```

Оператор VARIABLE переопределен, чтобы сделать последующие записи более компактными. Предположим, что в данный момент переменная BLP=28, тогда перезапись текста экрана можно произвести с помощью следующей программы:

```
12 BLOCK   ( в стеке адрес первого байта буфера, куда считан
           экран 12)
64 +               ( в стеке адрес первого байта строки 2
                   [ADR])
BLP @              ( в стеке значение переменной BLP = 28)
```

```
0 <# #S #>    ( число 28 преобразовано в последовательность
          кодов ASCII. В стеке адрес буфера и число символов)
R# !                              ( запись числа символов в R#)
4 R# @ - ROT                        ( в стеке  BUF 4-R#@ ADR)
DUP 3 BLANKS                ( заполнение места, где записано 27,
                              пробелами)
+ R# @      r                       ( в стеке BUF ADR+4-R#@ R#@)
CMOVE          ( передача символов, число которых равно R# @,
                 из буфера с адресом BUF в буфер экрана,
                 начиная с ADR+4-R#@)
```

Теперь, если выполнить команды UPDATE и FLUSH, а затем просмотреть экран, дав команду 12 LIST, то на месте числа 27 будет число 28. Данный фрагмент можно записать короче, если работать только со стеком и не использовать R#. Эту работу можно было выполнить, используя редактор EDT, предварительно выяснив значение переменной BLP. Но, во-первых, таких изменяемых переменных в данной программе шесть, а во-вторых, надо это делать каждый раз своевременно и безошибочно, иначе данные в справочнике начнут укладываться друг на друга. Автоматическая коррекция имеет ясные преимущества, да и сохранение наглядности немаловажный фактор. Посмотрев на экран 12, легко можно узнать, что новые имена ИС теперь запишутся на экран 28. В рассмотренном примере все корректируемые переменные лежат на одной строке на фиксированных местах, что предельно упрощает программу.

В данной программе выборка задания осуществляется через меню. Ниже приведен пример выдачи данных на экран после запроса <наличие>:

```
155ТМ7              SN7475  4-BIT LATCH, BIPOLAR OUT
УХОД  НАЛИЧИЕ  ПАРАМЕТРЫ  ТАБ.ИСТИН. СХЕМА  ЦОКОЛЬ  РЕДАКТОР
ИМЕЕМ      = 56
ЗАЯВКА     = 100
ПОЛУЧЕНО   = 50
ВЫДАНО     = 15
ГОД НАЗАД  = 75
2 Г. НАЗАД= 48
```

При этом курсор указывает на первую букву слова <УХОД>. Используя клавиши <→> и <←>, можно установить курсор на одно из слов этой строки. Положение курсора определяется переменной CURSOR, равной номеру слова, на которое указывает курсор, умноженному на 2. Слову <УХОД> соответстует CURSOR @ =0, а слову <РЕДАКТОР> – 12. Для реализации меню опишем слово TASK:

```
: TASK OUT HAVE PARAM TRUTH SCHEME PINAGE EDIT ;
```

Предполагается, что все слова, входящие в TASK, описаны ранее. Теперь по команде › TASK CURSOR @ + @ EXECUTE управление будет передано процедуре, на имя которой указывает курсор.

Ниже приведен фрагмент текста, содержащего начало списка заголовков описаний ИС:

```
: ALF ; ' ALF $Z !
0 $ 155ЛА3 76 $ 155ЛА8 152 $ 155ЛЕ1 228 $ 155ТМ7 304 $ 155ЛЕ
3 380 $ 155ЛП5 456 $ 155ЛА1 532 $ 155ЛИ1 608 $ 155ЛА13 684 $
155ЛА11 760 $ 155ТА3 836 $ 155ЛА12 912 $ 155ЛА9 988 $ 155АГ1
```

записанных самой программой-справочником. Именно поэтому между описаниями равные интервалы и переносы со строки на строку выполнены чисто функционально. Заголовок представляет собой имя переменной, которое соответствует типу ИС, а значение определяет номер экрана и номер байта на этом экране, с которого начинается описание для данной ИС. При загрузке программы-справочника экраны с заголовками также загружаются. Когда при запросе печатается имя ИС, то в стек записывается код, характеризующий место, где находится информация о данной ИС, извлечь же эти данные уже нетрудно.

Этот принцип можно реализовать и в упрощенных справочниках, например, для поиска зарубежных аналогов отечественных ИС. Достаточно сформировать описания слов вида

```
: 155ТМ7 ." SN7475 " ;        : 155ЛА3 ." SN7400 " ; и т.д.
```

и поместить их на экраны N, N+1 и т.д. Теперь, если загрузить такую цепочку экранов и напечатать, например, 155ЛА3 <ВК>, ЭВМ выдаст SN7400 ОК. Если в начале 1-го экрана (N) поместить пустое слово :ALFA;, то по завершении работы с таблицей аналогов достаточно напечатать FORGET ALFA и таблица будет удалена из словаря. Можно также описать слова : ANALOG N LOAD ; и : END FORGET ALFA ;, первое из которых должно быть загружено заранее. Тогда, набрав на пульте ЭВМ ANALOG <ВК>, вы загрузите таблицу аналогов, а дав команду END <ВК>, удалите ее из словаря. Таблица аналогов может быть и более сложной, в этом случае желательно написать управляющую программу, которая будет формировать ее текст. Ниже рассмотрен упрощенный вариант такой программы.

Введем переменную, которая является указателем в пределах экранного буфера: 0 VARIABLE POINT. Опишем также слово ENTER, которое переносит введенный с терминала текст в нужное место экранного буфера, указанное переменной POINT.

```
65 VARIABLE BLOK                          ( работа на экране 65)
: BLO BLOK @ BLOCK ;
: ENTER QUERY     ( ввод текста с терминала во входной буфер)
        BL WORD ( выделение слова и пересылка его в область,
                  указанную переменной HERE)
        HERE COUNT          ( в стеке HERE+1  и число введенных
                              символов)
```

```
        >R BLO POINT @ +   ( вычисление адреса начала записи)
        R CMOVE           ( запись введенного текста в буфер)
        R> POINT +! ;                  ( коррекция указателя)
: BEGI                   ( оформление начала нового описания)
        '20072           ( код ": ")
        BLO DUP 1024 +       ( в стеке  адреса начала и конца
                               экранного буфера)
        >R POINT @ +         ( в стеке  текущий адрес буфера)
        DUP R> OVER -      ( в стеке  ADR ADR и число байт до
                  конца буфера. ADR - текущий адрес буфера)
        BLANKS           ( заполнение свободной части буфера
                           пробелами)
        ! 2 POINT +! ;             ( запись кода ": " в буфер и
                                     коррекция указателя)
'27040 VARIABLE ANA '20042 , '21040 VARIABLE FIN '35440 ,
( массив ANA содержит последовательность символов < ." >, а
FIN - последовательность < " ;>)
: ANALOG ANA BLO POINT @ + 4 CMOVE                ( ввод < ." > в
                                                  в новое описание)
         4 POINT +! ;          ( коррекция указателя массива)
: $LOAD  BEGI ." IC>" ENTER    ( оформление начала описания и
                               ввод имени ИС-оригинала)
        ANALOG ." ANALOG>" ENTER        ( ввод имени аналога)
        FIN BLO POINT @ + 4 CMOVE     ( оформление завершения
                                      описания - засылка < " ;>)
        6 POINT +! UPDATE FLUSH ;       ( коррекция указателя,
                                      запись результата на диск)
```

Оператор $LOAD делает запрос на ввод, запись полученной информации, оформление описаний и запись результатов на диск. При обращении к $LOAD, если на запрос IC> ответите 155ИМ1, а на ANALOG – SN7480, то на строке 1 экрана 65 обнаружите 155ИМ1 ." SN7480 " ; . При повторном обращении к $LOAD и вводе данных о другой ИС, на экране 65 вслед за первой появится следующее аналогичное описание. Это только модель. Реальная программа $LOAD должна быть дополнена циклом для ввода последовательности описаний, а также командами контроля значения переменной POINT и оформления перехода на следующий экран (ввод последовательности символов –-> в конце текста экрана). При заполнении очередного экрана должна выполняться процедура 1 BLOK +! и корректироваться код 65 на экране, где записана эта программа (описание переменной BLOK). Последовательность операторов UPDATE FLUSH следует выполнять только после заполнения экрана или завершении работы.

При желании программу можно легко дополнить операторами, которые будут загружать и удалять из словаря описания именно тех серий ИС, какие вас интересуют. Можете выполнить эту работу сами.

4.2. ОТОБРАЖЕНИЕ ТЕКСТОВ ПРОГРАММ И ВЫВОД ДАННЫХ НА ПЕЧАТЬ

Для работы со словарем из-за особенностей виртуального файла FORTH.DAT необходимо иметь под рукой средства, которые бы позволяли легко ориентироваться в среде Форта. Дело в том, что в виртуаль-

ном файле FORTH.DAT может содержаться большое число экранов с разнообразными библиотеками. Найти нужный текст иногда непросто.

Можно в начале диска, например на экране 2, с помощью редактора создать оглавление этого файла. Но, если его нет, к вашим услугам оператор INDEX (табл.20).

Таблица 20. Сервисные операторы

Имя	Состояние стека	Версия	Функция
VLIST WORDS	— → —	F 9, 3	Отображает полный список слов в текущем словаре, заданном переменной, CURRENT
INDEX	n1 n2 → —	9, 3, F	Выводит на экран верхние строки экранов начиная с n1 и кончая n2
THRU	n1 n2 → —	9, 3, MV	Последовательно загружает экраны начиная с n1 и кончая n2
HERE	— → адр	9, 3, F	Выдает адрес первой свободной ячейки в словаре
PAD	— → адр	9, 3, F	Выдает адрес буфера для промежуточного хранения последовательностей символов
COUNT	адр → адр+1 u	9, 3, F	Преобразует последовательность символов, чья длина записана в первом байте, в форму, приемлемую для TYPE. При этом в стеке остается адрес первого символа и число символов в последовательности
TEXT	s → —		Читает последовательность символов из входного потока, используя символ s в качестве разграничителя, затем заполняет PAD кодами пробелов и переносит в PAD входную строку
>BINARY или CONVERT	d1 адр1 → d2 адр2	9, 3	Преобразует текст начиная с адр1+1 в двоичную форму с учетом BASE. Результат записывается в стек в виде числа двойной длины. адр2 — адрес первого преобразуемого символа

Окончание табл. 20

Имя	Состояние стека	Версия	Функция
–TEXT	' адр1 u адр2 →f		Сравнивает две строки начиная с адр1 и адр2. Длины строк =u. f=TRUE, если строки не совпали
DUMP	адр u → –	9, 3, M	Отображает u байт памяти начиная с адреса "адр"

Примечание. См. примечание к табл.19. MV – MVPFORTH, M – MMSFORTH.

При обращении N1 N2 INDEX этот оператор напечатает первые строки экранов начиная с N1 и кончая N2 включительно. INDEX будет эффективен, если на первых строках экранов помещена специальная справочная информация или хранящийся там текст не оставляет сомнения о характере информации, записанной на экране. Понятно, что пустая (заполненная пробелами) первая строка – наихудший вариант, желательно помечать пустые экраны специальными признаками, например ;S <EMPTY>, или ;S ПУСТО, или ." EMPTY". Оператор ;S избавляет вас от комментариев интерпретатора, если вы по ошибке загрузите такой экран, последний из предлагаемых вариантов выдаст сообщение EMPTY OK. Если по каким-либо причинам ваша версия базового словаря не имеет оператора INDEX (указатель), можно его описать и включить в текст экрана, содержащего служебные операторы:

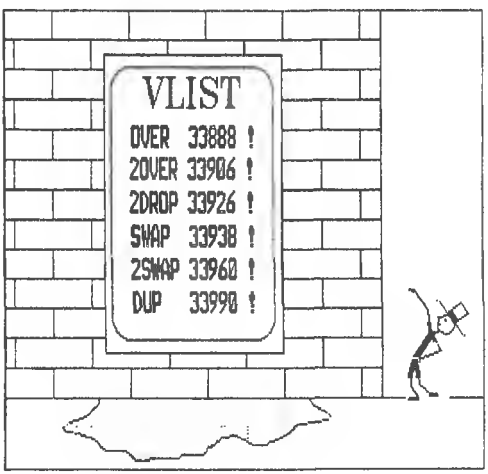
```
: INDEX 1+ SWAP DO CR I 3 .R SPACE 0  ( указание на нулевую
                                        строку)
        I .LINE LOOP ;    (см. табл. 20 )
```

При распределении информации на экранах следует придерживаться определенных правил. Приведенные ниже соображения относятся прежде всего к версии FIG-FORTH. Экран 1 полезно сохранить для программ стартовой загрузки, экраны 4 и 5 заняты в FIG-FORTH текстами системных сообщений, которые можно при необходимости расширять и дополнять (используются системой Форт, если системная переменная WARNING=1). Экран 3 удобно использовать для "быстрой" загрузки двоичного образа редактора, ассемблера или других часто используемых библиотек.

Для просмотра полного списка операторов, хранящихся в данный момент в словаре, предназначено слово VLIST (весь список):

```
: VLIST CONTEXT @ @  ( запись в стек адреса начала просмотра
                       словаря)
        BEGIN CR      ( начало бесконечного цикла просмотра)
          3 0 DO      ( информация о трех операторах помеща-
                        ется в одну строку)
                DUP ID.  ( печать имени очередного оператора)
                13 OVER C@ '37 AND  ( вычисление длины имени
                                      очередного оператора)
                - SPACES             ( организация табуляции
                                       операторов словаря)
                PFA DUP 6 U.R SPACE  ( печать PFA оператора)
                '41 EMIT SPACE                ( печать "!")
                LFA @      ( запись в стек значения константы
                             связи)
                DUP 0=                        ( конец словаря?)
                IF LEAVE              ( если да, выход из цикла)
                THEN
              LOOP -DUP 0=             ( это действительно конец
                                         словаря?)
        UNTIL ;
```

VLIST печатает имя оператора и адрес его поля параметров (PFA), информация о других словах в строке отделяется восклицательными знаками. Перечень начинается с последнего загруженного в словарь слова и завершается первым оператором базового словаря. Таким образом, слова из словарей, которые не являются в данный момент контекстными, хотя и есть в памяти, в отображаемый список не попадут.



Рассмотрим, как устроен оператор ID. , печатающий имя слова, записанное в соответствии с требованиями словаря Форт:

```
: ID. COUNT                ( в стеке адрес первого байта имени)
      '37 AND                ( вычисление числа байт в имени)
      TYPE SPACE ;      ( распечатка имени с пробелом в конце)
```

При обращении к ID. в стеке должен находиться адрес первого байта описания слова (NFA). При выдаче списка операторов с помощью VLIST можно обнаружить одно безымянное слово – 0 (цифра 0). Это может вызвать недоумение: зачем в словаре слово, к которому нельзя обратиться? С клавиатуры такое имя ввести действительно нельзя (так как вводятся только коды ASCII). Слово 0 используется для завершения (прерывания) процедуры интерпретации-исполнения. Так как ввод текста осуществляется оператором EXPECT, то любая вводимая строка завершается двумя нулевыми байтами.

Экранные буферы также имеют в конце два нулевых байта, которые прерывают интерпретацию при загрузке экрана. Описание слова 0:

```
: 0 BLK @                                        ( экранный буфер?)
    IF                                             ( если да)
       1 BLK +! ( переключение указателя на следующий экран)
       0 IN !                  ( обнуление указателя смещения)
       ?EXEC                   ( система в режиме исполнения?)
    THEN R> DROP ;                                 ( уход в QUIT)
```

Здесь имя 0 записано условно, в отличие от других описаний слово 0 (так же как ":" или ";") нельзя непосредственно смоделировать в Форте, ведь в приведенном примере оператор имеет имя, код ASCII которого равен '60, а не 0. Оно формируется и присутствует только в базовом словаре интерпретатора.

При документировании текстов программ оказывается удобным размещать тексты экранов по три на странице. Такую процедуру реализует слово TRIO:

```
: TRIO 12 EMIT       ( прогон бумаги в печатающем устройстве
                     на начало новой страницы. В случае выдачи
                     на экран никакого действия не вызывает)
       3 OVER        ( в стеке N 3 N)
       + SWAP        ( в стеке N+3 N)
       DO I LIST     ( распечатка текста очередного экрана)
       LOOP ;
```

Обращение к этому слову имеет форму : N TRIO, где N – номер первого экрана в этой триаде. При реализации программы, так же как и в INDEX, применен полезный прием. Вместо обычного цикла для трех экранов 3 0 DO...LOOP, организован цикл N+3 N DO...LOOP. Это сократило и ускорило выполнение программы, в чем читатель может убедиться сам. В версии FIG-FORTH процедура TRIO присутствует в базовом словаре. Приведенная программа выдает тексты на экран по умолчанию, что удобно для беглого просмотра.

Для управления выводом информации на печатающее устройство введена системная переменная PNT. Если PNT=0, информация выводится только на экран, при PNT≠0 – на экран и печатающее устройство

одновременно. Для переключения переменной PNT в ненулевое состояние используется оператор PRINT. Присвоение нулевого значения переменной PNT производится оператором TTY. Любые обмены, происходящие между PRINT и TTY, обязательно будут посланы и на печатающее устройство. Программист должен только позаботиться, чтобы оно было включено. Если вы рассеяны, ЭВМ напомнит вам о ваших обязанностях, но включить принтер на сможет.

Можно записать арифметические действия в привычной форме, например 2 * 2? Конечно, можно. Опишем сначала оператор CONVERT (версия FIG-FORTH):

```
: CONVERT BL WORD HERE NUMBER DROP ;
```

В некоторых версиях Форта операторы HERE и DROP будут лишними. Теперь переопределим основные арифметические операции:

```
: + CONVERT + ;
: - CONVERT - ;
: * CONVERT * ;
: / CONVERT / ;
```

При загрузке этой программы вы получите букет предупреждений о повторном описании уже известных операторов, но зато в пультовом режиме сможете наслаждаться привычной формой записи арифметических операций. Аналогично можно переопределить операторы LIST, LOAD и т.д. Однако, чтобы использовать традиционную форму записи и при описании слов, нужны более серьезные ухищрения, возможны и более эффективные реализации. Но это хорошо лишь в качестве упражнения. Форт эффективен именно благодаря своим особенностям и, если вы хотите воспользоваться его достоинствами, надо расстаться с некоторыми привычками.

4.3. РЕКУРСИЯ

К числу достоинств Форта относится и возможность рекурсии – разрешение процедуре обращаться к самой себе в процессе исполнения. Но если написать : AGAIN 1+ AGAIN ; , интерпретатор напечатает AGAIN?. А это означает, что он с этим оператором еще незнаком (ведь интерпретация слова AGAIN не завершена). Чтобы обойти эту трудность, воспользуемся оператором RECUR:

```
: RECUR LATEST   ( определение адреса поля имени (NFA) для
                 слова, в котором встретится оператор RECUR)
        PFA CFA  ( определение адреса поля команды этого
                   слова)
        , ;      ( занесение вычисленного адреса в поле па-
                   раметров на место слова RECUR и обеспе-
                   чение обращения оператора к самому себе)
        IMMEDIATE  ( оператор RECUR работает в режиме ин-
                     терпретации)
```

Позаботимся и о том, чтобы своевременно выйти из рекурсивного процесса, ведь рекурсия – еще один вид бесконечного цикла:

```
: LEV R> DROP ;
: TT DUP 1+ DUP . DUP 10 = IF LEV THEN RECUR ;
```

При обращении к рекурсивной процедуре TT: 0 TT <ВК> получим

1 2 3 4 5 6 7 8 9 10 ОК

Часть программы 10=IF LEV THEN – процедура контроля состояния процесса и выхода из цикла при выполнении определенного условия (содержимое стека 10). Существуют и другие способы выхода из рекурсивного цикла, например ту же процедуру можно переписать в виде

: TT DUP 1+ DUP . DUP 10 < IF RECUR THEN ;

Здесь управление передается на рекурсивный вызов только при определенном условии (содержимое стека <10).

Пусть описаны слова A и B, причем в слове A есть ссылка на слово B, а в слове B – на слово A. Это простейший пример косвенной рекурсии. Непосредственно смоделировать эту ситуацию с помощью системы Форт невозможно, так как при описании A будет выдано сообщение об ошибке, ведь оператор B еще не описан. Как быть, если все же очень нужно (или очень хочется) написать такую программу? Обманем интерпретатор. Опишем сначала пустой оператор : DUMMY 0 ; , а затем по порядку:

```
: B DUMMY ;
: A DUP 1+ DUP . DUP 10 < IF B THEN ;
```

и, наконец, выполним команду ' A CFA ' DUMMY !, которая подменит в операторе DUMMY ссылку на 0 ссылкой на A. Обратившись 0 A <ВК>, получим распечатку, идентичную приведенной для прямой рекурсии. Разумеется, между именем B и DUMMY может находиться текст некоторой программы. Используя описанную технику приведения косвенной рекурсии к прямой, можно написать довольно изощренную рекурсивную программу.

Эта реализация косвенной рекурсии не является полным аналогом прямой рекурсии, здесь при каждом цикле в стек возвратов заносится три кода, что при большом числе итераций может привести к переполнению стека возвратов. Поэтому уход из цикла, как в первом варианте прямой рекурсии, невозможен. Выполнив команду ' A CFA ' B !, можно сделать цикл рекурсии короче (без обращения к DUMMY), но тогда нельзя помещать какие-либо операторы между именем и DUMMY в описании B.

4.4. ВЫБОР ЗАДАНИЯ ИЗ "ДЛИННОГО" СПИСКА (ИСПОЛНИТЕЛЬНЫЕ ВЕКТОРЫ)

Выше был приведен случай выбора и исполнения задания, на имя которого указывает курсор. Но сходные проблемы возникают в самых разных задачах. Из них наиболее распространенная – это распознавание последовательности кодов, посылаемых при нажатии функциональной клавиши, и исполнение задания соответствующего этой клавише. Примером такой задачи является уже описанный экранный редактор (см. также приложение 1). Рассмотрим текст операторов-анализаторов управляющих кодов (см. также описание оператора EDT гл.4 ч.I). Программа предназначена для распознавания ESC-последовательностей различной длины, одиночных управляющих кодов, а также кодов ASCII и запрещенных кодов.

```
: TT 0      ( в стеке код символа и адрес списка управляющих
              кодов)
        DO 2DUP I + C@ =
          IF ( символ узнан) 2DROP LEAVE I 12 - ( кодирова-
                                                  ние номера символа)
          THEN
        LOOP DUP 0<
        IF 12 + 2* + @ EXECUTE          ( вычисление адреса и
                        передача управления узнанной процедуре)
        ELSE DROP DUP 31 >
             IF ( ввод символа в текст буфера) SS
             ELSE DROP IL  ( символ не узнан, дается звуковой
                            сигнал)
             THEN
        THEN ;

( Операторы-списки процедур)

: T3    LINE .T EL WST ++ CUT DS BUP SEE PAGE DW IL ;

: .P    ' T3                            ( запись в стек PFA T3)
        KEY ( ожидание нажатия клавиши) L1    ( запись в стек
                                  PFA списка управляющих кодов)
        14 + 12 TT ;

: T2    ^ V << >> HP GLD DLL .P ;     ( .P - процедура анализа
                    очередного символа ESC-последовательности)

: P.    ' T2                             ( запись в стек PFA T2)
        KEY ( ожидание нажатия клавиши) L1 6 + 8 TT ;

: T1    P. ^L DEL ^W DEL ;
```

Рассмотрим задачу в несколько более обобщенном виде. Пусть описаны процедуры A, B, C,...,K и пусть в каком-то месте программы вычисляется число N ($0 \le N \le 10$), которое определяет тип процедуры, подлежащей исполнению. Предполагается, что каждой процедуре из списка A, B, C,...,K поставлено в соответствие число 0, 1, 2,...,10. Выбрать нужную процедуру можно следующим образом. Опишем слово : TASK A B C D E F G H I K ; . Доступ к нужному оператору осуществит команда N 2* › TASK + @ EXECUTE. › TASK задает адрес поля парамет-

ров слова TASK, который указывает на код, равный CFA слова A. Если реализовать аналогичную функцию, используя IF...ELSE...THEN, запись окажется более длинной. Причем с увеличением списка процедур длина такой программы будет расти квадратично. Например:

```
: TTT -DUP IF DUP 1 = IF DROP B ELSE DUP 2 = IF DROP C ELSE
     DUP 3 = IF DROP D ELSE DUP 4 = IF DROP E ELSE ......
     ..... THEN THEN ..... ELSE A THEN ;     ( текст с целью
                                               экономии места сокращен)
```

Здесь еще нет проверки того, что N находится в нужном интервале значений.

Это решение приемлемо, если N≤4, когда значения N не являются числами из натурального ряда. Например, при N=26 должна выполняться процедура C, при N=27 – процедура A, при N=33 – процедура B и т.д. При длине списка заданий N>5 вычисляемый оператор EXECUTE становится предпочтительнее. При числе альтернатив более 10 даже при значениях N не из натурального ряда вычисляемый EXECUTE остается единственно возможным. В последнем случае надо только сначала преобразовать N в число из натурального ряда. Такое преобразование может быть выполнено различными методами, например, определим слова

```
N1 VARIABLE NN N2 , N3 , ... NM,          ( описан массив NN
                                            из M чисел)
```

N1, N2, ...NM не являются числами натурального ряда, т.е. $N_{i+1}-N_i \neq 1$.

```
: FIND M 0 DO DUP NN I 2* + @ = IF DROP I LEAVE THEN LOOP ;
```

В результате выполнения команды N FIND в стек будет записано число из натурального ряда 0≤K≤M−1, которое и будет использовано в вычисляемом EXECUTE.

4.5. ПОИСКОВО-ИНФОРМАЦИОННЫЕ ПРОГРАММЫ

Применение ЭВМ для управления и контроля не обходится без справочно-информационных программ. Это могут быть базы данных, программы типа HELP, которыми снабжаются практически все современные пакеты прикладных программ, или более простые чисто справочные программы.

Обычно данные хранятся на диске в рамках того или иного экрана или группы экранов. Простейшая информационная система может быть построена следующим образом. Пусть на экране N хранится текст типа

```
." ВОКЗАЛЫ "    : ВОКЗАЛЫ K0 LOAD ;  CR
." ДРУЗЬЯ "     : ДРУЗЬЯ  K1 LOAD ;  CR
." АВАРИЯ "     : АВАРИЯ  K2 LOAD ;  CR  и т.д.
```

Последнее описание может содержать телефоны аварийных служб, скорой помощи, лечащего врача, милиции и пр. При загрузке экрана N будет распечатана левая колонка заголовков-рубрик, сообщая о том, какие информационные операторы размещены на данном экране. Выбрав рубрику и напечатав, например, слово "ДРУЗЬЯ", мы загрузим экран К1 (в общем случае это будет цепочка экранов), который может содержать новый аналогичный список рубрик или непосредственно справочную информацию. Если у вас много друзей, и вы хотите записать в файл их дни рождений и другие семейные даты, а также любую другую информацию, то это может не поместиться на одном экране, например:

```
: ВАСЯ_СМИРНОВ ." дом. 319-24-22. раб. 122-00-02 Род. 7 июля
1938." CR ." Дочери: Наташа род 9 нояб. и Татьяна род. 10
дек." CR ." жена Лариса род 20 янв. Свадьба 14 авг. и т.д."
CR ; -->
```

--> после последней записи на экране автоматически обеспечит ввод следующего по порядку экрана. Если информация о друзьях записана не подряд, то вместо --> следует записать A LOAD, где A – номер экрана с нужной информацией. Чтобы текст оставался на экране и не вытеснялся последующими записями, в начале очередного экрана полезно поместить команду KEY DROP. Это потребует нажатия любой клавиши для просмотра очередной порции информации. DROP нужно для стирания кода клавиши из стека. Когда вы убедились, что экран с интересующими вас данными загружен, вы печатаете, например, "ВАСЯ_СМИРНОВ" <ВК> и получаете на экране весь приведенный выше текст.

Описанная структура справочной системы доступна для начинающих. Это, конечно, не база данных, но для личного телефонного справочника она подходит вполне.

Современное программирование развивается в направлении, при котором для освоения того или иного пакета программ требуется минимальное знакомство с его описанием или его не требуется вовсе. Достигается это за счет встраивания справочных систем HELP (см. приложение 5), которые по запросу могут предоставить пользователю исчерпывающую информацию о тех или иных функциях системы или подсказать возможные действия в сложившихся обстоятельствах. Обычно справочные системы имеют иерархическую структуру. При ее вызове сначала выявляется тема запроса, а затем уточняются разделы и подуровни. Ниже будут рассмотрены некоторые возможности формирования таких справочных систем на Форте.

Вызов программы осуществляется командой HELP (или нажатием соответствующей клавиши), загружающей нужный стартовый экран (или последовательность экранов) и в конце концов выдающей меню тем, по которым имеется справочный материал. Возможны варианты, когда пользователю предлагается самому напечатать тему запроса. При ошибке ввода или при отсутствии информации по затребованной теме пользователь получит соответствующее уведомление. Как правило, при таком способе нужную информацию можно получить после ряда проб и ошибок. Через меню дорога к цели короче, но это возможно только при малых объемах справочных данных.

Программа HELP, приведенная в приложении 5, предназначена для работы на терминале СМ7209 или на совместимом с ним по клавиатуре и управляющим последовательностям. Это влияет на технику фиксации и перемещения курсора по экрану, а также на способ переключения с русского алфавита на латинский и обратно. Загрузка стартового экрана может проводиться различными способами (см. гл.4 ч.I). При загрузке в верхней части экрана появится текст меню в виде, представленном в конце программы HELP. Курсор устанавливается перед словом "УХОД". Используя клавиши <→>, <←>, <↑>, <↓>, можно установить курсор и перед названием нужной вам темы и нажать клавишу <HELP>. Выполнение команд здесь проще, чем в экранном редакторе EDT. По положению курсора определяется номер числа в массиве TS (Text Screens) и загружается экран, на который указывает это число. Количество элементов в массиве TS определяет число входов в справочную систему (в данном случае 11, что соответствует числу тем в меню). Как видно, при такой системе адресации к справочному материалу не требуется, чтобы экраны с информацией были расположены подряд. Операторы, связанные с перемещением курсора (FIX, HT, EX), идентичны или сходны с теми, что использованы в экранном редакторе EDT. Слова RU и LA служат для переключения дисплея на русский и латинский алфавиты соответственно.

Справочный текст экрана 21 при выводе на экран выглядит куда пригляднее. Главной заботой при размещении его на диске была экономия места. Когда полный текст-комментарий на экране не помещается, с помощью оператора – –> вызывается следующий экран. Количество вызываемых друг за другом экранов не ограничено. При загрузке справочных экранов словарь Форта остается без изменений. Когда текст-комментарий отображен на экране полностью, курсор возвращается в меню и устанавливается снова перед словом "УХОД". Если при этом нажать клавишу <HELP>, управление будет передано в систему Форт, но часть словаря, созданная при загрузке справочной

системы, будет сохранена и вы можете ее снова вызвать командой HELP <ВК>. Если хотите удалить из словаря справочную систему, выдайте команду FORGET $ <ВК>. $ – первое слово, описанное на экране справочника. При необходимости создания многоуровневой системы запросов можно при загрузке очередного экрана сменить меню, а по завершении работы на данном справочном уровне восстановить старое меню, соответствующее более высокому уровню.

4.6. РАБОТА С ФАЙЛАМИ

До сих пор речь шла в основном о работе с файлом FORTH.DAT. Во многих случаях этого файла вполне достаточно. Для работы с произвольными файлами возможны различные решения. Все, что связано с файлами, к сожалению, специфично для каждой конкретной реализации Форт-интерпретатора. Поэтому прежде чем приступить к работе, проконсультируйтесь с представителем фирмы, откуда вы получили систему Форт, и запаситесь соответствующими описаниями и руководствами.

Рассмотрим вариант для операционной системы RT-11 (TSX). Идея этого варианта проста и основана на изменении имени виртуального файла в программе-драйвере диска. Для реализации этих функций введены операторы OFILE, CLOSE, DEFOL, FILCH и CHFILE:

OFILE – открытие нового виртуального файла; перед использованием OFILE имя файла должно быть изменено с помощью оператора CHFILE.

CHFILE – замена имени виртуального файла, например, RK:FORTH.DAT, на новое, адрес первого байта которого хранится в стеке (команда базового словаря Форта (RT-11)). Имя файла должно быть записано в кодах RAD50 (для персональных ЭВМ в кодах ASCII), в этом случае имя занимает четыре ячейки. Устанавливает каналы обмена с диском в исходное состояние. Если необходимо что-то сохранить на диске, это следует сделать до выдачи команды CHFILE.

CLOSE – закрытие файла, открытого ранее с помощью OFILE (EDT, LIST и т.д.), на диске возникает новый файл с тем что было в него записано между командами OFILE и CLOSE.

Для записи в новый файл экранов из других виртуальных файлов можно использовать оператор FILCH. Текст оператора FILCH, а также ряда других вспомогательных слов представлен ниже:

```
: RAD5 DUP 32 =  ( преобразование кода ASCII, хранящегося в
                   стеке, в формат RAD50)
        IF DROP 0                    ( если в стеке код пробела)
        ELSE DUP 36 =                      ( если это код "$")
```

```
            IF DROP 27
            ELSE DUP 45 > OVER 58 < AND
                 IF 18 -                        ( если код цифры)
                 ELSE DUP 64 > OVER 91 < AND
                      IF 64 -                   ( если код буквы)
                      ELSE DROP 0          ( если код не узнан)
                      THEN
                 THEN
            THEN
       THEN ;

: R50 DUP C@ RAD5 40 * SWAP 1+ DUP C@ RAD5 ROT + 40 * SWAP
      1+ C@ RAD5 + ; ( формирование словао в формате RAD50)

: NUM HERE NUMBER DROP ; ( преобразование строки символов в
                           число)
```

См. Приложение 7

```
: ADR HERE 1+ PAD ;                          : MOV HERE C@ CMOVE ;

: RCON PAD CFA 12 0 DO 2+ PAD I + R50 OVER ! 3 +LOOP ;

: FILCH PAD 20 BLANKS      ( заполнение буфера PAD пробелами)
        ." FILE=" QUERY      ( ввод имени файла с клавиатуры)
        '72 WORD ADR MOV '56 WORD ADR 3 + MOV 32 WORD ADR 9
        + MOV                ( передача имени файла, удаление
                               разделительных кодов ":" и ".")
        RCON     ( преобразование имени файла в RAD50-формат)
        PAD ( имя в буфере PAD) CHFILE ;      ( запись имени)
```

Оператор RAD5 преобразует код ASCII в код RAD50, а R50 объединяет три кода RAD50 в одно слово. Операторы ADR и MOV чисто вспомогательные, способствуют более компактной записи программы, их функции очевидны из описания. Оператор RCON преобразует последовательность из 12 кодов ASCII в последовательность из четырех слов в кодах RAD50. Исходная и конечная последовательности кодов хранится в буфере PAD. Слово NUM преобразует последовательность символов, записанную, начиная с HERE, в число одинарной длины.

Оператор FILCH работает в диалоговом режиме. Если нужно изменять имя файла без вмешательства оператора, это имя в соответствующем формате должно быть записано в массив, например NEWFILE, а команда NEWFILE CHFILE выполнит работу, эквивалентную FILCH.

Копирование экранов в пределах одного файла осуществляется оператором COPY (см. приложение 1). Обращение: M N COPY, при этом экран M записывается на место N. Содержимое экрана N теряется. Для перемены мест экранов M и N может служить оператор SWAS, обращение к которому аналогично: M N SWAS.

```
: SWAS >R                          ( запись N в стек возвратов)
      BLOCK              ( вызов в буфер информации, записанной
                           на экране M)
      CFA DUP @            запись в стек параметров адреса
                         заголовка экрана и его содержимого)
```

```
I BLOCK          ( вызов в "соседний" буфер экрана N)
CFA !                    ( присваивание ему номера M)
UPDATE                 ( установка флага "спасения")
R> '100000 OR        ( операция UPDATE  для блока N)
SWAP !                ( запись заголовка в буфере N)
FLUSH ;    ( запись содержимого обоих экранов на диск)
```

Теперь рассмотрим возможность обмена экранами между двумя виртуальными файлами (оператор SCOP):

```
: SCOP ." FROM " FILCH    ( запрос ввода имени файла и номера
                            экрана)
       32 WORD NUM      ( выделение кода, следующего за именем
                          файла и преобразование его в число)
       BLOCK CFA    ( считывание нужного экрана в буфер и за-
                    пись в стек адреса идентификатора буфера)
       ." TO " FILCH DROP 32 WORD
       NUM SWAP !            ( замена номера экрана N1 на N2)
       UPDATE          ( установка флага "спасения") FLUSH ;
```

Схема такого обмена проста: сначала в качестве имени виртуального файла записывается с помощью оператора FILCH имя файла-источника, считывается нужный экран в буфер, в заголовке блока записывается номер экрана-адресата и устанавливается флаг "спасения" (процедура UPDATE). Наконец, в качестве имени виртуального файла записывается имя файла-адресата и производится запись буфера на диск. Информация, которая хранилась там прежде, теряется. При обращении к слову SCOP на экране появляется запрос "FROM FILE=", на который надо ответить:

DXi:<NAME>.<EXT> N1

DXi – имя внешнего устройства, где лежит файл-источник (это может быть DK, DY, MX и т.д.), <NAME> и <EXT> – имя и расширение файла-источника (наличие "." и ":" обязательно, так как их коды используются в качестве разделителей оператором WORD в слове FILCH); N1 – номер экрана-источника. Формат ответа на запрос "TO FILE" идентичен, но имя внешнего устройства (D2), имя файла-адресата (<NAME2>.<EX2>) и номер экрана-адресата будут другими. По завершении обмена система Форт будет работать с виртуальным файлом D2.<NAME2>.<EX2>. Вы можете поменять их произвольно с помощью FILCH или на DK:FORTH.DAT с помощью оператора DEFOL.

Много неожиданных возможностей предоставляет версия Форта – FOREGROUND. Для работы с ней необходим FB-монитор. Версия FOREGROUND (RT=11) получается, если воспользоваться редактором связей LINK/FOREGROUND FORTH. Загрузка FOREGROUND производится командой FRUN FORTH из ОС RT-11. В общм случае для эффективной работы в режиме FOREGROUND надо произвести некоторые переделки в интерпретаторе, но для простых задач этого не требуется,

и вы это можете попробовать, если располагаете файлом FORTH. OBJ. Режим работы в FOREGROUND устраняет изоляцию Форта от операционной системы, так как в этом случае к вашим услугам весь сервис RT-11 в режиме BACKGROUND. Так, нажав ^B, перейдем в режим BACKGROUND, и используя команду DIR, можно просмотреть оглавление дисков. Нажмите ^F, и вы снова в Форте.

Интересные возможности при работе с файлами предоставляет оператор ASSIGN (OC). По приведенной ниже схеме можно работать с несколькими виртуальными файлами на разных дисках:

```
CLOSE                          ( закрывание рабочего файла)
^B                             ( переход в режим BACKGROUND)
ASS DX0: DK:          ( в DX0 находится еще один диск с файлом
                        FORTH.DAT, возможны и временные пере-
                        именования файлов)
^F                       ( возврат в режим FOREGROUND, здесь
                           также возможна замена имени
                           виртуального файла)

Работа с файлом на DX0.

CLOSE ^B
ASS DX1: DK:  ^F      ( возврат к работе со "старым" виртуальным файлом на DX1)
```

Здесь приведен лишь один вариант, режим FOREGROUND предоставляет много других возможностей. Ограничения здесь связаны только с памятью, оставшейся для задач BACKGROUND (RT-11).

Как всякая программа в FOREGROUND, Форт загружается непосредственно перед FB-монитором в "верхнюю" часть памяти. (С картой абсолютных адресов загрузки FOREGROUND можно познакомиться в [28].) Главное преимущество описанного режима заключается в том, что можно уходить в RT-11 и при возврате не тратить время на загрузку системы Форт.

Упражнение 1. Опишите оператор, который заносит в первые строки последовательности экранов, начинающиеся с экрана N, записи ;S <EMPTY>. Обращение к оператору: N L EMPTY, где L — число экранов, куда заносится указанная строка.

Упражнение 2. Напишите программу, которая распечатает содержимое всего файла FORTH.DAT. Полное число экранов в файле предполагается известным. Составьте также программу, выдающую текст любого заданного числа экранов начиная с N.

Упражнение 3. Опишите оператор N!, который вычисляет факториал числа, находящегося в стеке, используя рекурсию.

Важным средством усиления мощности Форта является Форт-ассемблер, который обычно поставляется в виде библиотеки, вводимой в словарь стартовым загрузчиком или самим пользователем. В отличие от традиционных ассемблеров, где сначала пишется программа, затем проводится трансляция и редактирование связей (линкирование) и наконец загрузка и исполнение программы, в Форт-ассемблере осталась только интерпретация, совмещенная с загрузкой. В результате в словаре появляются описания процедур, которые по своей структуре идентичны примитивам системы Форт.

Рассмотрим для примера примитивы D+ (сложение чисел двойной длины) для ЭВМ типа PDP (СМ1420) и IBM PC (ЕС1841).

```
      PDP                          IBM PC

.BYTE     202          DB   202Q       ; байт длины имени
.ASCII    /D+/         DB   'D'        ; имя
.BYTE     240          DB   253Q       ; слова
.WORD     LINK         DW   LINK       ; ячейка связи (LFA)
.WORD     .+2          DW   $+2        ; CFA
ADD  (S)+, 2(S)        POP  AX
ADD  (S)+, 2(S)        POP  CX
ADC  (S)               MOV  SI,  SP    ; установка указателя
                                       ; стека

                       ADD SS:[SI+2], CX
                       ADC SS:[SI],   AX
MOV  (IP)+, WP         LODSW           ; NEXT (переход к
JMP  @(WP)+            MOV BX, AX      ; следующей
                       JMP [BX]        ; процедуре)
```

Если записать эту же программу на Форт-ассемблере (PDP):

```
CODE D+ S )+ 2 S I) ADD S )+ 2 S I) ADD S () ADC NEXT C ;
```

получим в словаре описание, тождественное приведенному в левой колонке.

Рассмотренный пример позволяет также представить главные ограничения быстродействия системы Форт. В процессе исполнения примитива как минимум две команды чисто служебные: они осуществляют переход к исполнению следующей процедуры (NEXT). Сократить их число можно только с помощью специальных Форт-процессоров [41, 42]. А так как Форт-программа – это, в конечном счете, цепочка примитивов, ее быстродействие ограничено именно этими издержками. Чем длиннее примитив, тем меньше непроизводительные потери памяти и времени исполнения. Но набор примитивов в базовом словаре фиксирован. Отчасти именно это и толкало некоторых разработчиков на переработку Форт-программ, связанную с удалением лишних связей.

Форт-ассемблер предоставляет компромиссный путь. У программиста появляется возможность написать критическую по времени исполнения программу в виде одного или нескольких "длинных" примитивов. При этом потери на процедуры CFA и NEXT уменьшатся, а удобство написания и отладки, которые предоставляет Форт, останется. Программист может не знать почти ничего об ЭВМ, на которой работает. Любой же ассемблер предполагает знание и умение использования машинных команд. В этом отношении Форт-ассемблер не составляет исключения. С помощью выбора мнемоники и описания часто встречающихся процедур можно лишь сгладить это неудобство. Машинозависимость ассемблеров делает их разнообразие неисчислимым. К счастью, одной из особенностей Форта является простота написания интерпретатора и создания Форт-ассемблера. Объем интерпретатора с Форт-ассемблера 40–60 строк текста.

Перед тем как приступить к написанию программ Форт-ассемблера, надо внимательно ознакомиться с набором команд и основными особенностями процессора, на котором он будет работать. В первую очередь, следует выяснить, имеет ли процессор аппаратный стек и сколько регистров общего назначения доступно программно. Важно также знать, как осуществляется доступ к оперативной памяти и внешним устройствам. Полезно иметь информацию о том, какие регистры и как используются в интерпретаторе. Надо выбрать регистры-указатели стека параметров (S) и возврата (RP), а также регистры-указатели инструкции (IP) и слова (WP). От правильного выбора их зависит эффективность программы, так как позволяет уменьшить число операций сохранения-восстановления содержимого регистров.

Как видно из структуры примитивов, назначение ассемблера – преобразование последовательности мнемонических кодов в последовательность машинных команд и запись ее в ячейки вслед за полем CFA. Завершается эта последовательность командами перехода к следующей процедуре.

5.1. ОПЕРАТОРЫ ФОРТ-АССЕМБЛЕРА

Список операторов Форт-ассемблера (здесь и далее ассемблер MACRO-11) представлен в табл.21. Этот список, так же как и в системе Форт, открытый. Пользователь может добавить в него любой оператор, если, например, приобретен процессор с расширенным набором команд. В рассмотренной реализации Форт-ассемблера имеются практически все команды ЭВМ, совместимой с машинами СМ1420. Описание оператора в Форт-ассемблере имеет следующий формат:

CODE <NAME> { <опер1> MODE <опер2> MODE <КОП> } NEXT C; где <NAME> – имя нового оператора; <опер1> и <опер2> – операнды 1 и 2 соответственно (число операндов зависит от типа команды); MODE – указатель типа адресации (может отсутствовать); <КОП> – код операции; NEXT C; – оператор перехода к следующей процедуре (в Форт-83 это оператор END-CODE). В области, выделенной фигурными скобками, может быть любое число команд. Когда мнемокоды Форт-ассемблера и Форта совпадают, в Форт-ассемблере в начале имени добавляется звездочка, хотя и необязательно (см. операторы *IF, *ELSE и т.д. в табл.21). Оператор Форт-ассемблера, содержащий 10 команд, выполняется на 20% дольше программы, написанной в машинных кодах. В Форт-ассемблере, так же как и в системе Форт, метки, как правило, не используются[1]. Это является причиной введения операторов циклов и условных переходов.

Операторы Форт-ассемблера в процессе интерпретации заносятся в словарь с именем ASSEMBLER и в дальнейшем могут употребляться совместно с другими словами при описании новых операторов Форта.

Таблица 21. Операторы (мнемокоды) Форт-ассемблера СМ ЭВМ

Имя (мнемокод)	Функция
CODE	Эквивалентно функции ":" системы Форт
#	Эквивалентно символу # в MACRO-11, но ставится не перед, а после числа и отделяется от него пробелом
)+	Соответствует форме (Rn)+ в MACRO-11, где Rn – имя регистра общего назначения
@+	Соответствует @(R)+ в MACRO-11
–)	Эквивалентно –(R)
@–)	Эквивалентно форме адресации @–(Rn)
1)	Соответствует форме X (Rn), где X – целое число
@1)	Эквивалентно форме адресации @X(Rn)
@#	Соответствует @# в MACRO-11, но ставится после числа или символического имени и отделяется от него пробелом
()	Эквивалентно форме адресации (Rn)
NEXT C;	Заменяет оператор ";" Форта
*IF	Аналоги соответствующих операторов Форта
*ELSE *BEGIN	
*WHILE *UNTIL	
*REPEAT	
*THEN	

[1] Хотя метки в Форте и можно ввести, но это, так же как и прямая нотация, лишит его некоторых преимуществ.

Форт-ассемблер пригоден для управляющих программ внешних устройств и всех случаев, когда время исполнения является важной характеристикой программы. Преимущество Форт-ассемблера перед обычным макроассемблером – возможность немедленного исполнения сразу после написания программы (ведь трансляция и редактирование связей не нужны). Примером программы на Форт-ассемблере может служить описание процедуры определения абсолютного значения числа, лежащего в стеке (AABS), результат так же, как и в случае ABS, записывается в стек:

```
CODE AABS S ( ) TST LT *IF S ( ) NEG * THEN NEXT C;
```

При обращении – 5 AABS . ЭВМ откликнется 5 OK.

Словарь ASSEMBLER является частью словаря системы Форт. Адрес программы CFA, выполняющей описанную процедуру, в системе Форт в случае Форт-ассемблера, как и в примитивах, всегда тождествен адресу PFA (поля параметров). В ячейках после CFA хранятся машинные команды, которые реализуют функции TST (S) и т.д. (S – указатель стека параметров). Последние две команды обеспечивают возврат к программе, откуда было обращение к описанной процедуре. Из приведенного примера видно, что в Форт-ассемблере также используется постфиксная нотация – мнемокод, указывающий на тип адресации, записывается после кода операнда и отделяется от него пробелом. Мнемокод операции (например, TST) пишется после операндов (а не перед ними, как в макроассемблере!) и также отделяется пробелом.

Приведем еще один пример оператора Форт-ассемблера:

```
OCTAL                ( установка восьмеричной системы счисления)
CODE AEMIT *BEGIN 200 # 177564 @# BIT NE *UNTIL   ( ожидание)
           S () 177566 @# MOV NEXT C;    DECIMAL
```

Оператор AEMIT выполняет операцию EMIT, описанную в Форте (работает для микроЭВМ типа ДВК). При обращении 87 AEMIT ЭВМ выдаст W OK. В данном примере слово NE совершенно необходимо: именно оно формирует соответствующим образом флаг условия для оператора *UNTIL. Таким образом, для всех случаев, где требуется проверка условия, необходимо использовать соответствующий оператор из табл.22.

Таблица 22. Операторы Форт-ассемблера СМ ЭВМ

Имя	Функция
EQ	Формирует флаг для условных операторов перехода типа *IF, и т.д. f=TRUE, если предшествующий результат равен 0

Окончание табл. 22

Имя	Функция
NE	То же, что и EQ, но f=TRUE, если предшествующий результат не равен 0
MI	То же, но f=TRUE, если предшествующий результат — число со знаком "минус"
PL	То же, но f=TRUE, если предшествующий результат — число со знаком "плюс"
LT	f=TRUE, если предшествующий результат меньше 0
GT	f=TRUE, если предшествующий результат больше 0
LE	f=TRUE, если результат больше или равен 0
GE	f=TRUE, если результат больше или равен 0
VS	f=TRUE, если бит переполнения 1
VC	f=TRUE, если бит переполнения 0
CS (LO)	f=TRUE, если бит переноса 1
CC (HIS)	f=TRUE, если бит переноса 0
HI	f=TRUE, если биты переноса (C) и нулевого результата (Z) равны 0
LOS	f=TRUE, если C=Z=1

Как же работает интерпретатор Форт-ассемблера? Обычно он пишется на Форте. Его текст начинается со слов VOCABULARY ASSEMBLER IMMEDIATE. Возможен вариант, когда оператор словаря не является словом немедленного исполнения, но это сделает некоторые последующие описания более громоздкими и менее читаемыми. Одним из главных операторов интерпретатора является CODE. Именно он вносит в словарь имя слова Форт-ассемблера. Его структура становится понятной из описания:

```
: CODE CREATE [COMPILE] ASSEMBLER ;
```

CODE, так же как и ASSEMBLER, является словом из словаря FORTH. Далее должны следовать слова-описания процедур Форт-ассемблера, поэтому здесь необходимо записать в текст интерпретатора команду ASSEMBLER DEFINITIONS.

Двумя другими универсальными операторами Форт-ассемблера являются NEXT и C; . Описание NEXT:

```
: NEXT IP )+ WP MOV WP @) + JMP ;
```

полностью соответствует программе в начале гл.5. Оператор C; комбинирует некоторые функции Форт-операторов ":" и ";" :

```
: C; CURRENT @ CONTEXT ! SMUG ;
```

CODE и C; могут включать сохранение-восстановление значения BASE и контроль неизменности указателя стека параметров.

Вы, возможно, уже обратили внимание на две сходные структуры : CODE <NAME> XXXXX NEXT C; и : <NAME> XXXXX ; (например, в только что описанной процедуре NEXT), XXXXX – последовательность мнемокодов Форт-ассемблера. Вторая структура, знакомая из базового Форта, может показаться идентичной первой, но это не так. Первая формирует описание оператора Форта, которое при обращении выполнит последовательность машинных команд, соответствующую мнемокодам, вторая начиная с первой свободной ячейки, указанной HERE, скомпилирует последовательность машинных кодов, которая соответствует тексту между <NAME> и ";". Другими словами, первая структура формирует исполняемую программу, а вторая – компилирующую. Таким образом, возможна комбинация:

```
: TEST *BEGIN R4 () TST PL *UNTIL ;
CODE PUT TEST R0 R5 () MOV NEXT C;
```

Оператор TEST скомпилирует необходимый нам кусок программы в слово PUT. Операторы типа TEST фактически выполняют функции макроопределений.

А зачем рядовому программисту знать, как работает интерпретатор или как его написать? Во-первых, интерпретатор Форт-ассемблера может отсутствовать в вашей библиотеке, а во-вторых, его очень легко написать. Здесь же поясним основные идеи. Предположим, что мы хотим описать команды, где в качестве операндов используются только содержимое регистров общего назначения. Опишем сначала эти регистры:

```
0 CONSTANT R0    1 CONSTANT R1    2 CONSTANT WP
3 CONSTANT R3    4 CONSTANT IP    5 CONSTANT S
6 CONSTANT RP
```

Введем слово WW:

```
OCTAL
: WW <BUILDS , DOES> @ ROT 100 * + + , ;
```

которое является описателем операций типа регистр-регистр. Следующим шагом должно быть описание команд, например: 10000 WW MOV, 20000 WW CMP, 30000 WW BIT и т.д. Теперь, если в описании какого-то слова встретится команда R1 R3 MOV, в соответствующую ячейку PFA этого слова будет занесен восьмеричный код 10103, который соответствует машинной команде, выполняющей операцию MOV для этих регистров. Конечно, WW – это нереальный оператор, здесь не учтено

все многообразие модификаций адресации, непосредственные операнды и т.д., но форма описания самих команд MOV, CMP и т.д. соответствует настоящему интерпретатору.

Для безоперандных команд типа CLC, HALT, NOP, RTI все гораздо проще. Достаточно ввести описание слова OP:

```
: OP <BUILDS , DOE> @ , ;
```

а вслед за ним описать все такие команды: 2 OP RTI, ›240 OP NOP, ›240 OP CLC и т.д.

Следующий по сложности уровень представляют команды ветвления. Для них нужно ввести описатель BRO:

```
OCTAL
: BRO <BUILDS , DOES> @ , HERE - DUP 376 > OVER -400 < OR
  IF ." ILL BRANCH" . THEN 2/ 377 AND HERE CFA SWAP OVER
  @ OR SWAP ! ;
```

и вслед за этим описать все команды ветвления: 400 BRO BR, 1000 BRO BNE, 1400 BRO BEQ и т.д.

Ознакомившись с этим, а также имея примеры из приложения 2 и 6, вы без труда сможете написать интерпретатор Форт-ассемблера для нового процессора. Не забудьте завершить свой текст интерпретатора командой FORTH DEFINITIONS DECIMAL.

5.2. РАБОТА С ФОРТ-АССЕМБЛЕРОМ

Рассмотрим некоторые примеры использования Форт-ассемблера. Предположим, мы хотим описать процедуру сложения для чисел с плавающей точкой на ЭВМ, где есть операция FADD (ДВК-4, СМ1420). Такое описание будет иметь вид

```
CODE F+ S FADD NEXT C;
```

F+ – имя этого слова. При обращении f1 f2 F+ , где f1 и f2 – числа с плавающей точкой, F+ удалит из стека четыре кода одинарной длины (f1 и f2) и на их место запишет два кода (f3), представляющие собой сумму f1+f2.

Другим примером может служить оператор FABS, который заменяет число с плавающей точкой, записанное в стек, его абсолютным значением:

```
CODE FABS ›100000 # S (f) BIC NEXT C;
```

Команда 2>R, описанная в приложении 2, пригодна как для чисел с плавающей точкой, так и для кодов двойной длины, она переносит

два верхних числа из стека параметров в стек возвратов. Описание этой процедуры в Форт-ассемблере:

CODE 2>R S)+ RP –) MOV S)+ RP –) MOV NEXT C;

Может показаться, что это эквивалентно : 2>R >R >R ; . Попробуйте и убедитесь в обратном. Весьма вероятно, что вам придется перезагрузить систему, так как изменение указателя стека возвратов в структурах : <name> ... ; ведет к тому, что оператор ;S, к которому произойдет обращение в конце исполнения, передаст управление процедуре, адрес которой хранится в стеке возвратов. А что там окажется в данном случае, предугадать весьма сложно. В этом еще одно отличие примитива от описания : <name> ... ;. Последние для перехода к следующей процедуре используют стек возвратов, а примитивы нет.

Примером условных переходов в Форт-ассемблере может служить слово F0> (библиотека для работы с числами с плавающей точкой):

```
ASSEMBLER DEFINITIONS OCTAL
CODE F0> 2 S I) CLR       ( обнуление младшей части мантиссы)
         S )+ TST  GT              ( формирование флага, если
                                     число меньше 0 )
         *IF S () INC        ( запись в стек 1, если число
                               больше 0)
         *THEN NEXT C;
FORTH DEFINITIONS DECIMAL
```

F0> проверяет число с плавающей точкой, находящееся в стеке: если оно больше нуля, заменяет его флагом =1, в противном случае нулем.

Более сложным примером использования Форт-ассемблера может служить процедура извлечения квадратного корня из числа с плавающей точкой (SQRT) в стеке (алгоритм Герона [37]):

```
: SQRT 2DUP 2/ '17600 AND '40000 OR
       5 0 DO ( начало итеративного процесса, число итераций
                равно 5)
       2OVER 2OVER F/ F+ 2E F/ LOOP ( конец итераций)
       2SWAP 2DROP ;
```

Операторы Форт-ассемблера F/ и F+ выполняют операции деления и сложения над числами с плавающей точкой; 2E – число 2.0 (с плавающей точкой).

Другое важное приложение Форт-ассемблера – написание программ прерывания, где время исполнения является определяющей характеристикой. Но об этом речь пойдет позже. Сейчас же посмотрим, как будет выглядеть примитив U<, написанный на Форт-ассемблере:

```
CODE U<  AX POP   CX POP    CX AX CMP   AB ( Формирование флага
                                          для IF)
         *IF     1 AX MOV                    ( флаг = TRUE )
         *ELSE   0 AX MOV                    ( флаг = FALSE)
         *THEN   AX PUSH NEXT C;          ( запись флага в стек)
```

AB – оператор "превышает" (ABOVE). Программа написана на ассемблере персонального компьютера типа IBM PC. Для ЭВМ СМ1420 эта же программа будет выглядеть так:

```
CODE U<   S )+ S ()   HI   *IF   1 # S () MOV
                           *ELSE S () CLR
                           *THEN NEXT C;
```

Следующую программу на Форт-ассемблере предлагаю написать самим. Пусть необходимо просуммировать числа натурального ряда от 1 до N, N в стеке. Составьте описание слова SUM, которое решает эту задачу и записывает в стек результат. Сделайте это сначала на Форте, а затем на Форт-ассемблере. Ответ для последнего варианта:

```
CODE SUM   S () R1 MOV              ( внесение N в счетчик R1)
           S -) CLR                 ( очищение "сумматора")
           *BEGIN   R1 S () ADD     ( суммирование )
           R1 DEC   LE   *UNTIL   NEXT C;
```

Теперь, если напечатать 10 SUM., ЭВМ откликнется 55 OK.

Как только Форт-ассемблер заработал, для вас стали доступны и программные запросы, выполняемые через операционную систему с помощью команды EMT (для ЭВМ СМ1420) или INT (для персональных ЭВМ).

А как быть, если нужно написать небольшую программу в машинных кодах, а Форт-ассемблера нет или его загрузка нежелательна? Это вполне возможно, хотя и трудоемко. Для решения задачи надо написать текст программы в машинных кодах, а затем записать

CODE <NAME> C1 , C2 ..., Cn , NEXT C;

где C1, C2,...,Cn – последовательность кодов, образующая программу. Запятые в описании слова <NAME> – это не знаки препинания, а операторы записи кодов C1, C2,... в описание. Если программа содержит команды-байты, то после них вместо запятой следует писать C, . Если использовать описания CODE и NEXT, то такая программа может быть реализована непосредственно в системе Форт.

Интересные возможности при написании программ на Форт-ассемблере предоставляет оператор ;CODE, который работает исключительно при компиляции и используется в структурах слов-описателей вида

: <name> XXX CREATE YYY ;CODE ZZZ NEXT C;

где XXX – обычная Форт-программа; YYY – часть программы, выполняемая при компиляции (см. гл.2); ZZZ – последовательность мнемо-

кодов Форт-ассемблера. Обращение к такому слову-описателю имеет форму:

<name> <ИМЯ>,

где <name> – имя слова-описателя; <ИМЯ> – имя сгенерированного нового слова, причем CFA слова <ИМЯ> будет содержать адрес начала программы, текст которой лежит между ;CODE и NEXT в слове-описателе. В сущности, вам предоставляется шанс создать принципиально новый вид слов, где, например, содержимое поля параметров будет обрабатываться по программе, составленной вами. Стандартные типы таких программ предоставляют операторы : <name> ...; , VARIABLE, CONSTANT. Аналогом ;CODE можно считать DOES>, где исполнительная программа написана на Форте, а не на ассемблере, как в ;CODE.

Упражнение 1. Опишите на ассемблере оператор –ROT (см. табл.2).

Решение. Для IBM PC

```
CODE -ROT  AX POP  CX POP  SI POP  AX PUSH  SI PUSH  CX PUSH
     NEXT END-CODE
```

Для СМ ЭВМ

```
CODE -ROT  R5 )+ R0 MOV  R5 )+ R1 MOV  R5 )+ R2 MOV
           R0 R5 -) MOV  R2 R5 -) MOV  R1 R5 -) MOV
           NEXT C;
```

Упражнение 2. Опишите слово 2*, используя Форт-ассемблер. Решение смотри в тексте интерпретатора Форта (приложение 9).

Упражнение 3. Опишите оператор для извлечения квадратного корня на Форт-ассемблере (см. SQRT в приложении 2). Напишите программу для оценки времени исполнения операторов и сравните их быстродействие. Для этого можно использовать внутренние часы ЭВМ. Здесь также вам пригодится ассемблер, так как одним из путей доступа к внутренним часам является применение команды EMT для СМ ЭВМ или INT для IBM PC.

Глава 6. Векторы, строки, массивы

Структуры, перечисленные в заголовке, наряду с переменнными и константами являются объектами, с которыми программист работает постоянно. Векторы и строки – это одномерные массивы чисел и символов соответственно. Несколько примеров простых одномерных массивов уже было описано. Это A0 VARIABLE A1 , A2 , A3 , A4 , или 0 VARIABLE AA NN ALLOT. В обоих случаях размер массива известен, в первом – это 10 байт, во втором – NN+2 байт. Векторы и строки представляют собой более организованные структуры. Примером описания вектора может служить следующее [35]:

```
: VECTOR <BUILDS 2* ALLOT DOES> SWAP 1- 2* + ;
```

Обращение к этому оператору на фазе описания нового вектора: N VECTOR <NAME>, где N – число 16-разрядных кодов в векторе (длина вектора), а <NAME> – имя нового вектора. Возможен и другой вариант:

```
: VECTOR <BUILDS DUP , 2* ALLOT DOES> SWAP 1+ 2* + ;
```

В этом случае обращение к описанному вектору будет иметь вид: IN <NAME>, в результате в стек будет записан адрес кода вектора с номером IN (нумерация начинается с 0). Второе описание отличается тем, что по адресу PFA+2 находится число, равное длине вектора. Это удобно, когда размер вектора вычисляется программой до его описания. В более сложном варианте реализации оператора [35] при обращении к вновь описанному вектору контролируется условие 0≤IN≤L, где L – длина вектора. Такое усовершенствование может быть легко добавлено ко второму описанию.

Примером строки может служить описание оператора STRING (строка):

```
: STRING                    ( описание структур типа "строка")
        <BUILDS             ( описание нового входа в словарь)
        HERE BL WORD ( введение в описание строки ее значе-
                       ния по умолчанию)
        C@ 1+ ALLOT ALIGN  ( выравнивание конца описания по
                             четному адресу, что необходимо
                             для ЭВМ типа СМ)
        DOES> COUNT ;
```

Пример использования этого оператора приведен в приложении 3, где STRING использован для описания строки с именем ALF. Эта строка содержит последовательность из 48 символов, которые образуют алфавит для плакатной печати. Каждая буква при этой печати отображается матрицей 5×7 элементов ■. Модели всех букв и символов представлены в массиве PAT. Приведенный текст программы ориентирован на операционную систему RT-11, в случае персональной ЭВМ типа IBM PC она несколько упростится.

Более сложную структуру имеют двумерные массивы, например ARRAY:

```
: ARRAY <BUILDS 2DUP , , * ALLOT DOES>
        SWAP OVER @ * + + 4 + ( в PFA+2 и PFA+4 записаны
        коды числа рядов [RM] и колонок в массиве [CM] ) ;
```

Массив CM занимает в поле PFA C*R+4 байт (элементом массива является байт). В этом описании также нет контроля номера строки R и колонки C, о чем можно сожалеть особенно при отладке программы.

При описании новых массивов обращение к ARRAY имеет вид: CM RM ARRAY <NAME>, например 8 8 ARRAY CHESS ("Шахматная доска"). Пусть элемент с C=0 и R=0 черного цвета. Цвет клетки определяется знаком байта, а наличие и тип фигуры – кодом, записанным в этот байт. Таким образом, чтобы определить цвет поля и тип фигуры на поле R=7 и C=5, надо выдать команду 5 7 CHESS C@. В результате в стек будет записан искомый код.

6.1. БИБЛИОТЕКА ОПЕРАТОРОВ ДЛЯ РАБОТЫ СО СТРОКАМИ

Характерная задача для редакторов текстов (CHI-WRITER, TEX, VENTURE и т.д.), трансляторов, баз данных и пр. – работа со строками (последовательностями символов). Хотя операторы для работы со строками за редким исключением не входят в базовый словарь, многие версии Форта снабжаются библиотеками таких операторов, хранящимися на диске. Здесь рассмотрим вариант библиотеки из FIG-FORTH.

Многие операторы библиотеки функционально аналогичны процедурам над числами. Для строк создается специальный стек. Указатель стека строк ($SP) является стандартной Форт-переменной. Имена операторов для работы со строками начинаются с символа $ или ^(см. табл.23). Так, слово, устанавливающее стек строк (SS) в базовое состояние, имеет имя $CLEAR (аналог SP! для стека параметров). Данная библиотека работает со счетными строками, т.е. со строками, первый байт которых содержит код длины строки. Ниже дается краткое описание некоторых операторов из этой библиотеки:

Таблица 23. Операторы для работы со строками

Имя	Состояние стека	Версия	Функция
ARRAY	n → –	M	Слово-описатель, которое создает массив (вектор) чисел одинарной длины. Используется как: n ARRAY name, где name – имя массива, для которого выделяется n+1 ячеек памяти. Обращение к вновь описанному слову: j name. В результате в стек будет записан (j+1)-й элемент массива
ASC	адр → n	M	Записывает в стек первый символ счетной строки, с началом по адресу "адр"
2ARRAY	n1 n2 → –	M	Слово-описатель, создающее двумерный массив чисел одинарной длины с

Окончание табл. 23

Имя	Состояние стека	Версия	Функция
			именем <name>. Используется как n1 n2 2ARRAY <name>. Массив будет иметь n1 рядов и n2 столбцов. Обращение к вновь описанному слову: k1 k2 <name>, где k1, k2 — номера ряда и столбца
$!	адр1 адр2 → —	M, F	Переносит счетную строку с адр1 и записывает ее начиная с адр2
$"	— → адр	M	Записывает строку, выделенную ", в PAD, адрес которого заносится в стек
$+	адр1 адр2 → адр3	M, F	Добавляет счетную строку с началом по адр2 к счетной строке с началом по адр1 и помещает новую строку в PAD, адрес которого записывается в стек
$—TB	адр → адр	M	Удаляет пробелы в конце счетной строки и исправляет счетный байт
$.	адр → —	M, F	Отображает строку, начинающуюся по адресу "адр", на экране. Эквивалентно COUNT TYPE
$COMPARE ($COMP)	адр1 адр2 → а	M F	Сравнивает две счетные строки и записывает в стек —1, 0 или 1 в зависимости от того, является ли строка, начинающаяся с адр1, больше, равна или меньше строки по адр2
$CONSTANT	— —	M	Слово-описатель, создающее строку-константу. Обращение: $CONSTANT <name> <строка>" образует слово с именем <name>, текст строки выделен ". При обращении <name> в стек записывается адрес начала строки
$VARIABLE	n → —	M	Слово-описатель, создающее строку-переменную. Обращение: n $VARIABLE <name> образует строку с именем <name> и длиной n+1 байт. В исходный момент счетный байт равен 0. При обращении к <name> в стек записывается адрес этой счетной строки
$XCHG	адр1 адр2 → —	M	Меняет местами строки, начала которых лежат по адр1 и адр2. Длины строк должны быть идентичны

Примечание. F — FIG-FORTH, M — MMSFORTH.

$LEN выдает в стек значение длины строки, хранящейся в стеке SS, если же стек пуст, сообщает " $STACK EMPTY" и осуществляет уход в QUIT;

$DROP удаляет строку из SS (аналог DROP);

$. выдает строку на экран, удаляя ее из SS;

$COUNT преобразует адрес в стеке параметров в адрес начала строки и число символов в ней (последнее число на верху стека);

$@TEXT пересылает последовательность символов в стек строк при условии, что адрес начала этой последовательности и число элементов в ней записаны в стек параметров;

$@ пересылает строку, адрес которой лежит в стеке параметров, в стек строк. (: $@ COUNT $@ TEXT ;) ;

$! пересылает строку из SS в память начиная с адреса, указанного в стеке параметров;

$DUP дублирует последнюю строку в SS (аналог DUP для чисел в стеке параметров);

$SEG засылает в SS сегмент текста, адрес начала и конца которого определен в стеке параметров;

$OVER – аналог OVER для строк в SS;

$SWAP – аналог SWAP для строк в SS;

$COMP сравнивает две верхние строки в SS, удаляет их из стека, а результат сравнения записывается в стек параметров (0 означает, что все элементы короткой строки совпадают с элементами строки 2);

$+ соединяет две строки в одну;

$DIM создает в словаре Форт строку-переменную длиной, заданной числом в стеке параметров;

$NULL создает "нулевую" строку (одно нулевое слово в SS);

$STR преобразует число из стека параметров в последовательность кодов ASCII и засылает их в SS:

```
: $STR S->D SWAP OVER DABS                    ( в стеке  S |D|)
      <# #S SIGN #> ( преобразование числа в последователь-
                      ность кодов ASCII)
      $@TEXT ;
```

Здесь приведен ограниченный список операторов лишь для того, чтобы пояснить назначение и некоторые возможности библиотеки процедур для работы со строками.

Нередко требуется организовать хранение больших объемов данных на магнитном диске. Для информации объемом 1 Кбайт (экран) необходимый аппарат взаимодействия имеется в базовом словаре (BLOCK, BUFFER, LOAD и т.д.). Но 1 Кбайт – слишком крупная единица информации. Удобный доступ к данным на диске предоставляет описанная в FIG-FORTH система виртуальных массивов. Единица информации (запись) в виртуальном массиве может иметь любую длину. Желательно, чтобы на экране укладывалось целое число записей, поэтому ряд возможных значений длин записи имеет вид: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 байт. В виртуальном массиве могут храниться отдельные байты (символы), числа, последовательности символов (строки, тексты) и массивы чисел. Длина записи постоянная, для виртуального массива длина задается при его описании. Разумеется, у различных массивов длина записи может быть разной.

Пусть мы хотим описать виртуальный массив с именем VIRA из записей длиной L и знаем, что на диске начиная с экрана M достаточно свободного места. Поставленную задачу можно решить, записав L N M VIRRAY VIRA. В словаре появится новое слово с именем VIRA. Теперь при обращении К VIRA программа считает с диска нужный экран и запишет в стек адрес первого байта К-го элемента (записи) массива. Если K>N или K<0, система выдаст сообщение OUT OF V-ARRAY, т.е. заказанный элемент не может лежать в пределах данного массива. Если предполагается описать совокупность виртуальных массивов, целесообразно ввести переменные, где будут храниться номер экрана и номер первого свободного байта. Эта информация будет использоваться и корректироваться при каждом очередном описании виртуального массива, что избавит программиста от необходимости самому следить, где и сколько свободного места на диске.

Виртуальные массивы элементов – удобный инструмент при построении баз данных, всевозможных справочных систем, программ типа HELP и т.д. Если элемент виртуального массива исправлен или переписан, для его записи на диск необходимо до обращения к какому-либо другому экрану выполнить команду UPDATE, а по завершении всей процедуры – FLUSH (SAVE-BUFFERS). Допускается описание виртуальных массивов, перекрывающихся на диске частично или полностью и имеющих, вообще говоря, различную длину записей. Ниже приведено описание оператора VIRRAY:

```
: VIRRAY <BUILDS          ( образование нового входа в словарь
                            с именем описываемого виртуального массива)
        , , ( запись в PFA нового слова  номера экрана, где
```

```
        начинается виртуальный массив,и числа записей
        в нем)
DUP ,                    ( запись в PFA длины записи)
1024 SWAP / ,             ( запись в новое описание числа
                        записей, которые могут поместиться
                        на одном экране)
DOES>                     ( формирование CFA нового слова)
>R                    ( запись в стек возвратов адреса PFA,
                      где хранится номер стартового экрана;
                      начало обработки параметров виртуаль-
                      ного массива и входных данных)
DUP I                ( в стеке REC REC SCR, где REC - номер
                       записи, SCR - адрес, кода стартового
                       экрана)
2+                   ( запись в стек адреса  числа записей
                       в массиве)
@ < OVER 0< OR              ( номер запрашиваемой записи в
                              допустимых пределах?)
IF                          ( если номер записи не верен)
  ." OUT OF V-ARRAY"                ( печать сообщения)
  R> DROP
ELSE            (если номер записи в пределах допусков)
I 6 + @      ( запись в стек числа записей на экране)
/MOD     ( в стеке  номер записи, приращение к номеру
           экрана)
I @ +                       ( вычисление номера экрана  с
                              запрашиваемой записью)
BLOCK                        ( считывание нужного экрана)
SWAP R> 4 + @
* +        ( определение адреса первого байта записи в
             буфере)
THEN ;
```

В PFA слова VIRA хранится адрес ссылки на VIRRAY (PFA), номер стартового экрана (PFA+2), полное число записей в массиве (PFA+4), длина записи (PFA+6) и число записей на экране (PFA+8). Описанный так виртуальный массив должен начинаться с первого элемента на стартовом экране.

Упражнение 1. Опишите виртуальный массив, начинающийся с произвольного элемента на экране.

Упражнение 2. Как можно использовать массив VIRRAY при написании экранного редактора?

Упражнение 3. Опишите слово с формой обращения: адр n DUMP, которое распечатывает в восьмеричной и символьной форме n байт начиная с адреса "адр". С помощью этого оператора распечатайте описание переменной константы и массива.

Решение.

```
: DUMP BASE @ R              ( запись в стек возвратов основания
                               системы счисления)
    OCTAL                  ( устанавление восьмеричной системы)
    8 / 0 DO    ( начало цикла распечаток последовательности
                  байтов по 8 в строке)
          DUP 7 U.R SPACE  ( распечатка начального адреса
                             строки)
          8 0 DO          ( начало цикла распечатки очередных 8
                            байт)
              DUP I + C@ 4 .R          ( распечатка байта)
```

```
      LOOP 4 SPACES
  8 0 DO          ( начало цикла распечатки тех же 8
                    байт в символьной форме)
       DUP I + C@ DUP        ( код символа в стеке)
       32 < OVER '176 > OR     ( это код символа?)
       IF DROP '56 THEN ( если нет, печатаем ".")
       EMIT                    ( отображение символа)
      LOOP
   CR 8 +                  ( переход на новую строку)
 LOOP DROP R> BASE ! ;          ( завершение цикла и
                      восстановление системы счисления)
```

При определении адреса массива вспомните об операторе '(апостроф). Попробуйте переписать эту программу для распечатки чисел, а не байт. Сравните это описание с тем, что содержится в тексте интерпретатора в приложении 9.

Упражнение 4. Опишите слово, которое содержит список заданий, аналогично тому, что описано в гл.4, но занимающее меньше места в памяти (не требующее ";" в конце).

Решение. CREATE TASKLIST] TASK1 TASK2 ... TASK10 [

В данном примере в списке 10 слов-заданий, все они предполагаются описанными ранее. Использование аналогично рассмотренному в гл.4. Такая структура называется иногда исполнительным вектором.

Глава 7. Интерпретация

7.1. ЗАГРУЗКА ПРОГРАММЫ В СЛОВАРЬ

При загрузке системы Форт первой исполняется программа COLD (INI), которая очищает все буферы, устанавливает в исходное состояние указатели стеков возврата и параметров, формирует указатель области USER, загружает векторы прерывания (если требуется) и присваивает некоторым системным переменным (S0, R0, TIB, WIDTH, WARNINC, DP), а также указателям начала и конца экранных буферов (версия FIG-FORTH) рабочие значения. Последнее открывает возможность динамического перераспределения памяти и приспособления имеющейся у вас версии интерпретатора к конфигурации ЭВМ. Программа может быть исполнена и непосредственно из системы Форт.

Программу COLD условно можно разбить на две части: первая от начала до завершения записи системных переменных и остальная часть (называемая иногда WARM [14]), служащая для загрузки переменных USER, векторов прерывания и других вспомогательных операторов. В некоторых версиях Форта имеется слово WARM, которое передает управление этой части программы.

После выполнения стартовых операций производится установка десятичной системы счисления, а словарь Форта делается контекстным (выдается команда FORTH DEFINITIONS). Весьма полезным заверше-

нием программы COLD (INI) является уход на загрузку некоторого экрана (например, экрана 1), где записана программа стартовой загрузки. Эта программа может варьироваться с помощью редактора и подстраиваться под текущие требования конкретной задачи. Это может быть загрузка редактора, Форт-ассемблера или каких-либо других библиотек. Если для стартовой загрузки одного экрана мало, в конце экрана 1 следует написать M LOAD, где M – номер экрана, содержащего продолжение программы стартовой загрузки.

Команда LOAD (см. табл. 8) используется при загрузке не только системы Форт, но и любых библиотек или программ пользователя. Рассмотрим, как она работает:

```
: LOAD BLK @ IN @      ( в стеке номер загружаемого экрана N,
                        старые значения переменных BLK и IN)
       0 IN !         ( обнуление указателя экранного буфера)
       ROT                          ( в стеке  BLK,IN,N )
       BLK !                    ( BLK=N, в стеке  BLK, IN )
       INTERPRET      ( загрузка и интерпретация текста блока
                        с номером BLK слово за словом)
       IN !            ( восстановление прежнего значения IN)
       BLK ! ;        ( восстановление прежнего значения BLK)
```

Таким образом, если при загрузке экрана i встретится команда j LOAD, то производится интерпретация экрана j, а затем загрузка экрана i продолжится. Если команда j LOAD выдана в пультовом режиме (BLK=0), то после загрузки система Форт вернется в пультовой режим.

LOAD является основной командой загрузки операторов в словарь. Так как загружаемый экран N интерпретируется, то все описания типа : NNN XXX ; , CONSTANT и VARIABLE загружаются в словарь, остальные воспринимаются как программы непосредственного исполнения. Это можно использовать для присвоения нужных значений системным переменным, записи векторов прерывания и задания режима работы системы. В качестве примера ниже представлена программа стартового загрузчика, размещенная на экране 1:

```
DECIMAL TTY ( сброс флага печати) CURRENT @ CONTEXT ! [
: PNX 2+ DUP @ ;
: CNTX CONTEXT @ ;
: USAV DUP                  ( чтение с диска и запись в словарь
                              двоичного образа секции словаря)
       IF 6 0     ( загрузка последовательности из 6 экранов)
          DO DUP BLOCK  ( чтение в буфер содержимого экрана)
             DUP @ SWAP PNX
             SWAP PNX R# !                ( распаковка данных)
             PNX DP !       (установка нового указателя HERE)
             PNX CONTEXT @ !            ( установка контекста)
             2+ ROT ROT CMOVE ( перенос информации из буфера
                                в словарь)
```

```
        R# @ 0=        ( проверка условия конца загрузки)
        IF LEAVE
        THEN 1+
    LOOP
THEN DROP ;
```

Первая строка исполняется непосредственно, остальная часть загружается в словарь.

Могут оказаться полезными и специальные сообщения, введенные в текст экранов, например типа SCR# N, позволяющие контролировать процесс загрузки, команды ." E" ." D" ." I" ." T" ." O" ." R" (или ." E" ." D" ." ." ." ." и т. д.), помещенные в начале текстов экранов редактора. Первая из команд записывается на экране 1 редактора, вторая – на экране 2, соответственно шестая на экране 6. По завершении загрузки на экране появится надпись EDITOR. При сбое по последней напечатанной букве легко определить, где он произошел.

Более эффективным средством может стать специальный оператор GDE, который работает с модифицированным оператором LOAD:

```
: GDE ." S#=" PREV @ @ . R# @ 64 / ."LINE=" . ;
```

распечатывая номера экрана и строки, где имела место ошибка. Оператор GDE должен быть выполнен сразу после появления сообщения об ошибке.

Частичное или полное удаление загруженной части словаря возможно с помощью оператора FORGET (забыть):

```
: FORGET CURRENT @ CONTEXT @ - '30 ?ERROR          ( проверка
                соответствия переменных CURRENT и CONTEXT)
        [COMPILE] '   ( определение PFA слова, следующего за
                      FORGET)
        DUP FENCE @ U< ( проверка того, что указанное слово
                       хранится за пределами базового
                       словаря. FENCE указывает верхний
                       край базового словаря)
        '25 ?ERROR      ( сообщение об ошибке, если слово
                          оказалось в защищенной области)
        DUP NFA       ( определение адреса поля имени слова)
        DP !                    ( изменение указателя HERE)
        LFA @           ( указание на предшествующее слово)
        CONTEXT @ ! ;       ( изменение переменной CONTEXT)
```

Обращение: FORGET <NAME>, где <NAME> – имя оператора в загруженной части словаря.

Оператор U< применен вместо "<", так как адреса FENCE @ и PFA <NAME> могут оказаться по разную сторону от 32 Кбайт, т. е. восприниматься как числа разных знаков. Для ЭВМ с памятью более 64 Кбайт эта часть оператора должна быть усложнена: надо учесть содержимое сегментных регистров для каждого из адресов. При исполнении FORGET удаляются все слова начиная с указанного вплоть до конца словаря (CONTEXT @ @).

В режиме "покоя" система Форт ожидает ввода, это делается в рамках исполнения процедуры QUIT. Все, что вводится с терминала, поступает в кольцевой буфер (см. рис. 1), откуда коды извлекаются словом QUERY, которое является частью QUIT и, в свою очередь, пользуется услугами оператора EXPECT. Последний осуществляет выборку из кольцевого буфера с помощью слова KEY. После завершения ввода строки при нажатии клавиши <ВК> строка окажется во входном Форт-буфере TIB. Управление передается оператору INTERPRET (табл. 24), который и выполняет последующую обработку введенного текста.

Таблица 24. Управляющие операторы

Имя	Состояние стека	Версия	Функция
CONSTANT XXX XXX:	n → – – → n	9, 3, F	Формирует константу с именем XXX, равную n. При обращении к XXX в стек записывается число n
VARIABLE XXX XXX:	n → – – → адр	9, 3, F	Формирует переменную с именем XXX, равную n. При обращении к XXX в стек записывается адрес n
CREATE XXX XXX:	– → – – → адр	9, 3, F	Формирует слово с именем XXX (заголовок и CFA). При обращении к XXX в стек записывается его адрес
INTERPRET	– → –	F	Интерпретирует последовательность слов до тех пор, пока там что-то есть
WORD	s → – (s → адр)	9, 3, F	Считывает одно слово из входного или экранного буфера и размещает его начиная с адреса HERE. Первый байт содержит число символов в слове
EXECUTE	адр → –	9, 3, F	Исполняет слово, CFA которого хранится в стеке
[	– → – (I)	9, 3, F	Устанавливает режим компиляции
]	– → –	9, 3, F	Устанавливает ражим исполнения

Примечание. I – слово немедленного исполнения; 9 и 3 – стандарты Форт-79 и Форт-83, F – FIG-FORTH.

Интерпретация производится последовательно слово за словом (слово – это последовательность символов, завершаемая кодом пробела или нулем). Из входного или экранного буфера (что определяется переменной BLK) слово переносится в конец словаря, на адрес которого указывает оператор HERE. Введенное слово имеет вид k XXXXXXX, где k – число символов в слове; XXXXXXX – символы, образующие слово. Далее поведение программы зависит от того, в каком режиме она находится – исполнения или интерпретации. Если это режим исполнения и слово в словаре найдено, осуществляется переход к исполнению команд (EXECUTE), содержащихся в его описании. Если это режим интерпретации, то в случае обнаружения имени вновь описанного слова в словаре,ЭВМ предупредит программиста об этом (MSG# 4 для FIG-FORTH), но тем не менее приступит к его описанию (COMPILE). Могут происходить непредусмотренные совпадения имен, если максимальная длина имен выбрана малой (системная переменная WIDTH) и символы, лежащие за этой границей, отбрасываются. Эта работа проделывается в некоторых интерпретаторах помимо воли программиста и без оповещения его об этом. Поэтому нежелательно WIDTH < 15, обычно WIDTH ≃ 30. Вышеуказанные операции производятся словом CREATE, которое, кроме того, формирует поле связи с предшествующим описанием (LFA), а также поле команды (CFA). Описание слова приобретает форму, которая описана в гл. 6 ч.I.

Если в тексте описания встретится имя слова, которого в словаре нет, интерпретатор попытается его воспринять как число и при неуспехе выдаст сообщение об ошибке (MSG# 0 b FIG-FORTH).

Переключение системы Форт в режим интерпретации производится оператором ":", который можно описать следующим образом (на самом деле это ненастоящее описание, так как описать слово ":", используя оператор ":", нельзя):

```
: :                                        ( второе ":" - имя слова)
    ?EXEC          ( система в режиме исполнения? Если нет, то
                     сообщение об ошибке)
    CSP! CURRENT @ CONTEXT !                ( CONTEXT = CURRENT)
    CREATE           ( формирует имя нового слова и поле LFA)
    ]                                      ( вход в режим компиляции)
    (;CODE) ; IMMEDIATE                     ( формирование поля CFA)
```

Обычно по окончании описания оператора ":" в интерпретаторе размещена программа DOCOL [14] (или $COL), адрес которой записывается в CFA любого слова, описание которого начинается с ":". Оператор ":" является словом немедленного исполнения.

Выход из режима интерпретации осуществляет оператор ";":

```
: ; ?CSP COMPILE ;S    ( завершает описание слова, записав в
                         PFA нового слова ссылки на оператор ;S).
  SMUDGE [ ; IMMEDIATE
```

(Здесь справедливо замечание, высказанное выше к описанию ":".) Если оператора ";" нет, будет выдано сообщение об ошибке.

Операторы CSP! и ?CSP в ":" и ";" контролируют сохранение указателя стека параметров в процессе интерпретации слова; SMUDGE (SMUG) делает новое слово узнаваемым при очередных поисках в словаре. Оператор [переключает систему в режим исполнения и обнуляет системную переменную STATE.] присваивает этой системной переменной значение '300 (в некоторых версиях другое число) и переводит систему в режим интерпретации.

Другим важным оператором является уже упомянутый CREATE:

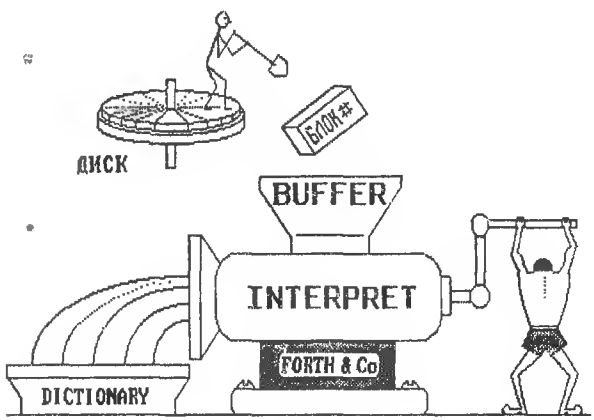
```
: CREATE -FIND                          ( поиск слова в словаре)
         IF                                    ( если оно найдено)
            DROP NFA          ( запись адреса поля имени в стек)
            ID.                                  ( печать имени)
            4 MESSAGE                      ( печать "MSG# 4")
         THEN HERE DUP C@        ( запись в стек числа символов
                                   в слове)
         WIDTH @    ( максимально допустимое число символов в
                      в имени )
         MIN           ( укорочение имени, если оно слишком
                         длинное)
         1+ ALLOT             ( выделение нужного числа байт)
         ?ALLIGN     ( выравнивание выделенного числа байт на
                       четную границу, что важно для ЭВМ типа
                       СМ)
         DUP '240 TOGGLE              ( коррекция байта имени)
         HERE 1- '200 TOGGLE      ( коррекция последнего байта
                                    имени)
         LATEST ,                    ( формирование поля LFA)
         CURRENT @ !               ( коррекция значения CURRENT)
         HERE 2+ , ;                       ( заполнение поля CFA)
```

После команды CFA переписывается при исполнении (;CODE), например при работе оператора ":", CONSTANT или VARIABLE. Оператор TOGGLE служит для приведения к соответствующему виду первого и последнего байтов имени вновь описываемого оператора. При обращении ADR b TOGGLE извлекается байт из ячейки с адресом ADR, выполняется операция XOR (Исключающее ИЛИ) над этим байтом и байтом b, а результат записывается снова в ячейку с адресом ADR.

В заключение рассмотрим алгоритм самого оператора INTERPRET:

```
: INTERPRET BEGIN                    ( начало бесконечного цикла)
          -FIND                ( поиск очередного имени в словаре)
             IF                              ( если слово найдено)
               STATE @ <          ( в стеке байт длины, STATE ;
                                  проверяется, не является ли слово
                                  оператором немедленного исполнения)
                IF                           ( если обычное слово)
                 CFA ,              ( запись CFA найденного слова
                                      в новое описание)
                ELSE    ( если слово немедленного исполнения)
                     CFA EXECUTE          ( исполнение найденного
                                            слова)
```

```
   THEN
ELSE        ( если слово не найдено, проверяется -
              может быть это число ?)
      HERE NUMBER DPL @ 1+ ( преобразование пос-
                  ледовательности кодов ASCII в число)
  IF                    ( если число двойной длины)
     DLITERAL   ( запись последовательности "LIT,
                  число" в описание нового слова)
  ELSE DROP LITERAL     ( запись в описание числа
                          одинарной длины)
  THEN
THEN
?STACK                ( указатель стека в норме ?)
0 UNTIL ;
```



Как же программа после завершения интерпретации может уйти в режим ожидания ввода? Это осуществляется при интерпретации слова немедленного исполнения с нулем в поле имени. Для этого оператор EXPECT пишет два нулевых байта в конце строки во входном буфере, аналогичные нули имеются в конце экранных буферов. Поэтому интерпретатор рано или поздно встретит это слово. Уход из бесконечного цикла здесь происходит с помощью R> DROP и процедуры ;S.

Определенный интерес представляет собой оператор COMPILE, который в режиме компиляции вводит исполнительный адрес следующего за ним слова в компилируемое описание:

```
: COMPILE ?COMP I R> 2+ >R @ , ;
```

Примером использования оператора COMPILE может служить описание оператора ASCII, который в режиме компиляции записывает в описание слова команду записи в стек кода символа, следующего за оператором ASCII, а при исполнении записывает этот код в стек:

```
: ASCII BL WORD ( выделение слова, следующего за оператором
                  ASCII)
        HERE 1+ C@      ( запись в стек кода первого символа
                          этого слова)
        STATE @ IF                      ( если режим компиляции)
          ,         COMPILE LIT ,       ( запись в описание ис-
        полнительного адреса оператора LIT и кода символа)
                THEN ; IMMEDIATE
```

Ниже приведен пример использования оператора ASCII. Слово TEST будет отображать бесконечную последовательность букв A до тех пор, пока вы не нажмете клавишу E:

```
: TEST BEGIN ASCII A EMIT ?TERM ASCII E = UNTIL ;
```

Для программистов, которые заняты прикладными задачами, структура интерпретатора — тема достаточно абстрактная и скучная. Транслятор представляется им весьма сложной программой, что до какой-то степени справедливо, когда речь идет о современных трансляторах для языков высокого уровня. Теоретически можно создать интерпретатор Форт, который займет всего несколько сотен байт [23]. Как правило, Форт-интерпретатор можно разделить на три части. Первая содержит тексты примитивов, реализующих наиболее часто используемые операции (+, –, *, /, >, < и т. д.). Эти примитивы организованы в соответствии с требованиями словаря Форта и написаны на Ассемблере. Вторая часть служит для описания процедур уже в рамках идеологии Форта. Здесь описание слов имеет форму интерпретируемых Форт-операторов, использующих ссылки на примитивы. Эти описания легко смоделировать на Форте. Третья часть интерпретатора содержит программы, управляющие вводом-выводом. Обычно Форт-интерпретатор имеет модульную структуру, которая диктуется структурой словаря. Модульность позволяет добавлять новые базовые процедуры в интерпретатор. Если вы имеете текст интерпретатора и знаете, как устроен словарь, то написание новой процедуры (или примитива) не составит труда. Дополнительное удобство создает возможность предварительного моделирования процедуры на Форте. В данной книге приведено немало текстов операторов базового словаря, которые можно рассматривать как описания-модели.

Если вы столкнулись с проблемой написания интерпретатора для процессора, не имеющего трансляторов для языков высокого уровня, Форт будет самым подходящим языком. Опытный программист напишет Форт-интерпретатор месяца за два. Сначала пишутся и отлаживаются примитивы, затем процедуры ввода-вывода и наконец все остальное. На этой последней фазе уже можно моделировать операторы на Форте. Текст Форт-интерпретатора для IBM PC приведен в приложении 9.

7.3. ВЗАИМОДЕЙСТВИЕ С ОПЕРАЦИОННОЙ СИСТЕМОЙ

Хотя Форт имеет много черт операционной системы (ОС) и даже при определенных условиях может функционировать автономно, обычно он использует ОС ЭВМ, на которой работает. Взаимодействие с ОС начинается, когда программист вводит с клавиатуры команды RU FORTH (или просто FORTH). Уже на этом этапе разные ОС требуют разных команд. Более серьезные отличия проявляются при работе с периферийными устройствами, в частности с дисками. СССР – страна

стандартов, их у нас так много, что почти на каждой ЭВМ свой стандарт записи. Поэтому, если к вам попала дискета с нужной вам программой, выясните сначала ее "происхождение", если стандарты несовместимы, попытайтесь получить ее копию на магнитной ленте. Репертуар стандартов для магнитных лент заметно беднее, и здесь больше шансов на успех. Форт не относится к числу распространенных языков, и в силу этого, вероятно, вы столкнетесь с проблемой использования программ, написанных на других языках, например Фортране, Паскале, Ассемблере.

Специфика Форта накладывает некоторые ограничения на возможности и способ реализации связи с этими программами. Существует несколько методов решения этой задачи. Один из вариантов – использование вспомогательной программы XLINK, написанной на Ассемблере и связанной с Фортом с помощью редактора связей (LINK). Эта программа содержит обращения к интересующим нас программам. При изменении имен или количества таких программ достаточно изменить соответствующие команды вызова в программе XLINK и повторно воспользоваться редактором связей. Программа, к которой планируется обращение из системы Форт, должна быть представлена в виде объектного модуля. При обращении к редактору связей в список модулей кроме FORTH следует включить XLINK и имена программ, к которым обращается XLINK.

В примере, представленном ниже, осуществлена возможность вызова двух подпрограмм, написанных на MACRO-11. Форма обращения к ним из системы Форт: n VLINK K + AWAY, где n – число параметров подпрограммы, хранящихся в стеке и удаляемых оттуда программой XLINK; AWAY – оператор, обеспечивающий сохранение содержимого регистров общего назначения и передачу управления внешней программе; K – константа переадресации, при обращении к первой из подпрограмм (TTT) K=0, ко второй (TTM) K=16. Для удобства записи программы полезно описать константы TTT и TTM, равные соответственно 0 и 16:

```
DECIMAL      0 CONSTANT TTT      16 CONSTANT TTM
```

тогда вызовы программ TTT и TTM будут иметь форму

```
0 VLINK TTT + AWAY   и 0 VLINK TTM + AWAY
```

Если обращений к внешним программам много, полезно ввести вспомогательные слова TTT и TTM:

```
: TTT 0 VLINK AWAY ;   и   : TTM 0 VLINK 16 + AWAY ;
```

0 после имен соответствует числу параметров, необходимых данным внешним подпрограммам. Теперь для обращения к этим подпрограммам достаточно написать TTT или TTM. Имена TTT и TTM могут уже использоваться в новых словах как элементы списка исполняемых процедур, например

: HOWAREYOU 0= IF TTT ELSE TTM THEN ;

Ниже приведен упрощенный текст программы XLINK, написанный на MACRO-11:

```
        .TITLE XLINK
        .MCALL .PRINT
        .GLOBL TTT,TTM
        .MACRO NEXT
        MOV   (SP)+     R4
        MOV   (R4)+     R2
        MOV   (SP)+     R5
        MOV   (R5)+     R1
        ADD   R1,       R5          ;"очистка стека"
        JMP   @(R2)+                ; возврат в Форт
        .ENDM
XLINK:: JSR   PC,   TTT             ;обращение к TTT
        NEXT
        JSR   PC,   TTM             ;обращение к TTM
        NEXT
        .END
```

Если Форт может воспользоваться чужой программой относительно легко, то обратная возможность проблематична. Обычная Форт-программа неразрывна со словарем, собственно она является его частью, в которой описано, к каким словам и в какой последовательности следует обратиться. Дерево этих ссылок завершается примитивами, написанными на Ассемблере. Такая структура программы практически исключает возможность использования ее программами, написанными на других языках. Имеются способы преодолеть этот барьер. Первый шаг в этом направлении позволяют делать программы, которые удаляют из словаря имена слов и поля связи (а также описания, ссылки на которые в словаре отсутствуют). Такой словарь уже не может пополняться. Программа в этом виде компактнее, но еще недоступна для программ, написанных на чужом языке.

Существуют программы [35], которые, используя дерево ссылок, формируют новую программу, состоящую только из обращений к примитивам. В результате получается, как правило, еще более компактная и быстродействующая программа. Если в программе-преобразователе следовать определенным правилам, то можно будет получить программные модули, доступные для использования наряду с другими, написанными на любом из языков, поддерживаемых действующей ОС. Преодолев этот недостаток, мы получим продукт, который уже не

является Форт-программой, и вместе с тем лишимся и всех преимуществ Форта. В этом отношении несколько особняком стоят программы, составленные исключительно на Форт-ассемблере, так как их текст включает только машинные команды.

Когда компактность и быстродействие являются определяющими и планируется длительная эксплуатация программы без каких-либо модификаций, описанные выше преобразования программ представляются разумными. Более того, в процессе преобразования могут быть применены алгоритмы оптимизации, которые сделают уже отлаженную программу еще эффективнее.

7.4. ДЕКОМПИЛЯТОР ДЛЯ СЛОВАРЯ ФОРТА

Нельзя сказать, что декомпиляторы были самые нужные инструменты программиста. Однако время от времени возникает ситуация, когда необходимо разобраться, как работает то или иное слово базового словаря или библиотеки, загружаемой в оттранслированном (двоичном) виде.

Из-за многообразия форм описаний (примитивы, переменные, константы, массивы, структуры ": NNN ... ;" и т. д.) создание универсального декомпилятора довольно сложная проблема. Возможности Форта и на этом поприще продемонстрированы в приложении 9 на примере простейшего декомпилятора с именем DECOD, который ориентирован в основном на декомпиляцию структур типа : NNN ... ;. Много ли вы знаете примеров, когда декомпилятор занимает несколько строк? Стиль, в котором написана эта программа, нельзя назвать хорошим, но она написана и отлажена за пару часов. Форма обращения к оператору: DECOD XXX, где XXX – имя слова в загруженной части словаря, структура которого вас интересует. Декомпилятор – эффективное средство диагностики повреждений загруженной части словаря.

Если вы попытаетесь декомпилировать переменную, константу или еще что-то отличное от структуры ": NNN ... ; DECOD назовет это примитивом. В качестве упражнения вы можете усовершенствовать этот декомпилятор и научить его распознавать переменные и константы.

7.5. ЗАПИСЬ НА ДИСК ЧАСТЕЙ СЛОВАРЯ В ДВОИЧНЫХ КОДАХ

Загрузка любой библиотеки складывается из чтения экранов с диска в буферы, интерпретации содержимого буфера и пополнения словаря. Этот процесс можно ускорить, если исключить интерпрета-

цию, которая в некоторых случаях может занимать 50% общего времени загрузки. Для этого нужная часть словаря записывается на диск в двоичных кодах (двоичный образ), чтобы при необходимости можно было быстро загрузить ее в память. Для записи части словаря (EDT) на диск служит программа

```
32 VARIABLE BAN 52 , 53 , 54 , 55 , 56 , 57 , 60 , 61 , 62 ,
 63 , 64 , 65 , 66 , 67 , 68 , 79 ,     ( "запретный" список)
40 LOAD                                  ( загрузка редактора EDT)

: CHK                                    ( сверка с "запретным" списком)
      1 BAN 2+ DUP BAN @ + SWAP
       DO OVER I @ =
         IF LEAVE DROP
            ." S# " . ." ? " 0           ( если экран в списке)
         THEN 2
       +LOOP ;

: LIST CHK IF LIST THEN ;          : LOAD CHK IF LOAD THEN ;

: EDT  CHK IF EDT THEN ;             : L$   R# @ 1014 MIN ;

: NXP  OVER ! 2+ ;

: SVE                                ( запись части словаря на диск)
      CURRENT @ CNTX - 24 ?ER
      [COMPILE] '                      ( запись в стек PFA слова с
                                         именем <NAME>)
      NFA                          ( замена PFA на  NFA этого слова)
      HERE    ( запись в стек адреса первой свободной ячейки
                словаря)
      OVER                               ( в стеке N, NFA, HERE, NFA)
      - DUP                    ( в стеке N, NFA, HERE-NFA, HERE-NFA)
      DUP 6100 >             ( контроль размера сохраняемой части
                               словаря)
      IF ." SO MUCH ? " . 2DROP        ( если лимит превышен,
                                         задача игнорируется)
      ELSE R# ! 0                ( запись размера части словаря в
                                   ячейку R#)
        DO                        ( начало цикла, в стеке N, NFA)
           DUP 3 PICK                    ( в стеке N, NFA, NFA, N)
           BLOCK             ( резервирование блок-буфера, запись
                               в стек его адреса)
           2DUP !             ( запись NFA в первое слово буфера)
           2+ L$         ( вычисление числа переносимых в буфер
                           байт =K)
           >R R NXP
           L$ MINUS R# +!       ( из содержимого R# вычтено K)
           R# @ NXP              ( запись длины остатка массива
                                   в следующий адрес буфера)
           HERE NXP          ( запись значения HERE в очередной
                               адрес буфера)
           CONTEXT @ @ NXP        ( запись контекстного адреса
                                    в следующую ячейку буфера)
           R CMOVE               ( запись части словаря в буфер)
           UPDATE                 ( установка флага "спасения")
           SWAP 1+ SWAP R + R>
        +LOOP
      THEN FLUSH ;                  ( запись всех буферов на диск)

: SAVE 64 SVE ;

( 64 USAV CR FORGET L$ ." EDT IS HERE"       1 WARNING ! ;S)
```

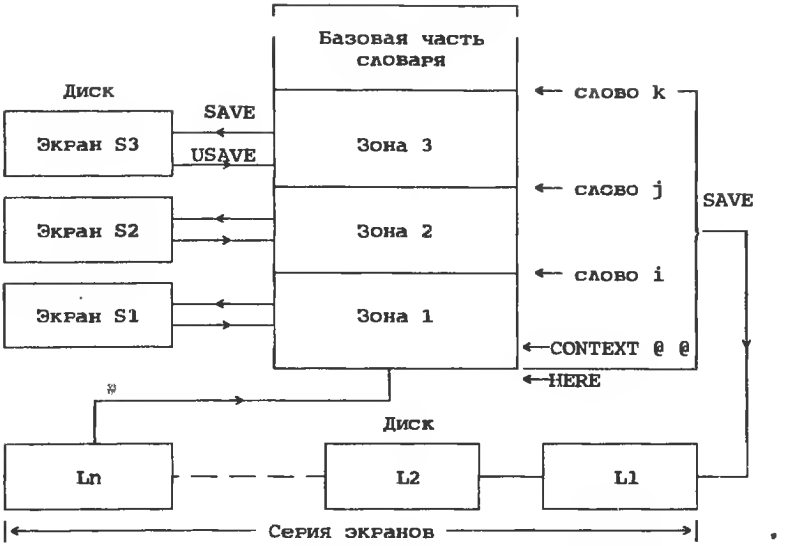


Рис. 8. Схема сохранения и восстановления двоичных секций словаря

Следует иметь в виду, что считываемая в такой форме с диска часть словаря может быть добавлена только к тому же словарю, от которого была отделена. Схема сохранения-восстановления словаря приведена на рис. 8.

Процедура S1 SAVE < Слово i> записывает на диск часть словаря начиная со слова <Слово i> и до конца словаря. Здесь S1 – номер экрана, начиная с которого происходит запись. Последнее слово в словаре начинается с ячейки, на которую указывает CONTEXT @ @, адрес первой свободной ячейки в словаре определяется словом HERE. Записанная часть словаря может быть удалена из словаря с помощью операции FORGET <Слово i>. Это будет работать, если слово i не лежит в запретной зоне, заданной системной переменной FENCE. Процедуру записи можно повторить для слова с именем <Слово j>. На экране S2 будет сохранена зона словаря 2. Далее аналогичная операция может быть проделана для зоны 3. Восстановление словаря выполняется в обратном порядке.

Пусть на экране S3 (на самом деле это может быть серия экранов) записан редактор, на экране S2 – Форт-ассемблер, а на экране S1 – рабочая библиотека. Теперь, если нужно только редактирование, восстанавливаем зону словаря 3 с экрана S3. При работе с Форт-ассемблером восстанавливаем зоны 3 и 2, а если нужна и рабочая библиотека, то зоны 3, 2 и 1. Нельзя восстановить зону 1, не восстановив 3 и 2,

или зону 2, не восстановив зону 3. Попытка это сделать обязательно приведет к сбою системы, поэтому полезно защититься от такой возможности программно. После загрузки двоичного образа какой-либо библиотеки можно обычным путем с помощью LOAD дополнить словарь недостающими операторами. По команде L1 SAVE Слово k на диске сформируется оттранслированный образ словаря со слова k до HERE (серия экранов в нижней части рис. 8).

Чтобы исключить ошибки при попытках чтения, редактирования или загрузки экранов, содержащих двоичные образы секций словаря, заново переопределены слова LIST, LOAD и EDT. Список "запретных" экранов описан в массиве BAN и включает также экраны, где хранится автономная версия Форта, имеющая чисто двоичную форму.

При записи секции словаря (редактора EDT, как в приведенном выше примере) секция программы, ответственная за восстановление этой секции в словаре, должна быть закомментирована (в примере с EDT это последняя строка программы). После загрузки системы Форт и указанной программы выдается команда SAVE BAN, и редактор в двоичных кодах будет записан на серии экранов начиная с 64. После этого с помощью редактора следует закомментировать всю приведенную программу, кроме последней строки, которая и будет использоваться для чтения секции и записи ее в словарь. Оператор USAV описан на экране стартовой загрузки (экран 1). Теперь после загрузки системы Форт достаточно дать команду 3 LOAD (приведенная программа записана на экране 3), и редактор будет быстро загружен.

Оператор CHK проверяет, нет ли данного экрана в "запретном" списке, и, если есть, выдается отклик в виде S# N ?, и команда LOAD, EDT или LIST игнорируется. Ошибочные обращения к "запретным" экранам происходят, когда оператор забывает, в какой системе счисления работает, и, например, пытается отредактировать экран 70, работая с восьмеричной системой счисления.

При желании можно написать программу, которая будет записывать нужные секции словаря в отдельные файлы.

При работе с SAVE/USAV нужно помнить, что, если операция SAVE проведена сразу после загрузки системы Форт, то и USAV следует использовать в тех же условиях. Если вы не уверены, что состояние словаря отвечает этим требованиям, выполните команду COLD (INI), которая вернет словарь к базовому состоянию.

7.6. ВЫПОЛНЕНИЕ ПРОГРАММЫ

В языках типа Фортрана или Паскаля процедура исполнения начинается обычно с загрузки в память подготовленной программы с

диска. Подготовка включает трансляцию исходного текста и редактирование связей. В интерпретаторах типа Бейсик отдельные фрагменты или вся программа сначала преобразуются в машинные коды, после чего начинается ее исполнение. В Форте исполнима только программа, загруженная в словарь. Элементы словаря можно условно разделить на примитивы, где в поле параметров записана последовательность машинных кодов, и на описания типа двоеточия, в поле параметров которых список процедур (CFA), подлежащих исполнению. Особый класс Форт-слов образуют переменные, константы, массивы, исполнительные векторы, слова без заголовков и т. д., часть из которых не нашла отражения в этой книге. Если система находится в режиме исполнения (STATE=0) и во входном потоке (входной или экранный буфер) встретилось слово с именем <NNN>, процедура его исполнения начинается с поиска имени <NNN> в словаре. При успешном завершении поиска в стек записывается CFA слова (Форт-79), после чего управление передается оператору EXECUTE (см. описание слова INTERPRET в § 7.2), который и исполнит слово <NNN>. Если слово является примитивом, в его CFA (поле программы) хранится адрес PFA (т. е. CFA+2), куда и передает управление оператор EXECUTE. Исполнение любого примитива завершается процедурой NEXT (см. гл. 5), которая передает управление следующему слову. Теперь рассмотрим исполнение слов типа двоеточия.

В поле CFA записан адрес исполнительной программы (DOCOL или $COL), которая подготавливает исполнение списка процедур, хранящегося в поле параметров описания. Эта программа может служить примером слова без заголовка. Описание типа двоеточия завершается ссылкой на слово ;S, которое является примитивом и засылает адрес следующей процедуры из стека возвратов в счетчик инструкций (IP). Если в описании содержатся ссылки только на примитивы, переход от исполнения предшествующего слова к последующему происходит через процедуру NEXT, которая в этом случае является единственным источником непроизводительных потерь времени. При наличии в списке слов типа двоеточия в работу включаются также процедуры типа DOCOL и потери времени возрастают. Наличие этих связующих программ – отличительная особенность Форта, с помощью них цепочка примитивов базового словаря образует программу, реализующую задание, описанное в слове программы пользователя. Именно поэтому Форт-программы называют иногда "шитыми" (или цепными) кодами.

Комплект программ DOCOL, NEXT и др. образует внутренний Форт-интерпретатор, работающий в режиме исполнения программы. Функцией внешнего интерпретатора является синтаксический разбор

входного потока и компиляция. При исполнении программы внешний интерпретатор передает управление внутреннему. По завершении исполнения система возвращается в состояние ожидания (QUIT).

7.7. ОТЛАДКА ФОРТ-ПРОГРАММ

Отладка программы на любом языке – процесс весьма индивидуальный, сильно зависящий от вкусов, опыта и традиций. Форт с его особенностями ставит программиста в определенные рамки. К таким ограничениям относится максимальное число символов (64) в экранной строке, отсутствие в тексте программы кодов "Возврат каретки" и "Перевод строки". На первый взгляд, это не столь важное ограничение. Но нужно принимать во внимание, что, стремясь к экономному использованию дискового пространства, программист будет стараться заполнять строку на экране. А это неизбежно приведет к ухудшению читаемости программы, усложнит поиск нужного описания.

Все знают, что комментарии – неотъемлемая часть программы. Но всегда ли мы следуем разумному правилу: комментировать программу в процессе ее написания? В Форте комментированию препятствует ограниченный объем экрана – 1024 символа, включая пробелы. Ведь чем больше объем комментариев, тем меньше "видимый" на экране текст. Именно это и стало причиной введения в некоторых реализациях Форта "теневых" экранов с комментариями к текстам на "основных" экранах. В такой ситуации можно порекомендовать следующее.

При отладке не жалеть места для комментариев. Имена всех слов должны максимально отвечать функции слова, в рабочей версии они могут быть условными (хотя последнее и небесспорно). В какой-то мере это будет способствовать защите авторских прав, ведь разобраться в программе с условными именами и без комментариев крайне сложно, а люди хотя и любопытны, но ленивы. Предлагаемая методика предполагает дополнительную переработку отлаженной программы (например, удаление комментариев). Эту работу можно поручить специальной программе. Другой путь – хранение отлаженной программы на диске в виде двоичного образа секции словаря или полная переработка текста с удалением имен, ячеек связи. Но в любом случае лучше оставить программу с комментариями, пожертвовав местом на диске, чем написать, отладить и оставить программу без комментариев. Через год вы будете смотреть на нее как на чужую, и любая переделка займет больше времени, чем самые обширные комментарии. Это ни в коем случае не относится к именам базового словаря. Держать в памяти имена операторов, функция которых неочевидна из названия, весьма обременительно. Даже имена таких операторов базового

словаря, как @ и !, могут вызвать неудовольствие, что проявилось в их замене в системе GRAFORTH на PEEKW и POKEW соответственно.

Основным инструментом при отладке программы является печать промежуточных результатов, состояния стека и содержимого переменных и массивов в процессе тестовых прогонов. Так как в Форте возможны ссылки только на слова, описанные и загруженные в словарь, именно эти слова должны отлаживаться первыми. Только после отладки слов-компонентов следует приступать к проверке операторов, куда эти слова входят.

Следует избегать слишком длинных описаний. Каждое слово должно выполнять определенную функцию. Причем имя слова должно соответствовать этой функции. Функционально сложные описания целесообразно разбивать на части (по крайней мере при отладке). В длинном описании можно временно исключать некоторые секции, заключив их в скобки (т. е. превратив их на время в комментарий).

Для распечатки стека следует использовать команды S. или O., которые не изменяют его состояния. Оператор O., идентичный S., представляет числа в восьмеричном виде, оставляя основание системы счисления неизменным:

```
: S. DUP U. ;   или   : S. DUP . ;
: O. BASE @ OCTAL S. BASE ! ;
```

Напомним, для положительных чисел оператор U. эквивалентен ".", т. е. он удаляет число из стека и отображает его на экране. Но отрицательные числа U. распечатывает иначе, например

```
DECIMAL –5 U.<ВК> 65531 OK
OCTAL   –5 U.<ВК> 177773 OK
```

U. рассматривает старший бит кода как обычный двоичный, а не как знак числа. Это целесообразно при распечатке частей словаря, программ, управляющих кодов и т. д. Когда для наглядности удобно осуществить двоичное представление чисел на экране (или печати), можно воспользоваться оператором B в виде

```
: B. BASE @ 2 BASE ! OVER U. BASE ! ;
```

Например: DECIMAL 15 B. <ВК> 1111 OK.

Чтобы распечатать второе сверху число, можно воспользоваться командой OVER . или OVER U. .Для распечатки чисел, положение которых в стеке произвольно, подходит кэманда k PICK U., которая также не изменяет состояния стека (k – номер позиции в стеке начиная сверху).

Для контроля состояния стека в различных точках программы полезен оператор DEPTH, который выдает в стек полное количество чисел, хранящихся в стеке до исполнения этой команды. Для распечатки всего содержимого стека пригодно слово STY, оставляющее стек без изменений. Первое выдаваемое число соответствует верху стека. Если в вашей версии Форта нет слов DEPTH и STY, вы можете их описать сами:

```
: DEPTH S0 @ SP @ 2+ - 2/ ;
```

S0 – системная переменная, равная базовому значению указателя стека параметров:

```
: STY DEPTH -DUP      ( определение глубины заполнения стека)
      IF 0
         DO I 8 MOD 0=                .( подготовка табуляции)
            IF CR ( ввод возврата каретки для того, чтобы при
                  большой глубине заполнения стека результат
                  имел вид таблицы по 8 чисел в строке)
            THEN I 1+ PICK 7 U.R
         LOOP
      THEN ;
```

При DEPTH=0 STY ничего не печатает.

Другим полезным при отладке словом может стать PRS, которое распечатывает n верхних слов стека без модификации указателя:

```
: PRS DEPTH MIN     ( защита от перехода через нижнюю границу
                      стека)
      -DUP
      IF ( если не 0) 0
         DO I 1+ PICK U.            ( печать числа без знака)
         LOOP
      THEN ;
```

Это слово лучше использовать, когда стек загружен на большую глубину и его полная распечатка нецелесообразна. А что будет, если стек пуст, а вы выдали команду . <ВК>? Ничего страшного, ЭВМ выдаст сообщение типа STACK EMPTY или MSG# 1. При переполнении стека ЭВМ выдаст сообщение STACK OVERFLOW или MSG# 2. Если такая ситуация сложится в процессе выполнения программы, возможны повреждения словаря, а выполнение программы в любом случае будет прервано.

При работе с массивами чисел Форт не контролирует переход через их границы, что, конечно, может вызвать большие неприятности, так как возможны повреждения словаря. В Форте, как впрочем и во многих других языках, забота о границах массивов лежит на программисте. И, если указатель элемента массива вычисляется, полезно проверять результат на соответствие границам массива, во всяком

случае при отладке. Такая проверка может быть встроена в описание самого массива. Важно также иметь в виду, что некоторые версии Форта не контролируют переполнение при арифметических операциях. Если есть опасность переполнения при нормальной работе программы, используйте числа двойной длины. При повреждении словаря возможны разнообразные последствия. На ЭВМ типа СМ это могут быть прерывания с попаданием в ловушку (TRAP) из-за недопустимого адреса или команды. В случае попадания в ловушку на экране появится сообщение TRAP-ERROR m N1 N2, где m – код ловушки (M=4 означает неверный адрес, а m=10 – недопустимую команду); N1 = PC (программный счетчик) и N2 = PS (регистр состояния). При попадании в ловушку, если "повреждения" не слишком велики, можно попытаться выполнить команду COLD (INI). Подумайте, почему это могло произойти, и проверьте вашу гипотезу. В случае серьезных "разрушений" придется перезагрузить систему с диска или даже воспользоваться копией на другом диске, предварительно изготовив еще одну. Особенно внимательным надо быть при отладке программ, производящих запись на диск. Следует, например, помнить, что последовательность "K BUFFER ... UPDATE" приведет к записи на экран К и утрате прежних записей. Копирование экранов также стирает экран-адресат. Поэтому при отладке всегда имейте копию программы на отдельном диске. Причем при работе с ОС полезно иметь под рукой и ее копию, так как ошибкам свойственно наносить максимально большой урон. Доступность из системы Форт абсолютных адресов создает не только определенные удобства, но и серьезные трудности. Неправильно вычисленный адрес присваивания может повредить словарь или какой-либо элемент ОС, хранящийся в памяти. Копия файла FORTH.DAT крайне необходима при редактировании текста редактора. Почему бы не внести какой-то новый редактирующий оператор, если вы считаете, что так вам будет удобнее работать? Но если совершена ошибка, а старой копии не сохранилось, то у вас не останется средств внести исправления, ведь редактор с ошибкой вряд ли будет работать. В этом случае хорошей альтернативой может быть "двоичная" версия редактора, загружаемая оператором USAV. При редактировании совершенно нового (пустого) экрана, где никогда не было текстов Форта, полезно сначала выполнить команду WIPE, так как на экране могут присутствовать "невидимые" (не имеющие печатного образа) символы, которые способны сбивать работу интерпретатора.

Очистить экран (здесь имеется в виду буфер на диске) можно, и не входя в редактор, для этой цели пригоден оператор SCLR:

```
: SCLR ." S# " S. ." CLEAR ? <Y/CR> "        ( входной диалог)
```

```
KEY '131 =                          ( нажата клавиша "Y" ? )
IF BUFFER                           ( резервирование буфера)
   1K BLANKS                              ( очистка буфера)
   UPDATE FLUSH ( установка флага "спасения" и запись
                  на диск)
THEN ;
```

На команду N SCLR ЭВМ сначала выдаст запрос S# N CLEAR ? <Y/CR>, где N – номер экрана, подлежащего очистке (заполнению кодами пробела), а после получения подтверждения <Y> выполнит эту команду. SCLR может помочь, если данный экран не читается из-за ошибки по четности, так как SCLR записывает данные на диск, не читая старого содержимого экрана.

Особый класс ошибок составляют конфликты из-за основания системы счисления. Так, попытка исполнить команду OCTAL 287 . будет интерпретатором пресечена: в восьмеричной системе счисления число 287 не имеет смысла. Аналогичные проблемы могут возникнуть с шестнадцатеричной системой. Определенную гарантию могут дать признаки системы счисления "+", "–" и ",", их можно использовать и для преобразования чисел из одной системы в другую. Так, если надо преобразовать серию десятичных чисел в восьмеричные, достаточно выдать команду OCTAL, а далее печатать с пульта числа со знаком "+" или "–". Например: OCTAL +18 . <ВК> 22 ОК –17 . <ВК> –21 ОК +64 . <ВК> 100 ОК и т. д. Если требуется восьмерично-десятичное преобразование, то можно воспользоваться командами DECIMAL '20 . <ВК> 16 ОК '200 . <ВК> 128 ОК. В версиях, где признаков счисления нет, можно описать специальные слова, например

```
: DH HEX . DECIMAL ;
```

преобразует число из десятичной системы (она предполагается действующей) в шестнадцатеричную: 15 DH <ВК> F ОК и 17 DH <ВК> 11 ОК.

Рассмотрим пример ввода отладочных распечаток в программу упорядочения элементов массива, имеющего имя AR и длину 66 байт:

```
0 VARIABLE AR 64 ALLOT
: FILLING 64 0 DO 64 I - AR I + C! LOOP ;
: ORDER BEGIN 0 R# ! 64 0
              DO AR I ." I=" S. + C@ ." A=" S.
                 AR I + 1+ C@ ." B=" S. CR >
                 IF 1 R# ! AR I + C@ >R
                    ." A" I' 1+ S. AR + ." =" C@ S.
                    AR I' + C! R> S. AR I + 1+ C!
                 THEN
              LOOP R# @ 0=                ( все упорядочено?)
      UNTIL ;
: TT 64 0 DO I 20 MOD 0=
           IF CR                ( "ВК", если индекс кратен 20)
```

```
        THEN AR I + C@ .
    LOOP ;
```

Оператор FILLING заполняет массив кодами с 64 до 0, а TT позволяет распечатать массив, расположив по 20 элементов в строке. Оператор ORDER сортирует и укладывает числа в порядке возрастания их значений. Введенные в слово ORDER распечатки дают исчерпывающую информацию о процессе сортировки и переукладки. Но разумно ли такое решение? Ведь если запустить сейчас ORDER, придется запастись терпением и временем: ЭВМ согласно программе выдаст такое количество информации, что пользы от этого практически не будет. Естественный путь при отладке многоцикловых программ – на первом этапе ограничить число циклов одним-двумя. И только после того, как установлена корректность поведения программы для ограниченного числа циклов, можно, убрав лишние отладочные печати, запустить программу в окончательном виде. При отладке программы слово за словом надо следить, чтобы каждое из них сохраняло указатель стека параметров, а, если и меняло, то только запланированно. Если в стеке по завершении работы слова остаются ненужные коды, они должны быть удалены с помощью DROP или 2DROP.

Алгоритм упорядочивания сводится к поочередному сравнению значений элементов. Если предшествующий элемент больше последующего, то они меняются местами. Внутренний цикл DO...LOOP осуществляет перебор всех элементов массива и, если переставлены хотя бы два элемента, выставляет флаг R#=1. Это указывает на необходимость очередной итерации, которую и запускает оператор UNTIL. В этом примере стек возвратов используется для хранения промежуточных результатов, но так как в цикле DO...LOOP применен индекс цикла (I), то после >R и вплоть до R> извлечь его можно только с помощью команды I', что и делается в операторе ORDER. Распечатка номера элемента выполняется последовательностью команд ." A" I'1 + S. ." =" AR + C@ S., в результате начало распечатки имеет вид A1=64.

7.8. ДИАГНОСТИКА ОШИБОК

В Форте имеется развитая система диагностического контроля, которая при желании может быть дополнена пользователем. К сожалению, пока не выработано единого стандарта на сообщения об ошибках и на методику их регистрации. Ниже описан диагностический аппарат одной из версий FIG-FORTH.

Главным оператором, который оформляет сообщение и выход в систему при выявлении ошибок, является ERROR. Включение в работу оператора ERROR осуществляется оператором ?ERROR (?ER):

```
: ?ERROR SWAP IF ERROR ELSE DROP THEN ;
```

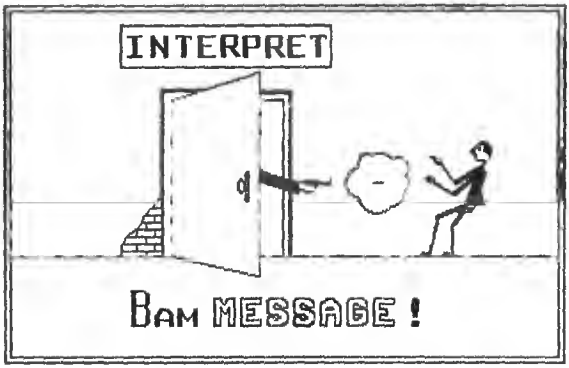
который предполагает наличие в стеке флага и кода ошибки (последний находится на верху стека). Флаг ошибки – это любое число, не равное 0. Например, опишем заново операцию деления:

```
: / –DUP 0= 26 ?ERROR / ;
```

Не беда, что при его загрузке будет получено предупреждение MSG# 4 (IT ISN'T UNIQUE), зато при попытке деления на нуль будет выдаваться сообщение MSG# 26 (0 DIVISION) и выполнение программы будет прервано. Аналогичное усовершенствование вы можете внести самостоятельно в описание операции извлечения квадратного корня (см. § 3.2). (Напомню, на ЭВМ типа СМ деление на нуль (в Форте, так же как и в MACRO-11) не вызовет прерывания, результат вычисления будет заведомо неверным.) Перечень диагностических сообщений FIG-FORTH представлен в табл.25. Кроме перечисленных в таблице сообщений в системе присутствует диагностика ошибок при работе с внешними устройствами (диском, цифропечатью и т.д.). Но они, с одной стороны,

Таблица 25. Диагностические сообщения

Сообщение при WARNING=0	Значения	Сообщения при WARNING=1
MSC# 0	Слово не узнано. Число не узнано. Нет соответствия системе счисления	
MSC# 1	Попытка извлечь нечто из пустого стека	EMPTY STACK
MSC# 2	Переполнение стека или словаря	STACK OR DIRECTORY OVERFLOW
MSC# 4	Повторное описание слова (не является фатальной ошибкой)	IT ISN'T UNIQUE
MSC# 17	Используется только при компиляции	COMPILATION ONLY
MSC# 18	Используется только при исполнении	EXECUTION ONLY
MSC# 19	IF и THEN или другие операторы не имеют пары	CONDITIONALS AREN'T PAIRED
MSC# 20	Определение не завершено	DEFINITION ISN'T FINISHED
MSC# 21	Нелегальный аргумент слова FORGET. Слово в защищенной части словаря	PROTECTED DIRECTORY
MSC# 22	Должно использоваться только при загрузке	USED AT LOADING ONLY
MSC# 26	Деление на 0	0 DIVISION



сильно варьируются от версии к версии, а с другой – достаточно легко понимаемы, поэтому их описание здесь опущено. Исключение составляют сообщения: DISK READ ERROR # N и DISK WRITE ERROR # N, где N – код ошибки (ОС RT-11):

N=1 Данного диска нет в системе
N=2 Отсутствует управляющая программа внешнего устройства
N=3 Отсутствует файл DK:FORTH.DAT (или его заменяющий)
N=5 Что-то со стеком
N=6 Неудача при чтении
N=7 Ошибка при записи

Определенный интерес представляет описание слова MESSAGE, которое в FIG-FORTH служит для выдачи развернутых текстовых сообщений длиной в одну строку:

```
: MESSAGE IN @ R# ! ( сохранение указателя входного буфера в
                      R#)
        WARNING @ IF                          ( если WARNING $0)
        -DUP IF    ( если код диагностического сообщения не
                     равен 0)
        4 OFFSET @ -                ( выбор экрана с текстами
                                    диагностических сообщений)
        .LINE                                 ( печать сообщения)
        ELSE ." MSG # 0" THEN    ( если код ошибки равен 0)
        ELSE ." MSG # " . THEN ;  ( печать сообщения, если
                                    WARNING =0)
```

Управление работой MESSAGE производится через системную переменную WARNING.

Оператор .LINE, используемый в слове MESSAGE, печатает строку L с экрана S, обращение имеет вид L S .LINE:

```
: .LINE >R              ( запись номера экрана в стек возврата)
        64 1024 */MOD    ( частное указывает, на каком экране
                           по отношению к первому лежит
                           нужный текст)
        R> +         ( запись в стек абсолютного номера экрана,
                       откуда будет проведено чтение строки)
        BLOCK +       ( считывание экрана в буфер и запись в
                        стек адреса начала нужной строки)
        64       ( в стеке адрес начала текста и длина строки)
        -TRAILING          ( из 64 вычитается число пробелов в
                             конце строки)
        TYPE ;                                  ( печать строки)
```

Из описания .LINE видно, что этот оператор способен распечатать любую строку из цепочки экранов, начинающуюся с S. Поэтому он может быть полезен и для других целей. Диагностические сообщения обычно размещаются на экранах 4 и 5. Если это не так, задачу можно решить, поменяв значение OFFSET или исправив число, стоящее перед OFFSET в описании MESSAGE. Коду L соответствует в MESSAGE код ошибки. Использованная в .LINE команда ADR N –TRAILING сокращает значение N на число пробелов между последним значащим символом и адресом ADR+N.

Этот оператор особенно полезен при печати, так как исключает холостые пробеги каретки. Видоизменив .LINE, можно получить оператор, пригодный для печати любых текстов (см., например, INDEX), записанных на диске. Вспомните о нем, если будете писать какую-либо HELP-программу или другие слова, которые выдают подсказки оператору за пультом ЭВМ.

Упражнение 1. Опишите оператор LOAD, который производит загрузку экрана S начиная со строки L. Обращение: S L SLOAD.

Решение.

```
: SLOAD BLK @ IN @                    ( в стеке S L (BLK) и (IN) )
        ROT 64 * IN !                  ( в стеке  S (BLK) и (IN) )
        ROT BLK !              ( в стеке остались (BLK) и (IN) )
        INTERPRET ( интерпретация текста  экранного  буфера
                    начиная со строки L)
        IN ! BLK ! ;
```

(BLK) и (IN) – номера экрана и указателя буфера до обращения к оператору SLOAD.

Упражнение 2. Используя SCLR, напишите оператор, который заполняет пробелами серию экранов длиной K начиная с N. Присвойте ему имя SCB. При отладке примите необходимые меры безопасности (запаситесь копией вашего рабочего диска).

Упражнение 3. Напишите программу, которая распечатывает полное содержимое стека, одновременно приводя его в базовое состояние (очищая его).

Упражнение 4. Напишите описание оператора для преобразования восьмеричных чисел в десятичные.

Упражнение 5. Используя .LINE, напишите программу LST, которая бы выполняла работу процедуры LIST.

Упражнение 6. Сформируйте слово с именем LD, которое загружает последовательность экранов, описанную в виде счетной строки с именем SLST.

Решение. Пусть имеется счетная строка

7 VARIABLE SLST 73 , 77 , 81 , 63 , 40 , 90 , 83 ,

(в списке 7 экранов). Тогда

```
: LD SLST @              ( в стеке число загружаемых экранов)
     0 DO SLST 2+ I 2* +   ( запись в стек адреса очередного
                           элемента списка)
     @ LOAD LOOP ;
```

Если номера экранов не превышают 256, список может состоять из байтов и в описании счетной строки следует заменить "," на C,.

Упражнение 7. Как можно описать слово, размещенное на двух или более экранах?

Решение. Это можно сделать, используя команды переноса --> или [N LOAD], которые вводятся в конце описания на экране i, продолжение описания располагается на экране i+1 для --> или N для LOAD. Оператор --> является, как правило, оператором немедленного исполнения. Операторы [и] осуществляют переход в режим исполнения и компиляции соответственно. В этом случае скобка] должна размещаться в самом начале экрана N. Для LOAD, а также для операторов DO, IF и других аналогичных это может привести к ошибкам, так как в процессе компиляции они засылают в стек флаги контроля парности операторов. Более длинное описание LOAD:

: LOAD BLK @ >R IN @ >R 0 IN ! BLK ! INTERPRET R> IN ! R> BLK ! ;

исключает эти проблемы. Поэтому использование оператора --> предпочтительнее.

Глава 8. Применение Форта для систем, работающих в реальном масштабе времени

Программированию для устройств, работающих в реальном масштабе времени, присуща зависимость от структуры и особенностей ЭВМ и периферийного оборудования. Аппаратная часть таких систем строится по магистрально-модульному принципу и обычно стандартизируется. Стандартизация позволяет сократить требуемый ассортимент интерфейсов, но полной унификации достичь не удается. Отсюда и разнообразие пакетов управляющих программ. В таких условиях немалым достоинством является простота адаптации программного обеспечения к новой аппаратуре. Форт предоставляет в этом смысле неплохие возможности. К числу наиболее известных и распространенных стандартов на магистрально-модульные системы относятся КАМАК, шина МЭК-625 (приборная шина, ГОСТ 26.003–80), Q-BUS, UNIBUS, ФАСТБАС, VME и Мультибас [42–44, 46]. Последние три стандарта ориентированы на работу с большим числом процессоров.

Приступая к написанию программ для управления магистрально-модульными системами, нужно выяснить устройство интерфейса

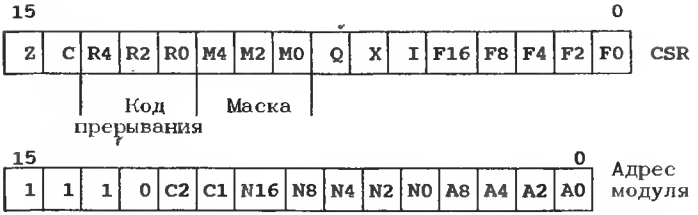


Рис. 9. Назначение разрядов регистра CSR и регистра данных в интерфейсе системы КАМАК

ЭВМ–магистрально-модульная система и способ взаимодействия процессора с внешними устройствами. Ниже будут рассмотрены примеры для случая, когда обращение к внешним устройствам осуществляется через специально выделенную адресную зону, как правило, это 4 Кслов нулевой или последней страницы памяти (ЭВМ СМ1420, ДВК и т.д.). Обычно в этой зоне для любого внешнего устройства выделяется два или более адресов. В простейшем варианте первому адресу соответствует регистр управления и состояния (CSR), второму – регистр данных. Взаимодействие внешнего устройства с ЭВМ реализуется через прерывания [48].

Предположим, мы работаем с системой КАМАК [42], интерфейс которой имеет следующую структуру CSR (рис. 9). Здесь R0, R2, R4 – разряды вектора (код) прерывания, адрес которого равен 340 + R*4, т. е. 344, 350, 354, 360, 364, 370 и 374 (все числа восьмеричные). Всего предусмотрено семь векторов прерывания. M0, M2, M4 определяют порог прерывания. Если код M равен коду R, прерывание не происходит. Код R записывается из системы КАМАК, а читается в ЭВМ на аппаратном уровне. F0, F2, F4, F8, F16 – поле команд системы КАМАК; Q и X – отклики, посылаемые модулями; I – флаг блокировки (запрета); Z и C – разряды для выработки команд подготовки и сброса соответственно. При прерывании управление передается по адресу, указанному в одном из семи векторов (344...374). C1 и C2 – разряды для идентификации номера каркаса, это означает, что данная схема позволяет обращаться максимум к четырем каркасам. N0, N2, N4, N8, N16 – номера станций, где расположены модули, A0, A2, A4, A8 – субадрес регистра в модуле.

Чтобы обратиться к тому или иному модулю КАМАК, надо записать в регистр CSR код команды (F0...F16), затем обратиться по адресу, соответствующему этому модулю. Этот адрес вычисляется по формуле

ADR = CAMAC_BASE + N*32 + A*2,

где CAMAC BASE =160000 (зависит от типа интерфейса); N – номер

станции, где расположен модуль в каркасе; A – субадрес регистра в модуле. Адрес может быть вычислен заранее или определен непосредственно перед обращением.

При отладке аппаратуры и в управляющих задачах удобно использовать операторы MODL и MODLA:

```
OCTAL 166000 CONSTANT CRATE
      160000 VARIABLE CSR
: MODL <BUILDS 40 * CRATE + ,        ( Вычисление КАМАК-адреса
                                     модуля и запись его в описание
                                     нового слова с именем <NAME>)
      DOES> @ ;
```

MODL позволяет сформировать слово, которое при исполнении имеет свойства константы. Обращение к MODL имеет вид N MODL <ИМЯ>, где N – номер станции, где расположен данный модуль; <ИМЯ> – условное имя модуля КАМАК. Обращение к MODLA несколько иное: SA N MODLA <NAME>, где SA – субадрес регистра в модуле:

```
: MODLA <BUILDS 40 * CRATE + SWAP 2* + , DOES> @ ;
```

MODL вычисляет адрес модуля КАМАК при нулевом субадресе, а MODLA – при произвольном. В обоих случаях в поле параметров записывается адрес модуля КАМАК, вычисленный в процессе интерпретации.

Пусть модуль аналого-цифрового преобразователя ADC1 установлен в станции 7. Его описание: 7 MODL ADC1. Выполнив команду 0 CSR ! ADC1 ?, получим на экране значение кода, записанного в модуле. ADC1 записывает в стек восьмеричное число 166340 – адрес модуля КАМАК ADC1. Если модуль допускает запись в него данных, такую операцию легко осуществить: FF CSR ! K ADC1 !, где FF – код команды записи; K – код, записываемый в регистр модуля ADC1.

Стандартный оператор чтения может иметь вид

```
: RD CSR ! @ ;
```

обращение к нему: ADR FF RD, где FF – код команды чтения, ADR – адрес модуля КАМАК. По команде CSR ! код команды записывается в регистр CSR КАМАК-интерфейса, @ считает содержимое регистра в модуле, используя команду FF. Например, ADC1 2 RD выполняет чтение со стиранием (F=2). Запись (: WT CSR ! ! ;) осуществляется аналогично: M ADC1 FF WT, где M – код, записываемый в регистр (или память) модуля; FF – код команды записи (например, FF=17).

Для проверки откликов, вырабатываемых модулем (Q и X), когда это не делается аппаратно, достаточно выполнить команду CSR @ 300 AND. В результате в стеке окажется число, в разрядах 6 и 7 которого

(см. рис. 9) будут флаги откликов Q и X соответственно. Установка системы КАМАК в исходное состояние выполняется командой '100000 CSR !, команда '40000 CSR ! сбрасывает все регистры КАМАК-системы в исходное состояние. Сброса в нуль соответствующих разрядов регистра CSR не требуется, так как в этом случае запись реально не производится.

Можно было бы решить задачу чтения содержимого регистров модуля ADC1 и с помощью процедуры 0 CSR ! 7 32 * CRATE + ? или внутри описания какого-либо слова 0 CSR ! [7 32 * CRATE +] LITERAL ?. Если чтение или запись ADC1 выполняется только один раз, оба решения приемлемы, хотя и громоздки. Первый вариант, кроме того, более "медленный", так как вычисление КАМАК-адреса производится во время исполнения программы. Эти решения, конечно, и менее наглядны. При работе с персональными ЭВМ в интерпретаторе необходимо иметь команды, эквивалентные IN и OUT (READ и PORT, см. описание программы "Подмосковные вечера" в § 8.3).

Примером управляющей программы для системы КАМАК может служить программа в приложении 4. Назначение программы – тестирование памяти емкостью 1024 слов (32-разрядных). Предполагается, что доступ к младшим 16 разрядам возможен по субадресу A0, а к старшим – по A1. Запись числа в адресный регистр памяти осуществляется по команде F17.

В первой строке описано имя КАМАК-модуля памяти (MEM) и адрес управляющего регистра CSR интерфейса КАМАК. Далее следуют описания процедур чтения и записи (RD и WT), а также записи в адресный регистр модуля памяти (MAW). Ниже записана интерактивная процедура определения положения модуля в каркасе и вычисления его КАМАК-адреса (START). На запрос MEMO BIN–> надо ввести номер станции, где расположен модуль памяти. Это число во входной буфер записывает оператор QUERY, а INTERPRET преобразует его и передает результат в стек. Далее полученное число умножается на 32 и складывается с содержимым регистра CSR. Оператор 'MEM записывает в стек адрес константы MEM, а "!" записывает адрес модуля КАМАК. Оператор MCLEAR очищает все регистры модуля памяти. В самой тестовой программе (MTEST) 0 MAW очищает адресный регистр модуля памяти. Проверка проводится путем записи и последующего чтения по субадресам 0 и 1 кодов 0, 1, 2, 4, 8, 16 и т. д. (побитовый контроль). При совпадении записанного и считанного кодов тестирование продолжается для следующих адресов. В случае несовпадения сообщается адрес, субадрес, а также записанный и считанный коды (операторы ERR и RED). Тестирование выполняется на процессоре "Электроника-60" менее 10 с.

Основу взаимодействия программы с периферийным оборудованием составляют прерывания. К сожалению, в базовых словарях Форта отсутствуют стандартные процедуры обслуживания прерываний. Конечно, можно написать программу обработки прерываний на Форт-ассемблере, но это не всегда удобно, ведь тогда надо дополнительно загружать несколько экранов этой библиотеки. По этой причи-

не в версию Форт-интерпретатора, работающую на ЭВМ типа СМ или ДВК, разумно ввести оператор, который позволял бы формировать программы для работы с прерываниями на Форте:

```
: [[ LIT DOIN LATEST PFA CFA LIT 24 CMOVE LIT 22 ALLOT HERE
     DUP LIT 6 - ! ; IMMEDIATE
```

Несколько необычна структура [[, но такой вид она имеет внутри интерпретатора. Именно поэтому здесь применен оператор LIT. Сразу вслед за описанием [[помещается программа на Ассемблере, первый оператор которой имеет метку DOIN:

```
DOIN:    JSR   R5,      @#SAVREG  ; Сохранение содержимого
                                  ; всех регистров
         MOV   #INIST,  R5        ; Инициализация стека
                                  ; прерываний
         MOV   @#USER,  R3        ; Восстановление
                                  ; указателя области USER
         MOV   (PC)+,   IP        ; Уход в программу
                                  ; пользователя
         HALT
       • NEXT                     ; Макроопределение пере-
                                  ; хода к следующей Форт-
                                  ; процедуре
```

Кроме того, вводится Форт-примитив, который имеет имя]] и содержит только одну команду RTS PC (ЭВМ типа СМ или ДВК). Как же выглядит программа обслуживания прерывания на Форте? Пусть имя этой программы XXX:

```
: XXX [[ HOME 50 SPACES ." Московское время "
       TI 7 EMIT ]] ;
```

где HOME – слово, устанавливающее курсор на экране в верхнее левое положение; TI – оператор, который выдает на экран время в виде ЧЧ:ММ (часы:минуты).

Программа, записанная между [[и]], служит для обработки прерываний. На этапе трансляции оператор [[первым делом запишет в стек адрес программы DOIN (LIT DOIN), далее адрес CFA слова XXX (LATEST PFA CFA) и наконец начиная с этого адреса в новое описание вставляется программа DOIN (LIT 24 CMOVE, программа DOIN содержит 20 байт), изменяется указатель HERE. В завершение на место HALT записывается ссылка на процедуру, следующую за [[. Так как прерывание может произойти в момент, когда указатели стека параметров и стека возвратов имеют произвольное значение (например, при исполнении –FIND), последовательность слов после [[пользуется специальным стеком – стеком прерываний. Это же касается и указателя

области USER. На программу прерывания не накладываются никаких ограничений. Не следует, разумеется, переполнять стек прерываний, все остальное разрешено.

Теперь самое время вспомнить про векторы прерываний: ' XXX CFA '100 ! (запись адреса программы прерывания в ячейку с восьмеричным адресом 100), '340 '102 ! (запись значения PS в ячейку с восьмеричным адресом 102). Теперь, если организовать прерывание один раз в минуту, то в верхней правой части экрана периодически будет печататься "Московское время ЧЧ:ММ" и одновременно будет даваться звуковой сигнал (7 EMIT).

Таким образом, программа обработки прерывания может иметь любое имя, после которого обязательно следует оператор [[. завершается программа оператором]], который возвращает управление прерванной программе, предварительно восстанавливается состояние всех регистров общего назначения. В конце описания ставится ";" никаким исполняемым операциям ";" здесь не соотвествует. Программа работает и в автономной версии Форта вне рамок какой-либо операционной системы.

При создании контроллеров внешних устройств и спецпроцессоров широко используется микропрограммирование, а это вызывает довольно серьезные проблемы комплексно-аппаратной наладки таких систем [33]. Трудности здесь связаны, с одной стороны, с использованием разнородного математического обеспечения (микроассемблеры, например ADMASM, управляющие программы внешних устройств и диагностические программы), а с другой – с не вполне отлаженной аппаратурой (поиск ошибок в микропрограммах), когда нельзя с полной уверенностью установить, аппаратная или программная ошибка имеет место. В такой ситуации гибкость и быстрая адаптация диагностического матобеспечения является привлекательным качеством.

Как можно использовать Форт в такой ситуации? В качестве примера в приложении 4 представлена тестовая программа для отладки микропрограммного 32-разрядного процессора. Процессор имеет независимую память для данных (входных и выходных) и микропрограмм. Микропрограмма может загружаться вручную (в диалоговом режиме) или из файла, подготовленного с помощью микроассемблера [45], в модуль памяти, выполненный в стандарте КАМАК. Взаимодействие между ЭВМ и системой КАМАК идентично описанному выше. Доступ к нужному файлу происходит через процедуру FILCH. Чтение файла производится с помощью оператора BLOCK по 1024 байт. По завершении чтения система возвращается в

состояние, когда именем файла по умолчанию является FORTH.DAT (оператор DEFOL). Формат записи соответствует используемому интерпретатором М1804 (AMDASM).

8.1. ГИСТОГРАММИРОВАНИЕ

При исследовании различных распределений результаты измерений имеют вид гистограмм. Гистограмма представляет собой обычно одно- или двумерный массив чисел, каждое из которых равно количеству событий, лежащих в определенном интервале по одному или нескольким экспериментальным параметрам (амплитуде, времени, частоте и пр.).

Пусть необходимо отобразить массив размером М слов (например, М=16) с именем НН в виде гистограммы. Это можно сделать, поставив в соответствие каждому числу в массиве строку символов X длиной, пропорциональной этому числу. Сначала определим максимальное число отсчетов в массиве (N MAX) и масштаб гистограммы, т. е. скольким отсчетам соответствует один символ X. Опишем оператор (HIST), который построит гистограмму на экране или печатающем устройстве. Ниже приводится текст одной из возможных реализаций:

```
16 CONSTANT K                          ( число каналов в гистограмме)

0 VARIABLE HH 32 ALLOT                    ( массив для гистограммы)

: CMAX        ( оператор для определения максимального числа
                отсчетов в каналах гистограммы)
       0 NMAX !                    ( присвоение 0 переменной NMAX)
       K 0 DO     ( начало цикла перебора каналов гистограммы
       HH I 2* + @ ( занесение в стек содержимого очередного
                     канала гистограммы)
       DUP NMAX @ >                    ( это число больше NMAX ? )
       IF NMAX !           ( если да, то меняем значение NMAX)
       ELSE DROP THEN ( в противном случае удаляем это число
                        из стека)
       LOOP ;

: FILHI            ( тестовое заполнение массива гистограммы)
       HH 40 ERASE 8 0 DO I 1+ HH I 2* + ! LOOP
            8 0 DO 8 I - HH 16 +  I 2* + ! LOOP ;

: LINE 0 DO '130 EMIT LOOP ; ( печать строки символов  X ,
                  обращение  N LINE печатает N символов  X )

: HIST CMAX ." HISTOGRAMM" CR 0 DO          ( печать заголовка
                        и начало цикла по каналам гистограммы)
       DUP I 2* + @ ( извлечение очередного числа из
                      гистограммы)
       DUP 3 .R SPACE       ( печать числа отсчетов в канале)
       60 NMAX @ */                          ( масштабирование)
       -DUP IF      ( если нужно отпечатать хотя бы один  X )
       LINE CR THEN LOOP DROP ;
```

Конечно, это примитивная гистограмма, так как здесь не указан масштаб, исходные данные и т. д., но при желании этот недостаток легко устранить. При обращении FILHI HH K HIST будет отпечатано

```
HISTOGRAMM
  1 XXXXXXX
  2 XXXXXXXXXXXXXX
  3 XXXXXXXXXXXXXXXXXXXXX
  4 XXXXXXXXXXXXXXXXXXXXXXXXXXXX
  5 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  6 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  7 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  8 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  8 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  7 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  6 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  5 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  4 XXXXXXXXXXXXXXXXXXXXXXXXXXX
  3 XXXXXXXXXXXXXXXXXXX
  2 XXXXXXXXXXXXXX
  1 XXXXXXX
```

Рассмотрим более сложный вариант печати гистограмм, когда каждому числу из указанного массива ставится в соответствие колонка (а не строка) символов X. Это более традиционный способ представления гистограмм. Здесь при выводе на печать нет ограничения на масштаб по числу отсчетов, так же как в описанном ранее методе не было ограничения на число каналов гистограммы.

Предположим, что для гистограммы зарезервировано 16 строк и 16 колонок. Практически на экране 24 – 25 строк и 60 – 132 колонки. Зарезервируем в памяти 16×16 байт. Это можно сделать с помощью оператора 0 BUFFER. Не беда, что буфер будет много больше, чем требуется, ведь мы занимаем его только на время. Заменив 0 перед BUFFER на номер свободного экрана, можно будет легко записать полученную гистограмму на диск. Опишем слово, которое заполняет таблицу 16×16 кодами пробела или X в соответствии с формой гистограммы. Масштабный коэффициент в этом случае 16/MAX. Именно такой коэффициент обеспечивает то, что каналу с максимальным числом отсчетов будет соответствовать колонка с 16 X. Ниже приводится текст описания оператора FILTAB:

```
0 VARIABLE BUF                                  ( указатель буфера)
: FILTAB CMAX ( поиск канала с максимальным числом отсчетов)
         0 BUFFER                    ( выделение временного буфера)
         DUP 16 16 * BLANKS          ( заполнение буфера пробелами)
         DUP BUF !                     ( запоминание адреса буфера)
         HH 16 0 DO                      ( начало цикла по каналам)
         DUP I 2* + @  ( занесение числа отсчетов в канале в
                         стек)
         16 NMAX @ */                              ( масштабирование)
         -DUP IF                           ( если число отсчетов $0)
         0 DO ( цикл заполнения колонки кодами символов  X )
         BUF @ 240     ( занесение в стек адреса буфера, 240=
                         16*16 -16 - адрес левого нижнего края
                       таблицы)
         + J                                    ( индекс внешнего цикла)
         + I 16 * -    ( вычисление истинного текущего адреса
                         элемента таблицы)
```

```
        '130 ( код "X") SWAP C!   ( занесение  X  в таблицу)
        LOOP THEN LOOP DROP DROP ;

: TYHI FILTAB K 0 DO BUF @ I 16 * + 16      ( размер колонки)
       TYPE CR                              ( печать строки таблицы)
       LOOP ;
```

который укладывает X в таблицу 16×16

```
                XX
                XX
               XXXX
               XXXX
              XXXXXX                    2
              XXXXXX                    3
             XXXXXXXX                   4
             XXXXXXXX
            XXXXXXXXXX
            XXXXXXXXXX
           XXXXXXXXXXXX
           XXXXXXXXXXXX
          XXXXXXXXXXXXXX
          XXXXXXXXXXXXXX
        XXXXXXXXXXXXXXXX
        XXXXXXXXXXXXXXXX
```

Оператор TYHI заполняет таблицу и печатает гистограмму в виде, приведенном выше. В этом варианте печать числа отсчетов в канале также трудоемкая задача, так как (1–4)-разрядные числа должны печататься в столбик, например так, как это показано для 234 справа от гистограммы.

8.2. ВРЕМЯ И ДАТА

При проведении экспериментов и в процессе технологического контроля важно фиксировать время и дату измерений. Для хранения информации о времени и дате в автономной версии выделено три ячейки непосредственно перед "нижней" границей словаря, первые две для времени. При работе с операционной системой можно воспользоваться системным сервисом.

При стартовой загрузке Форт-системы в ячейку с абсолютным адресом '100 (как и в ОС RT-11) записывается адрес программы обработки прерываний от внутренних часов ЭВМ. Для начальной установки часов служит оператор TIME, для оперативного отображения служит обращение к TIME TIME ЧЧ ММ СС, например TIME 9 43 00. Выходной формат данных для TI имеет вид ЧЧ:ММ:СС.

```
: C CONSTANT ; 1 C JAN 2 C FEB 3 C MAR 4 C APR 5 C MAY 6 C
      JUN 7 C JUL 8 C AUG 9 C SEP 10 C OCT 11 C NOV 12 C DEC

: TI  FENCE @ 6 -      ( вычисление адреса первой ячейки, где
                         лежит код времени)
      DUP @            ( запись в стек первого из кодов времени)
      SWAP 2+ @          ( запись в стек второго [старшего] кода
                         времени)
```

```
    50 DIV 3600 M/MOD DROP          ( в стеке число часов)
    2 .R                                  ( печать часов)
    58 EMIT                        ( печать символа ":")
    60 /MOD 2 .R                          ( печать минут)
    58 EMIT 2 .R ;                       ( печать секунд)

: TIME !CSP '                      ( фиксация состояния стека)
    INTERPRET                ( интерпретация введенного кода)
    CSP @ SP@ - DUP 4 >        ( определение и анализ числа
                                 введенных цифр)
    IF 6 = IF 0 THEN
    SWAP 60 * + 0 ROT 0 3600 MUL D+ 50 MUL      ( упаковка
                                              входных данных)
    FENCE @ 6 -  ( в стеке адрес первой из ячеек времени)
    >R R 2+ ! R> ! ( запись результатов в ячейки времени)
    ELSE SP! TI THEN ;
```

Соответствующие операторы для даты имеют имена DATE и DA. Обращение : DATE DD MON YY, например 18 JAN 89. Алгоритм работы оператора DA схож с TI. Отличительной особенностью DA является выделение имени месяца из ряда описаний констант-месяцев.

```
: DATE INTERPRET      ( интерпретация введенного кода даты)
       82 - SWAP 32 * + 32 * +
       FENCE @ CFA ! ;     ( запись результата в ячейку даты)

: DA   FENCE @ CFA
       @ 32 /MOD 32 /MOD ROT 2 .R 45 EMIT 1- 10 * 7 -
       ' JAN + 3 TYPE 45 EMIT
       82 + 2 .R ;
```

Выходной формат имеет вид, например, 18-JAN-89. Форматы хранения и представления информации о времени и дате аналогичны используемым в ОС RT-11.

При работе с ОС (в частности, RT-11) можно пользоваться системными запросами, но для этого должен быть загружен Форт-ассемблер:

```
OCTAL
CODE DATA    12 400 * # R0 MOV    374 EMT   ( системный запрос)
                    R0 S-) MOV    NEXT C;

( DATA заносит в стек код даты)

DECIMAL
: DAT DATE DUP '37 AND 72 + SWAP 32 / '37 AND SWAP 32 / '17
      AND SWAP 2 .R '55 EMIT DUP 10 < IF '60 EMIT 1 .R ELSE
      2 .R THEN '55 EMIT . ;
```

DAT извлекает, дешифрует и отображает на экране (или печатает) дату. Если сегодня 7 июля 1990 г., то при обращении DAT <ВК> ЭВМ отпечатает 7-07-90 OK.

8.3. ГРАФИКА И МУЗЫКА

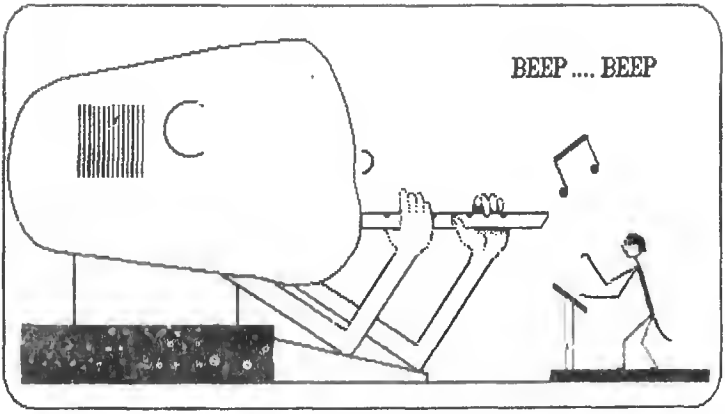
Сегодня трудно представить современную вычислительную машину без графических внешних устройств (монитор, графопостроитель, графическая печать и т. д.). К сожалению, до сих пор не существует приемлемого стандарта на входной формат данных и программный

интерфейс для этих устройств. Поэтому для каждого устройства приходится создавать свой пакет графических программ, а если сюда добавить многообразие интерфейсных схем, то станет ясно, что и для таких программных средств адаптивность является одним из важнейших качеств. В рамках системы Форт имеется специальный пакет программ для графических операций GRAFORTH. Некоторые операторы, связанные с построением простейших графических объектов, стали в Форте стандартными [19]. Любой графический образ в простейшем случае можно представить в виде совокупности точек.

Оператор для отображения точки на экране (или другом внешнем графическом устройстве) имеет имя PLOT. Обращение к нему: X Y L PLOT, где X и Y – значения координат "x" и "y"; L – код цвета/яркости точки, L=0 означает гашение изображения в данной точке. X=Y=0 соответствует нижнему левому краю графического поля либо верхнему левому в зависимости от типа графического адаптера. В любом случае X и Y – числа целые и положительные.

Другим графическим оператором является DRAW, который рисует прямые линии, обращение к нему: X Y L DRAW, где X и Y – координаты конца линий; L – код цвета/яркости, как и в случае PLOT. Линия начинается из точки, заданной текущим положением курсора. Если положение курсора не известно, а графический адаптер не позволяет его считать, то можно использовать комбинацию PLOT и DRAW, например 100 100 0 PLOT 50 50 10 DRAW. По этой команде будет нарисован диагональный отрезок прямой. Если теперь выдать команду 100 100 0 DRAW, то этот отрезок прямой будет стерт. В некоторых реализациях параметрами оператора DRAW являются приращения координат X и Y по отношению к текущему положению курсора. Структура L-кода индивидуальна для различных графических адаптеров, поэтому прежде чем пользоваться графическим пакетом, который вы приобрели, необходимо произвести его настройку под имеющийся адаптер.

В графическом пакете программ обычно имеются операторы для включения-выключения графического режима, например G-MODE и T-MODE (графический и текстовый режимы). Полезным оператором может оказаться LABEL, который временно переключает монитор в текстовый режим и печатает строку текста длиной L. Обращение к нему: ADR L X Y LABEL, где ADR – адрес первого байта текста; X и Y – координаты положения первого символа в строке. Нетрудно описать слова, перемещающие курсор в графическом режиме вправо, влево, вверх или вниз на один стандартный шаг (RIGHT, LEFT, UP и DOWN), а также операторы ON и OFF, которые включают и выключают режим



рисования, т. е. при ON любое перемещение курсора приводит к вычерчиванию отрезка прямой. Размер шага можно задать переменной STEP.

Изображение можно сопровождать музыкой. Для этого введен оператор BEEP. Обращение: F D BEEP, где F – частота, Гц; D – длительность звучания в некоторых единицах, например в секундах. Простые мелодии можно описать массивом, состоящим из определенного количества пар чисел, первое задает длительность, а второе – частоту. Вне зависимости от того, как реализуется воспроизведение, через программируемый таймер или через звукосинтезатор, исходная запись мелодии описывается последовательностью чисел. Ниже приводится текст программы (TUNE), которая исполняет первые такты мелодии "Подмосковные вечера":

```
1 VARIABLE TU 6223 , 1 , 5232 , 1 , 4153 , 1 , 5232 ,
2 , 4662 , 1 , 5232 , 1 , 5532 , 2 , 4153 , 2 , 4662 ,
2 , 6223 ,
3 CONSTANT ADJ ( подстройка частоты)
: FREQ ADJ / 182 67 PORT DUP '377 AND 66 PORT 256 / 66 PORT ;
: WAIT 0 DO I DROP I DROP LOOP ;
: NOTA FREQ 15000 * WAIT ;
: TUNE 97 READ 3 OR 97 PORT 20 0  ( воспроизведение мелодии)
      DO TU I 2* + DUP @ SWAP 2+ @ NOTA 2
      +LOOP 97 READ '177774 AND 97 PORT ;
```

Предполагается, что интерпретатор имеет процедуры PORT и READ. Первая из них берет из стека номер порта (P) и посылает туда байт (B), лежащий в стеке. Обращение: B P PORT. Оператор READ считывает состояние порта, номер которого находится в стеке. Результат записывается в стек. Для записи мелодии описываем массив TU, который содержит в четных словах длительность звука, а в нечетных – частоту.

Программа использует программируемый таймер персональной ЭВМ (порт готовности – 67, а порт загрузки – 97). Слово FREQ устанавливает частоту звукового сигнала. В качестве исходного параметра FREQ берет из стека коэффициент деления частоты задающего генератора. Оператор NOTA задает частоту и длительность звука, обращение: T D NOTA, где T=1,2,3,... – множитель, задающий длительность сигнала; D – делитель частоты, определяющий частоту звучания. Последовательно задавая частоту и длительность сигнала, можно воспроизвести нужную мелодию. По завершении цикла воспроизведения необходимо выключить звук, для этого сбрасываются в нуль два младших разряда регистра, управляющего динамиком (порт 97).

8.4. ТЕЛЕИГРЫ

Эффективность Форта при решении графических задач предопределяет широкое его использование при написании телеигр [5]. Это прежде всего динамичные игры, где требуется скорость реакции. Событие в таких играх происходит, как правило, на фоне статических декораций (лабиринты, границы площадок и полей, иллюстративный фон, меню и справочные данные, например счет и т. д.). Эти декорации можно заменять частично или полностью, но делается это сравнительно редко. К другой категории относятся подвижные объекты, положение которых вычисляется программой или задается игроком. Причем результаты работы программы зависят от сложившейся ситуации и предшествующих действий игрока (или игроков). Поэтому такие телеигры – это логические интерактивные программы, результат работы которых отображается в виде графических образов. Знакомство с этим разделом полезно, даже если читатель телеиграми не интересуется, так как сходные проблемы возникают в системах автоматизации проектирования, в тренажерах (а это уже настоящие телеигры!), а также в системах управления, работающих в реальном масштабе времени, где реакция и скорость отображения реальной ситуации на экране должны быть максимально быстрыми (пульт управления атомной электростанцией, ускорителем или другой аналогичной установкой, машинное моделирование сложных процессов). Одним словом, проблемы, решаемые при создании телеигр, весьма серьезны.

Условно программы для телеигр можно разделить на две части. Первая анализирует ситуацию, воспринимает прерывания с клавиатуры, игрового манипулятора или "мыши", рассчитывает движение объектов с учетом ограничений. Вторая представляет ситуацию на экране кадр за кадром, генерирует звуковое сопровождение, выдает сообщения (например, счет игры) и подсказки игроку. Решение второй

задачи сильно зависит от типа дисплея ЭВМ (алфавитно-цифровой или графический, черно-белый или цветной, имеется ли возможность полифонического звукового синтеза и т. д.). Характерной особенностью программ телеигр является многозадачность – отображение информации, звуковое сопровождение, обработка прерываний и расчет новых положений объектов ("мяча", "врагов" или помех).

Любой кадр содержит статическую составляющую, на воспроизведение которой время процессора и канала связи не занимается, и переменную. Последняя должна быть объектом особых забот, так как скорость обмена с дисплеем невелика.

Посмотрим, как в обобщенном виде выглядит структура программы телеигры:

```
: TEST POSITION LIMITS-TEST ;
: GO BEGIN LOOK_AT_LIST INPUT TEST CHANGES 0 UNTIL ;
: GAME SET_START_STATUS GO ." GAME IS OVER " CR ;
```

POSITION вычисляет положение объектов на игровом поле;

TEST сверяет вычисленное положение объектов с игровой обстановкой и в случае нарушения условий (LIMITS-TEST) прекращает игру;

LOOK_AT_LIST выбирает в соответствии с положением объектов очередные условия из списка и отображает их на игровом поле;

SET_START_STATUS задает и отображает начальные условия игры;

INPUT ожидает ввода с терминала, кодирует и засылает результат в стек;

CHANGES изменяет обстановку согласно алгоритму и в соответствии с внешними воздействиями.

Цикл GO бесконечный, но если внутри оператора TEST будет выполнена команда R> DROP, то по завершении процедуры ;S управление передается не CHANGES, а оператору, следующему за GO в описании GAME.

Рассмотрим, как это делается, на примере игры "Шар в коробке". Алгоритм этой задачи является основой многих игровых программ типа "теннис", "баскетбол", "футбол", "сквош". Ниже приводится программа, которая рассчитана на работу с дисплеем СМ7209:

```
: $ VARIABLE ;   1 $ DY   1 $ DX  ( значения шагов по Y и X)
: ESC 27 EMIT ;                   ( выдача кода ESC на терминал)
: GRAF ESC 70 EMIT ;                    ( вход в псевдографику)
: RC 13 EMIT ;          ( возврат каретки без перевода строки)
: FIX RC '54433 PAD ! PAD 2+ ! PAD 4 TYPE ;        ( фиксация
                                                положения курсора)
: LINE 80 0 DO DUP EMIT LOOP CR ;            ( рисование линии)
```

```
: FRAME GRAF CR LINE 21 0 DO DUP EMIT 78 SPACES
  DUP EMIT CR LOOP LINE ;                    ( рисование рамки)

: CONTROL R# @ DUP 255 AND DUP 52 >  ( управление положением
                                       курсора)
    IF DROP -1 DY ! RC ( смена знака приращения, если курсор
                         достиг "дна")
    ELSE 34 <
       IF 1 DY ! RC THEN  ( смена приращения DY, если курсор
                            достиг "потолка")
    THEN 256 / DUP 109 >
    IF DROP -1 DX ! RC    ( смена приращения DX, если курсор
                            достиг правого края)
    ELSE 34 <
       IF 1 DX ! RC THEN   ( смена DX, если курсор достиг
                            левого края)
    THEN ;

: SCL '20040 FIX ESC 74 EMIT ;   ( очистка [гашение] экрана)

: NR ESC 71 EMIT ;          ( возвращение дисплея к нормальному
                              режиму)

: STEP R# @       ( запись в стек текущей координаты курсора)
      FIX           ( установка курсора-шара в нужное место)
      SPACE                ( гашение изображения курсора)
      DX @ 256 * DY @ + R# +! ( вычисление новой координаты
                               курсора)
      R# @ FIX 97 EMIT        ( перемещение курсора в новое
                                положение и его отображение)
      CONTROL ;            ( контроль границ и смена знака DX
                             и/или DY)

: TTT SCL '22043 R# ! '141 FRAME        ( отображение рамки)
      BEGIN STEP ?TERM '105 =       ( нажата клавиша <E> ? )
      UNTIL NR ;
```

При реализации алгоритма на персональной ЭВМ IBM PC программа несколько упрощается (так как операторы FIX и SCL присутствуют в базовом словаре).

Слово FRAME рисует рамку размером 80×23 (рабочая зона 78×21 знакомест), GRAF переводит дисплей из обычного в псевдографический режим (NORM возвращает его в исходное состояние).

Переменные DX и DY хранят значения приращений по оси X и Y. FIX перемещает курсор в точку с заданными координатами, а SCL – гасит экран в начале игры.

TTT производит запуск игры и после подготовки входит в бесконечный цикл шагов (STEP), выход из которого возможен, если нажать клавишу E (END).

Положение курсора задается системной переменной R#, используемой для той же цели в экранном редакторе. Курсор (шар) перемещается в "коробке" прямолинейно ($|\Delta X| = |\Delta Y| = 1$), при достижении стенки коробки он отражается под прямым углом.

Чтобы превратить эту программу в реальную игру можно, например, ввести две вертикально расположенные "ракетки" в правой и

левой части коробки, перемещаемые вверх и вниз с помощью манипуляторов или двух пар клавиш терминала. Надо будет заставить "шар" отражаться не только от стенок, но и от "ракеток". "Ракетка" может рассматриваться как фрагмент стены с переменными координатами. Нажатие той или иной клавиши может контролироваться так же, как это делается для клавиши E в приведенном тексте. Разумеется, это лучше делать через прямой "перехват" прерываний при нажатии соответствующих клавиш. При работе в условиях более сложной геометрии (например, лабиринты) в памяти формируется двумерная таблица (см. описание игры "Жизнь"). Все объекты перемещаются из ячейки в ячейку этой таблицы. В байтах таблицы фиксируется, что там находится ("игрок", "враг и его тип", "стенка" и ее свойства и т. д.). Если объект по своим размерам не помещается в одной ячейке на экране и имеет сложную структуру, тогда отдельная программа отображает этот объект, "привязав" его положение к координатам, соответствующим положению оговоренной ячейки. На основе представленного текста, дополнив его и усложнив, вы без труда сможете создать широкий спектр игр.

Игра "Жизнь", предложенная Дж. Конвеем, заключается в описании зарождения, жизни и смерти простых объектов согласно простым правилам. "Среда обитания" объектов – прямоугольное поле из m×n клеток (существуют и трехмерные версии игры). Сам объект – это одна клетка в таблице. Если объект имеет двух или трех живых соседей, то и он остается живым в следующем поколении (на очередном кадре). Объект, имеющий 0, 1 или более 3 живых соседей, умирает (в последнем случае из-за "перенаселения"). Соседями считаются клетки, имеющие друг с другом хотя бы одну общую точку границы. Если же "мертвый" объект имеет трех живых соседей, то он "оживает". Вот и все правила. Остается описать такую таблицу, "заселить" ее и наблюдать, как она будет эволюционировать. Одна из Форт-реализаций игры "Жизнь" представлена в [5].

Чтобы описать таблицу, введем оператор TABLE, который может пригодиться и для других целей. В сущности это упрощенный оператор 2ARRAY (см. табл. 22).

```
: TABLE <BUILDS                    ( задание имени таблицы)
        OVER ,                ( ввод в описание таблицы ее длины)
        * ALLOT           ( резервирование нужного объема памяти)
        DOES>             ( начало программы, работающей в режиме
                            исполнения для любой вновь описанной
                            таблицы)
        DUP @               ( в стеке адрес начала таблицы и ее
                              длина в байтах)
        ROT               ( перенос кода номера столбца на верх
                            стека)
```

```
        * + + 2+ ;      ( вычисление адреса элемента таблицы с
                          заданными номерами столбца и строки)
```

Обращение к слову TABLE производится, например, в форме: NX NY TABLE AREA, где NX – длина строки; NY – длина столбца; AREA – имя таблицы. Обращение к AREA: R C AREA, где R – номер ряда; C – номер колонки. По сравнению с 2ARRAY здесь нет проверки того, что элемент находится в пределах границ таблицы. Вы можете дополнить описание TABLE таким контролем. Для этого надо записать в список параметров число рядов, введя между "," и "*" команду "DUP ,". Программа проверки должна быть встроена в текст после оператора DOES> (в описание TABLE). Текст остальной программы представлен ниже:

```
1001 VARIABLE RS ( заготовка для генератора случайных чисел)
40 CONSTANT DX                ( размер таблицы по горизонтали)
8 CONSTANT DY                   ( размер таблицы по вертикали)
DX DY TABLE AREA                      ( описание поля таблицы)
: SHOW '20040 R# ! DY 0       ( отображение таблицы на экране)
      DO DX 0                                  ( цикл по строкам)
         DO                                   ( цикл по столбцам)
           I J AREA C@          ( извлечение кода из таблицы)
           DUP 2 AND
            IF DROP
            ELSE R# @ I SWAB + FIX     ( фиксация положения
                                         курсора)
               1 AND  ( проверка наличия "живого" объекта)
               IF '141 EMIT                ( отображение ■)
               ELSE SPACE ( если в данном месте нет живого
                            объекта)
               THEN
            THEN
         LOOP 1 R# +!       ( перевод указателя на следующий
                              объект)
      LOOP ;
: NORM DY 0 DO DX 0
             DO I J AREA DUP C@ DUP 128 AND
                IF 1 XOR 1 AND ELSE 2 OR THEN
                3 AND SWAP C!
             LOOP
          LOOP ;
: TEST DY 0 DO DX 0       ( оператор для определения и пометки
                            "живых" и "мертвых" объектов)
           DO 0 J 2+ DY MIN J 1- 0 MAX
             DO J 2+ DX MIN J 1- 0 MAX
               DO I J AREA C@          ( запись в стек кода
                                         объекта)
                  1 AND     ( выделение флага "жив/мертв")
                  +              ( подсчет "живых" соседей)
               LOOP
             LOOP DUP I J AREA DUP C@ 1 AND
                  IF ( если был "жив")
                    ROT 3 < ROT 4 > OR          ( проверка
                                  условия выживания объекта)
```

```
                    IF DUP C@ 128 OR SWAP C!
                       (установка флагов в байте объекта)
                    ELSE DROP
                    THEN
                  ELSE                ( если объект "мертв")
                     SWAP 3 =
                     IF             ( если выполнено условие
                                      "воскрешения")
                       DUP C@ 128 OR SWAP C! DROP
                     ELSE 2DROP
                     THEN
                  THEN
              LOOP
            LOOP ;

: RND RS @ 2725 U* 13947 S->D D+ DROP DUP RS ! 0 > ;
: INIT DY 0 DO DX   0 DO RND I J AREA C! LOOP   LOOP ;
: LIFE GRAF INIT                ( заполнение таблицы с помощью
                                  генератора случайных чисел)
       SHOW              ( отображение результата заполнения на
                           экране)
       BEGIN                    ( начало бесконечного цикла)
       TEST              ( проверка состояния объектов и пометка
                           "живых" и "мертвых")
       SHOW NORM          ( коррекция таблицы с использованием
                           меток, расставленных оператором TEST)
       ?TERM 69 =               ( с терминала введена буква E?)
       UNTIL NR ;          ( если нет, то продолжение цикла, в
                           противном случае прерывание работы)
```

Функции операторов GRAF, NR, ESC и FIX описаны выше. RND – примитивный генератор случайных чисел, используемый в некоторых библиотеках FIG-FORTH и дополненный операторами, которые позволяют формировать "случайные" последовательности 0 и 1. Переменные DX и DY задают размеры таблицы (40 колонок и 8 рядов). Оператор SHOW – хороший пример использования слов I и J для вложенных циклов. Здесь J во внутреннем цикле позволяет пользоваться индексом внешнего цикла. ›141 EMIT выдает в нужном месте экрана символ ■, если там был пробел, в противном случае туда заносится пробел. Каждая ячейка таблицы представляет собой байт, младший бит которого указывает, "жив" объект или "мертв" (1 соответствует состоянию "жив"). Следующий по старшинству бит используется для того, чтобы указать, изменилось ли состояние объекта по сравнению с предшествующим (1 соответствует отсутствию изменения). Дело в том, что из соображений быстродействия на дисплей передаются только изменения картинки, а не вся картинка. Старший бит байта =1 отмечает тот факт, что состояние "жив/мертв" данного объекта должно быть изменено. После определения "судьбы" объекта сразу нельзя изменять его состояние, так как "старая" информация нужна для выяснения "будущего" других объектов (оператор TEST). Если использовать две табли-

цы вместо одной, можно избежать этих трудностей и заметно ускорить работу программы.

Наблюдение за эволюцией таблицы показывает, что мало-помалу там остается только два вида образований: "маяки" и "стационарные станции". Примером последней может служить группа вида

■■ , где у каждого объекта три "живых" соседа. Примером "маяка" является группа

Все элементы этого "кольца" имеют двух "живых" соседей, центральный элемент будет попеременно появляться и исчезать.

Упражнение 1. Пусть имеется модуль аналого-цифрового преобразователя (АЦП), выполненного в стандарте КАМАК и установленного в станции 4 каркаса. По команде F2 этот модуль посылает в ЭВМ 10-разрядный код, полученный в результате преобразования, а выходной регистр сбрасывает в нуль. Предполагается, что КАМАК-интерфейс имеет структуру управляющих регистров, описанную выше. По завершении преобразования модуль выдает запрос обслуживания, который вызывает прерывание. Опишите программу HISTO, которая в случае прерывания считывает код из АЦП, выбирает элемент массива HI, соответствующий этому коду, и добавляет к числу, лежащему в указанном элементе, 1. Массив HI описан с помощью команды 0 VARIABLE 2048 ALLOT.

Решение. Предположим, в CSR интерфейса ЭВМ записан код команды КАМАК F2 (выполнена команда 2 CSR !), а КАМАК-адрес АЦП имеет форму константы с именем ADC (4 MODL ADC):

```
: HISTO [[ 1 ADC    ( в стеке прерываний 1 и КАМАК-адрес АЦП)
        @            ( запись в стек кода, считанного из АЦП)
        HI +       ( вычисление адреса, номер которого должен
                     быть увеличен на 1)
        +! ]] ;
```

Если требуется контроль считанного кода, оператор HISTO можно видоизменить:

```
 40 CONSTANT PEDESTAL ( размер пьедестала) 900 CONSTANT TOP
 : HISTO [[ 1 ADC @ PEDESTAL - 0 MAX      ( если значение кода
                меньше пьедестала событие заносится в 0-канал)
          TOP MIN    ( если код превышает значение TOP, событие
                       заносится в канал с номером TOP)
          HI + +! ]] ;
```

Чтобы проверить наличие модуля в нужном месте, достаточно выполнить команду TEST_P:

```
: TEST_P ADC @ DROP CSR @ '100 AND 0=      ( проверка наличия
                                             отклика "X")
     IF ." NO "X"-response of MODULE" THEN ;
```

Упражнение 2. Напишите программу для вычерчивания окружностей, эллипсов и дуг.

Упражнение 3. Вспомните детство и нарисуйте домик.

Упражнение 4. Если вы хорошо рисуете, попытайтесь нарисовать автопортрет, если плохо, то лицо человека, который вам наименее симпатичен.

Упражнение 5. Используя в качестве основы программу "Подмосковные вечера", составьте программу для какой-либо другой мелодии.

Упражнение 6. Попытайтесь переделать программу "Шар в коробке" в игру типа "теннис". В простейшем случае для перемещения "ракеток" используйте клавиши <↑> <↓> для одного игрока и <Q> и <A> для другого. Для ускорения перемещений "мяча" перепишите основные процедуры, используя Форт-ассемблер.

Глава 9. Версии Форта

Существует целое семейство Форт-интерпретаторов для самых разных машин, причем большая часть для персональных ЭВМ. Многие из них в своем наборе слов отклоняются от известных стандартов. Некоторые отличаются заметно, другие имеют несколько слов сходного назначения, но разного наименования (например, WORDS и VLIST), существуют и такие, которые идентичны версиям FIG-FORTH или Форт-79, но дополнены специальными операторами, например для работы с графикой в реальном масштабе времени. Наибольшее распространение имеют версии Форт-79, Форт-83 и FIG-FORTH, хотя существуют и другие (MMSFORTH, MVPFORTH, F83, HS/FORTH, Полифорт и т. д.). Отмечу ряд отличий Форт-79 и Форт-83 (FIG-FORTH довольно близок к Форт-79). В Форт-79 числа в стеке нумеруются начиная с 1, а в Форт-83 – с нуля, поэтому команда 1 PICK в первом случае эквивалентна DUP, а во втором – OVER. Имеются отличия в контроле завершения циклов DO...LOOP, способе округления частного при делении, диагностике ошибок, форматах представления чисел операторами .R, D.R и т. д. Отличаются ограничения, накладываемые на число слов (байт), которые пересылаются операторами MOVE (CMOVE или <CMOVE). Если в вашей версии отсутствуют те или иные слова, вы без большого труда можете ее дополнить. Проблемы возникают при адаптации библиотек и пользовательских программ, написанных в рамках "чужого" стандарта из-за слов, которые имеют идентичные имена, но принципиально разные функции. Назовем лишь наиболее важные.

Оператор '(апостроф) <name> в Форт-79 выдает в стек PFA слова <name>, а Форт-83 – CFA. Ряд слов имеют различия в поведении при компиляции и исполнении. Так, ." XXX" в Форт-79 применимо как при компиляции, так и при исполнении, в Форт-83 только при компиляции

(для Форт-83 см. также .(XXX)). Оператор EXPECT в Форт-79 заносит в конец строки два нулевых байта, а Форт-83 заменяет <CR> пробелом. Совершенно по-разному работает слово FIND. В Форт-79 обращение к нему имеет вид: FIND <name>, а стек преобразуется как (-→ адрес (или 0 в случае неудачного поиска)). В Форт-83 схема преобразования стека имеет вид (адр1 → адр2 n), где адр1 – адрес счетной строки, содержащей имя искомого слова; адр2 – адрес этого слова в словаре; n= –1 при успешном поиске для обычного слова, n=1 для слов немедленного исполнения. При неудачном поиске адр1=адр2 и n=0. (Более детальное описание читатель найдет в текстах стандартов или в книге [14].)

Причины этих различий отчасти в сильном отличии наборов команд ЭВМ, на которых реализованы эти интерпретаторы (например, фирм DEC и IBM), отчасти в особенностях их операционных систем и организации периферийного оборудования. Немалый вклад в это разнообразие вносится также простотой написания и модификации Форт-интерпретаторов. Проблемы совместимости весьма остры при написании программ ввода-вывода.

Персональные ЭВМ фирмы IBM (отечественный аналог ЕС1841) имеют дисплей, функциональная часть клавиатуры которого отличается от известных терминалов фирмы DEC (ЭВМ типа СМ и "Электроника-85"). Кроме того, командные клавиши их дисплеев формируют принципиально другие коды. Работа функциональной клавиатуры ЭВМ типа СМ основана на применении ESC-последовательностей (строк кодов, начинающихся с числа 27, различной длины), а IBM PC использует однородную систему кодов (нажатие любой клавиши передает на вход ЭВМ два байта информации). Еще значительнее различие при работе с графикой. Перечисление этих отличий можно было бы и продолжить. К счастью, имеется возможность некоторыми ухищрениями сделать эти различия малозаметными для прикладного программиста. Разумеется, отличие в клавиатуре неустранимо, и здесь приходится рассчитывать на терпение и быструю адаптацию пользователя.

При разработке экранного Форт-редактора программист сталкивается с дилеммой: стремиться ли к максимальной совместимости со старыми версиями или обеспечить совпадение с функциями штатных редакторов для IBM PC (например, EDT). Автор пошел по второму пути, в частности использовал для смены направления движения курсора клавиши <4> и <5>, а для непосредственного перемещения курсора клавиши F5, F7, F8, F9. При реализации экранного редактора для персональной ЭВМ пришлось столкнуться с тем, что в нормальном режиме клавиша <5> не дает отклика (не передает каких-либо кодов

при нажатии). Здесь возможно несколько решений, использовано временное переключение вспомогательной клавиатуры в числовой режим, для чего служит байт 417H на нулевой странице памяти ЭВМ. В версии Форта для IBM PC можно встретить операторы, схожие по функции с KEY, но выдающие в стек оба байта, которые характеризуют нажатую клавишу. Почти неизбежны отличия вариантов Форта, использующих область памяти более 64 Кбайт для 16-разрядных ячеек памяти, так как стандартов на страничную адресацию (сегментацию) в Форте также пока не существует.

Имеются версии Форта, где для редактирования виртуальных файлов типа FORTH.DAT используются стандартные экранные редакторы операционной системы. Эти варианты могут работать с текстами, содержащими символы <CR> (возврат каретки). Персональные ЭВМ благодаря использованию блоков управления файлами (FCB) позволяют применять практически любое число виртуальных файлов прямого доступа, что затруднено для микроЭВМ семейства ДВК и СМ, где для Форта обычно доступно не более двух виртуальных файлов.

Многие Форт-интерпретаторы имеют модульную структуру и практически все написаны на Ассемблере. Каждый модуль выполняет строго заданную функцию, поэтому перенос интерпретатора с одной ЭВМ на другую даже при сильном отличии их наборов команд, структуры памяти и операционных систем большого труда не составляет. Конечно, знание ассемблеров и ОС ЭВМ при этом совершенно необходимо. С этой точки зрения Форт привлекателен при написании математического обеспечения вновь создаваемых оригинальных микропроцессоров, ведь для Форта даже наличие операционной системы не обязательно. В этой сфере применения Форт конкурирует с системой UNIX [24].

Наметилась тенденция к стандартизации некоторых графических процедур [19], впрочем эта проблема выходит за рамки Форта и носит межязыковый характер.

В последнее время просматриваются тенденции по созданию процессоров, ориентированных на тот или иной язык (Фортран, Си, Ада и т. д.). Коснулась эта тенденция и языка Форт. Система Форт легко адаптируется к аппаратной реализации, ведь стековые структуры – весьма характерные устройства современных ЭВМ и микропроцессоров. Работа с аппаратным стеком обеспечивает достаточно высокое быстродействие; исключается обращение к оперативной памяти. Если аппаратно осуществлять переход от одной Форт-процедуры к другой, а также применить схемную реализацию стека возвратов, можно обеспечить еще бóльшую скорость выполнения операций. Форт-процессор

может успешно использовать RISC-архитектуру (ЭВМ с сокращенным набором команд), получившую широкое распространение в ЭВМ новых поколений [41]. Форт-процессор, созданный фирмой Silicon Composers (США) в 1986 г., это одноплатная ЭВМ PC400, построенная на базе 16-разрядной интегральной схемы NC400 (компания NOVIX Inc., Купертино, шт. Калифорния), непосредственно выполняет инструкции языка Форт [40], обеспечивая быстродействие 10 млн. опер./с. Компания VME Inc. выпустила плату, в которой объединены Форт-процессор и двухпортовая статическая память емкостью 128 Кбайт. Плата обеспечивает производительность в 5–20 раз больше, чем Motorola 68020, Intel 80386 и VAX 11/780. Разработчик языка Форт Ч. Мур создал на базе новой микросхемы NC400 графическую плату, способную переключаться с одного изображения на другое в пределах одного кадра.

Основное направление применения Форт-процессора – экспертные системы реального времени, где система управляет процессом, а не просто дает советы, а также системы искусственного интеллекта. Такая система уже создана компанией FORTH Inc. (Гермоза-Бич, шт. Калифорния). Существуют Форт-сопроцессоры, которые можно использовать в персональной ЭВМ, повышая ее производительность. Рассматриваются пути адаптации PolyFORTH [15] для аппаратного Форт-процессора. Созданы трансляторы с языков Си и Пролог для работы с Форт-процессором. Ведутся такие работы и в СССР (в Эстонии) [47], там также созданы образцы Форт-процессоров.

9.1. АВТОНОМНАЯ ВЕРСИЯ ФОРТА

Так как Форт имеет многие черты операционной системы, возможно его автономное использование. Но для этого нужно, чтобы в процессе автоматической загрузки системы с магнитного носителя в память была перенесена программа загрузки Форт. Для этого она должна быть заранее записана. Кроме того, двоичный образ словаря должен быть записан на диск, это можно выполнить с помощью оператора BILOAD:

```
BILOAD:

: BILOAD DUP R# ! DO I R# @ - 1024 * ( откуда)
                    I BLOCK ( куда) 1024 CMOVE UPDATE
                 LOOP ( завершение переноса) FLUSH ;
```

Обращение: M N BILOAD, где M – номер конечного экрана; N – номер первого экрана, куда записывается двоичный образ словаря. Программы первичной и вторичной загрузки должны быть написаны в машинных кодах, для чего следует воспользоваться Форт-ассемблером. Программа первичной загрузки записывается в зону носителя, опреде-

ленную начальным загрузчиком. Программа вторичной загрузки записывается туда же, где хранится двоичный образ Форта. Если выполнить эти условия, то при включении ЭВМ Форт будет загружен автоматически без участия операционной системы.

Автономная версия Форта особенно эффективна на микроЭВМ малых конфигураций, где емкость дисков невелика. В таких ЭВМ операционная система занимает один диск практически целиком. При работе с языками высокого уровня, например Фортраном, на диске почти не остается места для программ пользователя. Автономная версия Форта занимает на диске не более 8 Кбайт, для Форт-библиотеки требуется еще несколько килобайт, для Форт-ассемблера – 5 Кбайт, для библиотеки работы с числами с плавающей точкой 4–5 Кбайт, для экранного редактора 4 Кбайт и т. д. Итого, системными Форт-программами на диске занято менее 30 Кбайт, остальное пространство остается для прикладных библиотек и программ пользователя. Ни одна другая операционная система не предоставит такой возможности.

Автономная версия Форта зависит от особенностей периферийного оборудования, поэтому неизбежно возникают трудности при ее переносе даже на ЭВМ с совместимым набором команд, но другим контроллером магнитного диска. Характерной особенностью автономной версии является и то, что диск для Форта представляет собой однородную среду, разбитую на зоны по 1 Кбайт (экраны). Оглавление диска не используется, и в этом случае файловая структура диска для системы не имеет никакого смысла. Диск может выглядеть нечитаемым на ЭВМ, где загружена обычная операционная система, ведь автономный Форт может использовать для своих целей и зону оглавления диска, хотя это и нежелательно. Но "нечитаемое" оглавление может использоваться для предотвращения несанкционированного копирования программ. Копирование дисков в этом варианте должно проводиться поблочно. Возможно копирование с диска на диск и без выхода из системы Форт. Например, для мини-дисков (с MX0: на MX1:):

```
: COPYD 40 0 DO HERE I 0 RTS ." TR=" I . CR I 40 +
                TRACK WRTR
          LOOP ;
```

RTS и WRTR, TRACK – операторы чтения и записи трека и перехода на соседний трек.

Разработанные автономные версии предоставляют дополнительные удобства, например управление выдачей на экран или печать с помощью команд <^Q>, <^O>, <^S>, простой способ возврата в Форт-систему с помощью <^C>, в то время, как операционная система возвращает управление терминальному монитору, удаляя Форт из

памяти. ^C ^C-перехват, предусмотренный в ОС RT-11, позволяет решить ту же задачу, введя специальный оператор в рабочую программу, но это заметно замедляет ее исполнение. Пожалуй, только автономная версия Форта пригодна для небольшого стенда для испытания электронной аппаратуры или других измерений, где объем поступающих данных невелик. Для таких задач при работе с Фортом хватит даже одного гибкого мини-диска с одинарной плотностью записи.

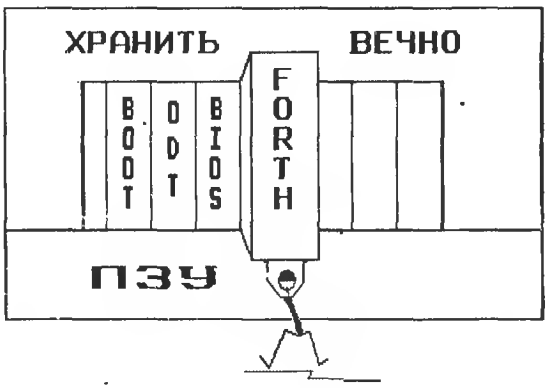
Пример текста программы для формирования автономной версии для гибкого мини-диска представлен в приложении 6.

Хотя автономная версия может работать, не пользуясь оглавлением диска, во многих случаях целесообразно сохранить файловую структуру. Это позволяет использовать и копировать программы с диска на диск, когда эти программы сформированы с помощью различных версий Форта. Чтение оглавления диска (DIRECTORY) из системы Форт не предусмотрено, что создает определенные трудности, так как для знакомства с содержимым диска приходится выходить из системы Форт, а в автономном варианте загружать операционную систему.

Программа чтения оглавления диска (приложение 7), находясь в системе Форт (предполагается, что файловая структура соответствует системе RT-11). Оператор RASC преобразует N слов RAD50, лежащих начиная с адреса ADR1, и укладывает последовательность кодов ASCII по адресу ADR2 и далее, обращение: ADR1 ADR2 N RASC. В программе DIR предполагается, что имена 12 месяцев описаны ранее в виде 1 CONSTANT JAN 2 CONSTANT FEB 3 CONSTANT MAR и т. д.

9.2. ФОРТ В ПОСТОЯННОМ ЗАПОМИНАЮЩЕМ УСТРОЙСТВЕ

Пользователям, имеющим в своем распоряжении хорошо оснащенную ЭВМ и работающему в комфортных условиях, трудно понять



необходимость записи Форт-интерпретатора в постоянное запоминающее устройство (ПЗУ) (этот раздел они могут пропустить).

Рассмотрим некоторые преимущества Форт в ПЗУ:

устойчивость системы к помехам в сети и наводкам;

экономия оперативной памяти (4–8 Кбайт в случае ОС RT-11[1] [38]) и места на диске (места на диске, как денег, всегда не хватает).

появляется возможность наладки контроллеров внешних устройств с использованием языка высокого уровня, когда в ЭВМ отсутствуют какие-либо внешние устройства долговременной памяти, а следовательно и ОС.

Последнее свойство таких систем позволяет использовать их также для целей диагностики ЭВМ с неисправной периферией.

При подготовке версии интерпретатора и прикладных Форт-программ, предназначенных для записи в ПЗУ, нужно следить, чтобы в теле программы не было модифицируемых ячеек, т. е. все массивы и переменные должны быть вне области ПЗУ. Для ЭВМ типа "Электроника" и СМ следует избегать применения команд MTPS, MUL, DIV, ASH и ASHC, оперирующих с ячейками памяти, так как они пытаются модифицировать содержимое этих ячеек, а это с неизбежностью вызовет отсутствие отклика и остановку (прерывание) процессора. Возможно решение, где плата ПЗУ обеспечивает фиктивный отклик на попытку записи.

Один из вариантов размещения системы Форт в памяти для работы с ПЗУ изображен на рис. 10. Интерпретатор помещен на месте, где обычно находится RMON, "хандлер" системного устройства и программа USR (работа с файлами).

Определенные трудности возникают с описаниями слов ;CODE, FORTH и END (конец базового словаря), так как некоторые ячейки этих описаний должны модифицироваться (FORTH при инициализации, а ;CODE при загрузке Форт-ассемблера). Поэтому при запуске системы эти слова-описания переносятся в область SOFE, одновременно автоматически редактируются их константы связи. Как видно, словарь новых описаний начинается с ячейки 1000 (восьмеричная). Расположение стеков, экранных буферов и т. д. является традиционным.

Данная версия Форт-ПЗУ может занимать 8 Кбайт, позволяет работать с гибким диском, обрабатывать прерывания (см. гл. 8), по сравнению с FIG-FORTH она дополнена словами-операторами S., O., COPY, TY, C,, DUMP, STY и SWAS (их описания приведены в гл. 4, 5 и 7), что делает эту автономную версию более эффективной. Хотя Форт-ПЗУ

[1] В этом разделе везде подразумевается ОС RT-11.

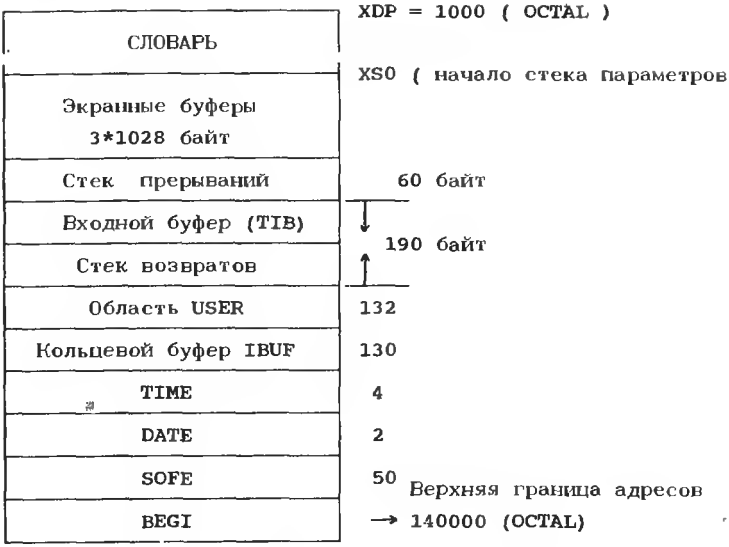


Рис. 10. Размещение секций Форта в ПЗУ

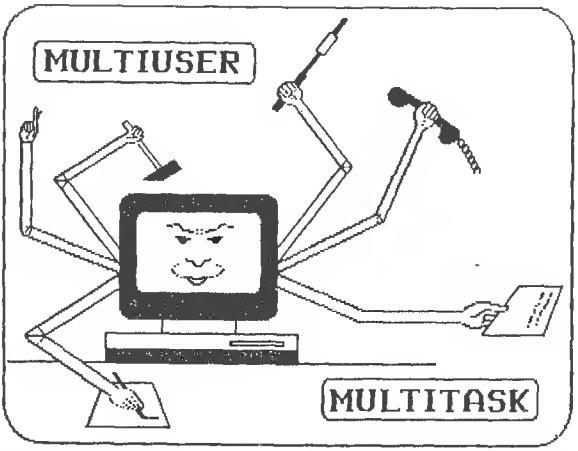
работает с гибким диском, ОС ему не нужна и весь объем дискового пространства можно использовать исключительно для хранения программ. Практика показывает, что благодаря компактности Форт-программ одного 5-дюймового гибкого диска с одинарной плотностью записи вполне достаточно для хранения управляющих программ довольно сложных измерительных комплексов.

Форт-ПЗУ может быть полезным и эффективным там, где набор слов фиксирован (бортовые программно-управляющие средства, медицинские мониторные системы), а также в дешевых персональных ЭВМ с ограниченной памятью и дисковым пространством.

Возможно два представления Форт-программ – стандартное и компактное, лишенное имен и кодов связей. Второе требует на 20–30% меньше памяти и работает на 25–45% быстрее, но исключает дополнение словаря новыми описаниями. Стандартное представление сохраняет для пользователя все возможности системы Форт без изъятия.

9.3. МНОГОЗАДАЧНАЯ ВЕРСИЯ ФОРТА

Форт наиболее эффективен для задач управления. Современные задачи характеризуются тем, что требуют контролировать несколько одновременно происходящих процессов. Эти процессы могут быть независимыми или взаимодействующими.



Версия FIG-FORTH, а также описанная в [28–30] ориентированы на одну задачу-процесс. Эти версии позволяют, конечно, управлять и многопроцессорными экспериментами, но они не могут быть эффективными, особенно в тех случаях, когда требуется "мгновенная" реакция на события в сочетании с быстрым обменом информацией между процессами. Именно эти обстоятельства и послужили стимулом для создания многозадачной версии Форта (MT) [39].

Разработанная версия рассчитана на четыре независимых задачи-процесса. Каждая задача имеет три буфера (экрана) для работы с дисками и независимые области для словарей, стеков и входных буферов. Базовый словарь общий для всех задач. Распределение ресурса памяти между словарями задач жесткое, определяется при генерации интерпретатора. Размещение буферов в памяти показано на рис. 11. Эта версия разработана для микроЭВМ ДВК. Главным ее ограничением является малая емкость памяти ЭВМ. Однако этот вариант легко развить для ЭВМ с расширенной памятью (более 64 Кбайт).

Главной трудностью при создании интерпретатора было распределение ресурсов внешних устройств между процессорами. Для решения этой задачи в данной реализации применены слова-операторы EXB, FRE, SDS, CDS, SPL и CBL, введены специальные флаги-семафоры, управляющие "захватом" того или иного ресурса.

Переключение процессора с одной задачи на другую выполняется по внутренним часам ЭВМ под управлением программы MAN. Программа просматривает состояние всех задач, выделяет из них те, выполнению которых ничто не препятствует (задача не ожидает какого-либо занятого ресурса внешних устройств). Для этих задач

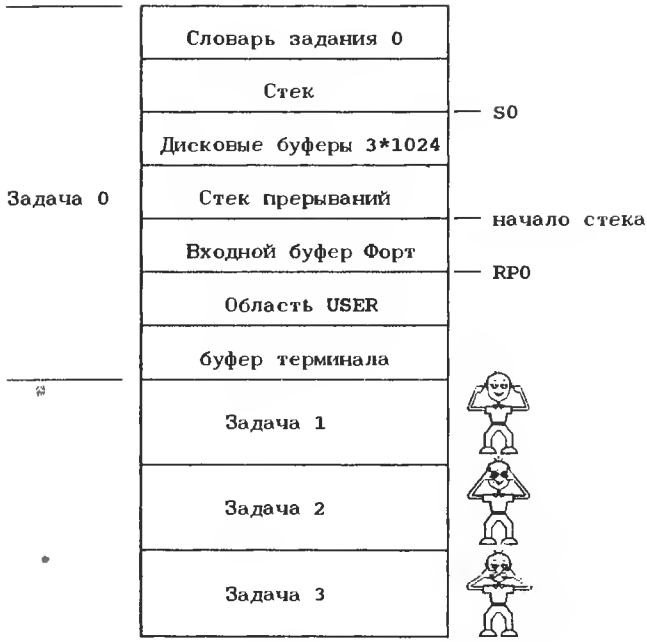


Рис. 11. Структура буферов многозадачной версии Форта

вычисляется произведение PRIO*DELAY, где PRIO – приоритет задачи, а DELAY – число квантов времени пребывания задачи в пассивном состоянии из-за выполнения более приоритетного задания. Задача с наибольшим произведением запускается в работу на время очередного кванта. Значение PRIO устанавливает программист, оно может изменяться в процессе выполнения задачи. Если задание выходит в точку, где требуется печатающее устройство, вывод на дисплей или обмен с диском, а данный ресурс занят, производится перевод задачи в режим ожидания до момента освобождения необходимого ресурса. Более сложна конкуренция при ожидании ввода с терминала. Для системы Форт в пассивном состоянии (QUIT) характерно ожидание ввода с терминала. Таким образом, сразу после загрузки все виртуальные задачи ожидают ввода с клавиатуры. В процессе работы любое задание также может потребовать ввода.

Для организации процесса перераспределения ресурсов каждая из задач имеет в пользовательской области статусную переменную STA. (адрес 0 в пользовательской области USER). Каждый ресурс снабжен флагом-семафором: TERB – ввод с клавиатуры, DSB – вывод на экран, PRIB – вывод на печать, DIB – обмен с диском. Если ресурс занят,

флаг-семафор равен USN+1, где USN – номер задания, им владеющего, в противном случае он равен нулю.

Подпрограмма STD используется в словах-операторах KEY и EXB. Если задание является "хозяином" клавиатуры, то при входе в STD никаких изменений не происходит и система ожидает ввода. Если "хозяином" клавиатуры является другое задание и его STA >0, данное задание входит в цикл ожидания ресурса, в противном случае задача захватывает ресурс сама, установив TERB=USN+1 (USN – номер этого задания). Для управления вводом-выводом используется флаг TERB+1, который устанавливается командой ^X (при этом блокируется любой вывод на экран терминала). Очистка TERB+1 производится сигналом <ВК>, одновременно сбрасываются старший и младший биты переменной STA. Таким образом, вывод, прерванный ^X на время ввода команды с пульта, будет продолжен.

Задание, непосредственно потребовавшее ввода, может его отнять у задачи, пребывающей в состоянии QUIT. Если право на ввод было получено заданием в процессе EXECUTE, оно остается единственным владельцем этого ресурса до тех пор, пока пользователь не нажмет клавишу "Возврат каретки". Все прочие задания, также требующие ввода, будут вынуждены ожидать этого события. Аналогичным приоритетом по вводу обладает задание, активированное командой N TASK (N – номер виртуальной задачи).

Для других ресурсов захват возможен лишь при нулевом значении семафора, т. е. только после их освобождения. Каждое прерывание по часам ЭВМ (50 Гц) приводит к обращению к планировщику MAN и просмотру состояния заданий. Процессорное время предоставляется только задаче, готовой к исполнению. Во время работы MAN внешние прерывания запрещены. Аналогичная ситуация имеет место при обработке прерываний с клавиатуры терминала.

Каждое задание имеет свой входной кольцевой буфер емкостью 134 байт, куда заносятся коды ASCII, поступающие от клавиатуры. Из этого кольцевого буфера они извлекаются оператором KEY. Соответствующего выходного буфера для дисплея или принтера в данной версии не предусмотрено.

Логика переключения семафора исключает ошибочный ввод символов в буфер "чужого" задания. В рассматриваемой версии с целью ускорения работы системы предусмотрено наличие трех экранных буферов для каждого задания (сокращается число чтений-записей с диска и на диск). Одновременно это создает широкие возможности обмена экранами, а также меньшими порциями информации между активными заданиями. При этом, правда, виртуальный файл FORTH.DAT

является общим для всех задач, что практически не налагает серьезных ограничений, хотя и не избавляет программиста от необходимости быть внимательным.

Таким образом, одновременно в оперативной памяти может разместиться 12 экранов (12 Кбайт). Это, конечно, ограничивает предельный объем словарей. При необходимости расширения словарей можно передать эти буферы под словари на уровне генерации системы. Надо только помнить, что в этом случае одно или более заданий будет лишено возможности пользоваться диском.

Еще одной особенностью многозадачного Форта является наличие областей пользовательских переменных. В этих областях хранятся переменные, полностью характеризующие состояние задачи. При прерывании по часам и переключении с одной задачи на другую меняется указатель области пользовательских переменных и сохраняются-восстанавливаются слово состояния JSW и указатель стека возвратов. В данном случае слово состояния JSW используется только для управления вводом-выводом (разряды 6 и 12) аналогично ОС RT-11. Такая схема упрощает программу смены задач и сокращает непроизводительные потери времени на сохранение-восстановление большого числа переменных.

Механизм управления флагами-семафорами для клавиатуры терминала показан на рис. 12. Операторы EXB и FRE устанавливают и сбрасывают управляющие биты переменной состояния STA (область пользовательских переменных). Оператор KEY входит в состав слова

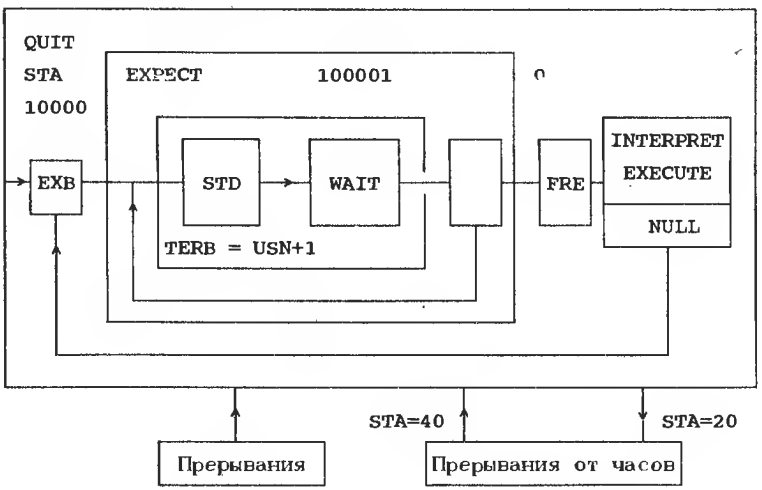


Рис. 12. Управление семафорами и схема обработки прерываний

EXPECT (это справедливо и для FIG-FORTH), но может использоваться и самостоятельно, в последнем случае в конце его исполнения флаг занятости сбрасывается. Но эта процедура внутри EXPECT блокирована, чтобы подавить переключение ввода на другой входной буфер до завершения процедуры QUERY, составной частью которой является EXPECT. Таким образом, вводимая команда будет целиком загружена в нужный буфер.

Область пользовательских и системных переменных. Системные переменные образуют особый класс слов Форт. В FIG-FORTH они размещены в специальной области памяти – области пользовательских переменных USER. Такое решение упрощает создание многозадачной и многопользовательской версий интерпретатора. Часть системных переменных является универсальной для многих версий, другие используются только в конкретных реализациях.

Имена и функции основных системных переменных приведены в табл. 26. Часть из них используется в качестве констант, задающих режим работы интерпретатора: WIDTH фиксирует максимальную длину имен операторов в словаре Форта, WARNING (предупреждение) определяет характер отклика при ошибках и прочих ситуациях, ненормальных с точки зрения интерпретатора; PTN при неравенстве нулю дублирует все выдачи, поступающие на экран, цифропечатающее устройство; FENCE (ограда) служит для защиты базового словаря от стирания с помощью оператора FORGET, указывает адрес, предшествующий первой свободной ячейке в словаре сразу после загрузки системы Форт, т. е. при этом FENCE @ равно HERE-2.

Другие системные переменные задают текущий режим работы и могут изменяться в процессе исполнения программы: STATE задает режим исполнения (=0) или интерпретации (='300); CONTEXT/CURRENT – контекстные переключатели словаря; BASE – основание действующей системы счисления; BLK переключает ввод информации на терминальный буфер (BLK=0) или на экранный (BLK, равный номеру экрана); DP указывает на первую свободную ячейку словаря (DP @= HERE); SCR служит для хранения текущего значения номера экрана (например, при редактировании); IN – внутренний указатель входного буфера (терминального или экранного в зависимости от значения BLK); R# – указатель положения курсора на экране; VOCL – переменная связи контекстных словарей.

Третью группу образуют переменные, которые при работе обычно не меняются и задают взаимоположения отдельных частей системы Форт: S0 и R0 определяют положение начала стека параметров и стека возвратов соответственно; TIB – указатель начала входного буфера;

Таблица 26. Системные переменные Форт

Имя	Функция
S0	Указатель начала стека параметров
R0	Указатель начала стека возврата
TIB	Указатель начала входного буфера
IN (>IN)	Указатель смещения во входном (или экранном) буфере
BLK	BLK=0 — работа системы со входным терминальным буфером, BLK≠0 — работа с блоком номер BLK@
WIDTH	Определяет максимальную длину имени в словаре
DP (H)	Указатель на первую свободную ячейку в словаре. DP@ = HERE
BASE	Основание системы счисления
FENCE	Граница базового словаря Форт. Используется оператором FORGET
CURRENT	Указатель, к какому словарю будет отнесено новое слово
CONTEXT	Указатель, с какого словаря следует начинать просмотр при интерпретации
SCR	Номер редактируемого экрана
R#	Указатель положения курсора на экране при редактировании
STATE	STATE=0 — исполнение, STATE='300 — компиляция

FIRST – адрес первого байта экранных буферов, LIMIT – указатель конца экранных буферов и некоторые другие.

Если область памяти USER имеет достаточные размеры, можно описать свои переменные, имеющие аналогичный статус. Для этого используется оператор USER, который по своим функциям сходен с оператором VARIABLE, но определяемые им переменные лежат не в словаре, а в области пользовательских переменных. Форма описания также несколько иная, например: 66 USER NEW, где NEW – имя новой переменной (описание в словаре); 66 – положение ячейки, где записано ее значение, отсчитанное от начала области пользовательских переменных. В отличие от обычной переменной NEW будет иметь в первой ячейке поля параметров число 66, которое при обращении будет использовано как указатель ячейки области USER, из которой следует взять ее значение. Для пользователя эта переменная ничем не отличается от обычной. При обращении к NEW в стек будет записан адрес значения этой переменной. При извлечении ее значения в стек следует выполнить команду типа NEW @.

Структура контекстных словарей виртуальных задач. Схема связей словарей задач 0,..., 3 представлена на рис.13. Словари TASK-1, TASK-2 и TASK-3 в отличие от FIG-FORTH связаны с PFA+6 (адрес поля параметров + 6) FORTH. В FIG-FORTH PFA+6 TASK-2 должно было бы указывать на PFA+6 TASK-1, а VOCL (переменная из области USER,

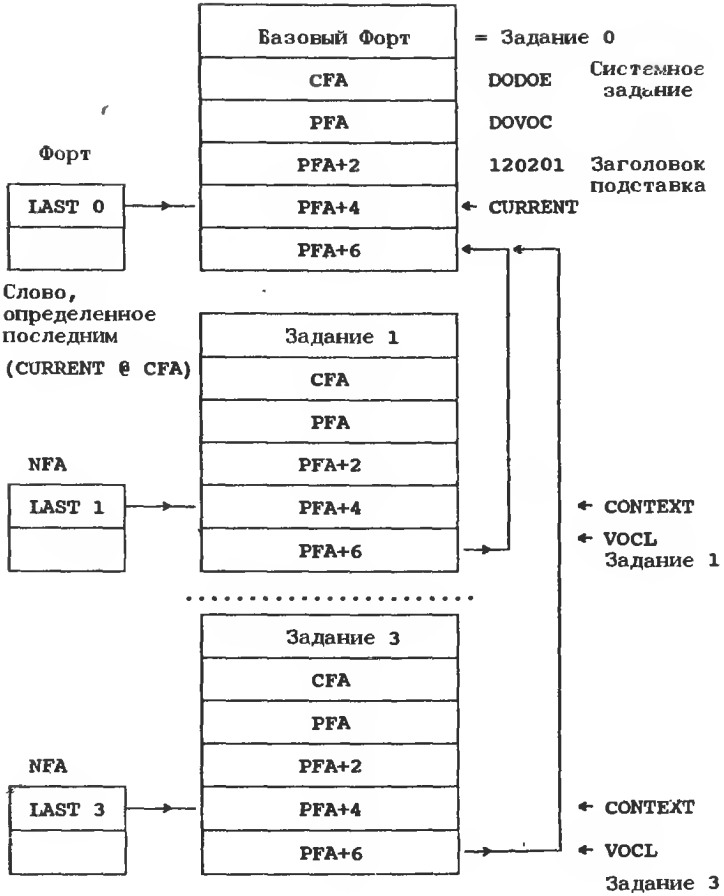


Рис. 13. Схема связей между словарями задач

содержащая адрес связи словарей с разными контекстами) – на PFA+6 TASK-2. Структура словаря обеспечивает свободный обмен информацией между задачами. Для реализации такого обмена общие переменные или массивы должны быть описаны в TASK-0 (Форт) или храниться в одном из экранных буферов. Возможны варианты, где буфер находится в словаре TASK-i, а указатель на него – в TASK-0.

Работа с многозадачным Фортом. Загрузка многозадачного Форта не отличается от обычной. По завершении загрузки появляется сообщение FORTH-MT IS HERE. При этом инициализированы все четыре виртуальные задачи (если сгенерирована именно такая версия). Далее запускается основная задача. Вход в задачу i+1 можно осуществить,

нажав ^X, при этом ЭВМ выдаст Tj>, где j – номер очередной виртуальной задачи. Переадресацию ввода на задачу j можно выполнить также с помощью команды J TASK, где J может принимать значения 0, 1, 2 и 3. Подтверждение выполнения команды аналогично ^X. Теперь вплоть до нажатия клавиши <ВК> все вводимые символы поступают в кольцевой буфер указанной задачи. Принадлежность ввода можно определить, нажав <ВК>. В норме отклик должен быть, как и при ^X или J TASK. Но если задача находится в активном состоянии (например, производится счет), отклика может и не быть. Кроме того, после нажатия клавиши <ВК> за ввод позволено бороться любой задаче. Если нужно ввести текст, содержащий несколько строк, в некоторых случаях это может потребовать манипулирования вводом через ^X или J TASK, что на практике встречается редко. В процессе ввода команды все задачи (кроме J) продолжают работать.

Вывод на экран управляется командами ^X, ^S и ^Q: ^X прерывает выдачу на экран и возобновляет ее по завершении ввода после нажатия <ВК>, ^S прерывает вывод, ^Q возобновляет как и в случае ОС RT-11. Исполнение задачи – хозяина ввода – прерывает команда ^C, переводя ее в режим QUIT (ожидание новых инструкций). Перевод многозадачного Форт в базисное состояние производится командой INI. В этом случае все буферы очищаются, инициализируются стеки и уничтожаются все словари, введенные после стартовой загрузки. Если требуется установить в базисное состояние только одну виртуальную задачу J, следует выдать команду J INIT. Функции INI и INIT те же, что и в случае COLD [17,30].

Многозадачный Форт устроен так, что пользователю достаточно освоить только несколько новых команд, связанных с переключением ресурсов и задач. Поскольку выдача на экран терминала или печать может производиться несколькими задачами попеременно, пользователю рекомендуется начинать каждую выдачу с пропечатки номера задачи. Для этого предусмотрена команда TSK, которая в начале первой строки пишет Tj>, где j – номер задачи, откуда производится выдача. В случае персональной ЭВМ возможно использование разными задачами различных цветов (или фонов).

Для преобразования версии FIG-FORTH [30] в многозадачный вариант Форта пришлось переделать или написать заново 32 слова. Интерпретатор позволяет легко перейти от этой версии к многопользовательской (PolyFORTH [15,16]). Для этого в области USER предусмотрена переменная (CSR), которая хранит адрес терминала пользователя. Интерпретатор для многопользовательской версии, сохранив все возможности многозадачной, несколько упростится, так как из него

будут убраны части, ответственные за обслуживание нескольких задач с одного терминала. Это справедливо, если каждому пользователю разрешен запуск только одной задачи. Для круга проблем, где Форт эффективен, такое ограничение не существенно. В многозадачном Форте защита задания от влияния ошибок в других задачах целиком лежит на программисте (ведь он хозяин всех задач). В многопользовательском варианте это становится более актуальным, желательны аппаратные средства, которые на системном уровне защитят одного пользователя от ошибок другого.

Примеры использования многозадачного Форта. Простейшим примером может быть выдача на печать протяженных массивов. При работе с версией МТ-Форта можно запустить такую печать и, перейдя к другой виртуальной задаче, заняться редактированием любой (в том числе работающей) программы.

В условиях эксперимента одна из задач может вести накопление данных (обработка прерываний, гистограммирование и т. д.), вторая представлять результаты на графическом мониторе, третья записывать данные на магнитный носитель, четвертая обслуживать запросы оператора. Возможно и другое распределение обязанностей.

Достоинствами многозадачной версии являются:

совместное использование четырьмя задачами-процессами базового словаря Форт, что заметно снижает требования к емкости памяти;

простота обмена информацией между задачами (удобнее, чем FOREGROUND-BACKGROUND в ОС RT-11).

Последнее свойство служит причиной главного недостатка – отсутствия защиты от взаимного влияния заданий в случае программных ошибок. Переделка МТ-Форта ЭВМ типа СМ для персональных ЭВМ IBM ВС больших трудностей не составит. Но к описанным недостаткам в этом случае добавится несовершенство базовой системы ввода-вывода BIOS. Дело в том, что версия BIOS этих ЭВМ сильно ориентирована на одну задачу и затрудняет использование времени ожидания при вводе-выводе для работы параллельных заданий. Таким образом, для обеспечения эффективной работы Форта на IBM PC в многозадачном или многопользовательском режиме желательно переделать BIOS ЭВМ.

Важной особенностью описанной версии реализации многозадачного Форта является возможность его работы автономно без операционной системы.

ЗАКЛЮЧЕНИЕ

Если у вас хватило времени и сил просмотреть эту книгу, вы поняли, что не существует задач, которые бы было нельзя решить с помощью Форта. Этот относительно молодой язык успешно соревнуется как со старожилами мира программирования (Фортраном, Паскалем), так и с такими новыми, но мощными конкурентами, как Си. В литературе около 10 лет назад предсказывалась триумфальная победа Паскаля над Фортраном. И хотя Паскаль имеет на своем счету немало успехов, вытеснить Фортран из его традиционной "среды обитания" – научных расчетов – Паскалю не удалось. Этому есть несколько причин. Среди основных – появление новой версии Фортрана, а главное – огромная библиотека прикладных программ, с которыми потребители, естественно, не желают расставаться. Сделать правильный прогноз для Форта не легче. Появление Форт-процессоров, в том числе с сокращенным набором команд, удобство применения в качестве промежуточного языка при написании трансляторов, простота работы с периферийным оборудованием и, конечно, великолепные перспективы в сфере управления в реальном масштабе времени – все это говорит в пользу Форта. Но судьба языков определяется не только их удобством (пример тому – история Алгола), здесь играет роль и активность энтузиастов, и появление доступных пакетов прикладных программ, и поддержка мощными фирмами и многое другое. Одно ясно – Форт имеет право на жизнь, многие идеи, заложенные в нем, несомненно, окажут влияние на пути эволюции языков программирования. Если вы заинтересуетесь Фортом и попытаетесь написать на нем какие-то программы, автор будет считать свою задачу выполненной.

Пожалуй, ни одна система программирования не вызывала у меня чувств, близких к тем, какие испытал главный герой романа Г. Гессе "Игра в бисер" Кнехт – он наслаждался гармонией математических формул. Форт может доставить немало мгновений эстетического наслаждения программистам-профессионалам.

Приложение 1

Экранный редактор EDT

Для дисплея СМ7209, Электроника МЭ15 и их аналогов; ОС RT-11. Экранный редактор для персональной ЭВМ типа IBM PC имеет практически тот же объем, но предоставляет существенно большие возможности.)

```
: SET   0 36 ! ;                                ( установка режима в JSW)
: ES    27 EMIT ;                               ( передача кода ESC на терминал)
: $     VARIABLE ;
VOCABULARY EDITOR IMMEDIATE                     ( описание словаря EDITOR)
: R@    R# @ ;                                  ( запись в стек координаты курсора)
: OF    20480 36 ! ;                            ( запись управляющего кода в JSW)
: G.    0 GL ! ;                                ( обнуление переменной GL)
: CR#   0 R# ! ; ( запись нуля в ячейку-указатель позиции курсора)
: R!    R# ! ;   ( запись кода в ячейку-указатель позиции курсора)
: ED    -DUP IF SC ! CR# SC @ BLOCK DROP [COMPILE] EDITOR
        THEN ;                                  ( чтение экрана в буфер)
EDITOR DEFINITIONS                              ( начало описания процедур EDT)
( описание массивов и переменных)
0 $ B 2 ALLOT       0 $ GL       0 $ SY       1 $ D       0 $ SC
0 $ LBF 64 ALLOT                                        ( буфер строки)
0 $ BF 1K ALLOT                                         ( "CUT"- буфер)
0 $ WBF 64 ALLOT                                        ( буфер слова)
: IL    7 EMIT ;                                        ( звуковой сигнал)
: RC    13 EMIT ;                                       ( возврат каретки)
: C#    R@ 64 MOD ;                             ( вычисление номера колонки)
: +C    R@ + 0 MAX 1023 MIN R! ;    ( изменение содержимого ячейки
                                    R# с учетом ограничения 0 ≤ R# ≤ 1023)
: L#    R@ 64 / ;                               ( вычисление номера строки)
: L*    L# 15 < ;                                       ( последняя строка?)
: G     GL @ ;
: CA    SC @ BLOCK UPDATE R@ + ;    ( вычисление абсолютного адреса
                                    в буфере редактора и установка флага "спасения")
: LA    CA C# - ;           ( вычисление адреса первого байта в строке)
: TL    64 C# - ;           ( вычисление числа байт до конца строки)
: B#    0 BUFFER DUP 1K BLANKS ;        ( резервирование буфера и его
                                        очистка)
: FX    RC 22811 PAD ! PAD 2+ ! PAD 4 TYPE ;    ( фиксация курсора
                                                в заданной точке экрана)
: HT    8224 FX ;           ( установка курсора в начало поля команд)
: AL    1K R@ - CMOVE ;             ( вычисление числа байт от курсора до
                                    конца буфера)
```

```
: TY   L# 2 .R SPACE LA 64 TYPE 92 EMIT ;  ( печать строки с сим-
                                            волом "\" в конце)
: SCL  HT ES 74 EMIT ;                          ( гашение экрана)
: V    64 +C ;                          ( перемещение курсора вниз)
: NX   TL +C CA ;      ( вычисление адреса первого байта следующей
                         строки)
: ^    -64 +C ;                         ( перемещение курсора вверх)
: >>   1 +C ;         ( перемещение курсора на одну позицию вправо)
: <<   -1 +C ;        ( перемещение курсора на одну позицию влево)
: LE   TL 1- ;
: LB   0 C# - +C RC ;          ( возврат в начало следующей строки)
: D@   D @ ;
: BLA  TL BLANKS ;              ( очистка строки справа от курсора)
: D?   D@ 1+ ;            : F@   BF @ ;
: LCL  HT ES 75 EMIT ;                   ( очистка командной строки)
: OL   B# LA OVER AL LB LA BLA L*       ( открывание новой строки)
       IF V CA AL ^ ELSE DROP THEN ;
: EDG  D?                                          ( край строки?)
       IF C# 63 < ELSE C# 0 > THEN ;
: PUT  LBF CA TL CMOVE ;    ( перенос содержимого буфера в строку)
: CB   0 B ! ;
: LI   R@ R@ 0=          ( распечатка текста редактируемого экрана)
       IF SCL CR CR ." SC# " SC ? CR CR
       THEN LB 16 L# - 0
       DO TY V CR LOOP R! ;
: CBF  LBF 64 BLANKS ;                     ( очистка буфера строки)
: GAP  CBF CA LBF TL CMOVE ;  ( засылка части строки в  строчный
                                буфер)
: ^W   ES 61 EMIT CR# LI ;          ( "освежение" текста на экране)
: ON   L# 36 + C# 35 + SWAB + FX ;     ( приведение в соответствие
                                позиции курсора и содержимого
                                ячейки-указателя его позиции)
: BUP  G IF G. CR# ELSE -1 D ! THEN ;   ( изменение флага направ-
                                          ления перемещения)
: LINE G IF G. OL LI ELSE LB D@ 64 * +C THEN ;
: SHW  RC TY ;                          ( "освежение" текста строки)
: .TO  BEGIN CA C@ BL = OVER
         IF 0= THEN EDG *
       WHILE D@ +C
       REPEAT DROP ;
: WST  0 .TO 1 .TO ;                              ( шаг "по словам")
: ..   GAP CA C! LE IF >> PUT THEN ;
: DS   G          ( стирание символа, на который указывает курсор)
       IF G. SY C@ ..                  ( восстановление символа)
       ELSE CA C@ SY C! LE
            IF >> GAP << PUT ELSE BL CA C! THEN
       THEN SHW ;
```

```
: DEL  << DS ;                    ( стирание символа слева от курсора)
: SS   .. SHW ;              ( ввод символа и печать измененной строки)
: DL   G                           ( стирание-восстановление строки)
       IF G. C# 0=
          IF OL THEN PUT SHW
       ELSE GAP C# 0= L* *
          IF  V B# CA OVER AL ^ CA AL  ELSE  CA BLA  THEN
       THEN ;
: DLL  DL LI ;                  ( стирание строки и распечатка текста)
: PAGE G    ( вызов следующего экрана или уход в режим "команда")
       IF G. HT ." COMMAND: " SET QUERY LCL INTERPRET OF
       ELSE FLUSH D@ SC +! ^W
       THEN ;
: QUIT ES 62 EMIT EMPTY-BUFFERS SCL SET ABORT ;  ( уход из редак-
                тора без записи отредактированного текста на диск)
: .T   G                                    ( выделение части текста)
       IF G. CB ELSE R@ B 2+ ! 1 B ! THEN ;
: EXIT FLUSH QUIT ;      ( уход из редактора с записью результатов
                           редактирования на диск)
: GLD  1 GL ! ;
: EL   LE D@ 64 * + +C ;        ( установка курсора в конец строки)
: DW   G                           ( стирание-восстановление слова)
       IF G. GAP WBF 2+ CA                          ( восстановление)
          WBF @ TL MIN DUP +C CMOVE C# 0=
          IF <<
          THEN LE
          IF PUT
          THEN
       ELSE R@ WST R@ D?                                  ( стирание)
          IF SWAP
          THEN OVER R! DUP LE
               IF GAP
               ELSE CBF 1- SWAP
               THEN R! - DUP WBF ! CA WBF 2+ ROT CMOVE PUT
       THEN SHW ;                  ( отображение измененной строки)
: WIPE CR# CA 1K BLANKS ^W ; : HP ;          ( стирание содержимого
                                               экрана)
: ^L   G                                ( перенос и соединение строк)
       IF G. R@ L*
          IF V LB DL R! PUT THEN
       ELSE GAP CA BLA V OL PUT ^
       THEN LI ;
: ++   G                        ( перемещение курсора в конец экрана)
       IF G. 1023 R!
       ELSE 1 D !
       THEN ;
: CUT  B# >R G          ( вырезание и вставление фрагментов текста)
       IF G. R@ R@ GAP NX R AL R! BF 2+ CA F@ CMOVE F@
          +C PUT R> NX AL R!
       ELSE B @
            IF GAP CA L*
               IF NX R AL THEN B 2+ @ R! CA - DUP 0>
               IF BF ! CA BF 2+ F@ CMOVE PUT R@ R> NX AL R!
               ELSE R> 2DROP IL
```

```
            THEN
          ELSE LEV IL
          THEN
       THEN ON LI CB ;

: SEE .G                          ( поиск последовательности символов)
       IF G. HT ." MODEL: " SET QUERY 1 WORD OF LCL
       THEN R@ R@ D?
       IF 1K SWAP - THEN 0
       DO D@ +C CA HERE COUNT OVER + SWAP
             DO DUP C@ I C@ -
                IF DROP 0 LEAVE THEN 1+
             LOOP 1-
                IF DROP R@ LEAVE THEN
       LOOP R! ;

( список управляющих символов)

3099 $ L1 6015 , 8 , 16961 , 17220 , 20561 , 16210 ,

28272 , 29042 , 29556 , 30070 , 30584 , '46571 ,

: TT   0              ( декодировка и выполнение команд редактора)
       DO 2DUP I + C@ =
         IF 2DROP LEAVE I 12 - THEN
       LOOP DUP 0<
       IF 12 + 2* + @ EXECUTE
       ELSE DROP DUP 31 >
            IF SS ELSE DROP IL THEN
       THEN ;

( команды-списки редактирующих операторов)

: T3   LINE .T EL WST ++ CUT DS BUP SEE PAGE DW IL ;

: .P   ' T3 KEY L1 14 + 12 TT ;

: T2   ^ V << >> HP GLD DLL .P ;

: P.   ' T2 KEY L1 6 + 8 TT ;

: T1   P. ^L DEL ^W DEL ;

FORTH  DEFINITIONS          ( словарь FORTH снова стал контекстным)

: EDT  ED EDITOR LI ON ES 61 EMIT OF CB    ( вызов редактора EDT)
       BEGIN ' T1 KEY L1 5 TT ON 0 UNTIL ;

: COPY SWAP BLOCK CFA ! UPDATE FLUSH ;     ( копирование экранов в
                                    пределах одного файла: N1 N2 COPY)

: GDE  ." SCR=" PREV @ @ . R@ 64 / ." LINE=" . ;    ( поиск ошибок
                                                      при загрузке)
```

Приложение 2

Библиотека для работы с числами с плавающей точкой

```
( Формат чисел соответствует требованиям СМ ЭВМ )

: ODER 16 0 DO DUP 0< IF 32 I - CSP ! LEAVE THEN 2*
            LOOP DROP ;                          ( преднормализация)

: E+ CSP +! ;

: 2@ DUP 2+ @ SWAP @ ;          ( извлечение числа двойной длины по
                                  адресу, лежащему в стеке)
```

```
: 2! DUP >R ! R> 2+ ! ;        ( аналог "!" только для чисел двойной
                                 длины)

: NORM >R BEGIN    ,                          ( оператор нормализации)
          ,DUP '76000 AND
          WHILE 2 DIV I
             IF 1 E+ THEN
          REPEAT
          BEGIN DUP 512 AND 0=
          WHILE 2 MUL I
             IF -1 E+ THEN
          REPEAT LEV ;

: FL   ( преобразование чисел из входного формата [после NUMBER ]
       в формат с плавающей точкой)
    0 CSP ! DUP 32768 AND R# ! DPL @ DUP -1 =
       IF DROP S->D 0 THEN MINUS $EX +! R# @
       IF DABS THEN DUP
       IF DUP ODER
       ELSE OVER -DUP
            IF ODER -16 E+ THEN
       THEN CSP @
       IF 0 NORM $EX @ 3 * E+ BASE @ 10 =
          IF $EX @ -DUP
             IF ABS 0
                DO $EX @ 0<
                   IF 8 MUL 10 DIV
                   ELSE 10 MUL 8 DIV
                   THEN 1 NORM
                LOOP
             THEN
          THEN 511 AND 4 DIV R# @ OR CSP @ 127 AND CSP @ 0>
             IF 128 OR THEN 128 * OR
       THEN ;

: NORA BEGIN DUP '77600 AND
       WHILE 10 DIV 1 E+
       REPEAT
       BEGIN DUP '3600 AND 0=
       WHILE 10 MUL -1 E+
       REPEAT 10 DIV 1 E+ ;

: 2**  -DUP                          ( функция 2^N, обращение: N 2**)
       IF 1- -DUP
          IF 2 SWAP 0
             DO 2*
             LOOP
          ELSE 2                  ( если N=1)
          THEN
       ELSE 1                     ( если N=0)
       THEN ;

(изменение знака числа с плавающей точкой)
CODE FCHS '100000 # R0 MOV, R0 S () XOR, NEXT, C;

: FTY    DUP 0<                    ( печать числа с плавающей точкой)
         IF "-" 32767 AND
         ELSE SPACE
         THEN DUP 128 / 128 - >R 127 AND 128 OR 2 DIV
         I I 3 / DUP CSP ! DUP R# ! /MOD 2** DPL
         ! ABS 2** R> 0<
```

```
        IF DIV ELSE MUL THEN R# @ ABS -DUP
        IF 0
           DO R# @ 0<
             IF 10 MUL DPL @ DIV
             ELSE DPL @ MUL 10 DIV
             THEN NORA
           LOOP
        ELSE NORA
        THEN ." ." 7 0
        DO 10 MUL DUP 128 / 48 + EMIT 127 AND
        LOOP 2DROP ." E" CSP @ . ;

: FC    FL <BUILDS , , DOES> 2@ ;  ( описание оператора "константа
                                     с плавающей точкой")

1E0  FC 1E                          ( единица с плавающей точкой)

2E0 FC 2E                                ( 2 с плавающей точкой)

: F0=   OR 0= ;                                    ( аналог 0=)

: F?    2@ FTY ;          ( аналог "?" для чисел с плавающей точкой)

CODE    F+ S FADD, NEXT, C;          ( аналоги + - * / ABS для чисел
                                     с плавающей точкой)

CODE    F- S FSUB, NEXT, C;

CODE    F* S FMUL, NEXT, C;

CODE    F/ S FDIV, NEXT, C;

CODE    FABS '100000 # S () BIC, NEXT, C;

CODE    F0> 2 S I) CLR, S )+ TST, GT IF, S () INC, THEN,
        NEXT, C;                ( аналог 0>)

CODE    F0< 2 S I) CLR, S )+ TST, LT IF, S () INC, THEN,
        NEXT, C;                ( аналог 0<)

: F+!   >R I 2@ F+ R> 2! ;

: 1/X   2DUP OR          ( функция 1/X для чисел с плавающей точкой)
          IF 1E 2SWAP F/
          ELSE 26 ERROR              ( сообщение об ошибке при X=0)
          THEN ;

: F** DUP R# ! ABS -DUP          ( функция  X^N, обращение X N F**)
      IF 1- -DUP
         IF >R 2DUP R> 0
            DO 2OVER F*                          ( N-1 умножение)
            LOOP 2SWAP 2DROP
         THEN
      ELSE 2DROP 1E              ( если N = 0)
      THEN R# @ 0<
      IF 1/X                     ( если N < 0)
      THEN ;

: INT   DUP 16384 AND     ( выделение целой части из числа с плава-
                            ющей точкой)
      IF >R I I '37600 AND 128 / 16 MIN 2** >R 128
         OR '377 AND 256 DIV R> MUL SWAP DROP R> 0<
         IF MINUS THEN
      ELSE 2DROP 0
      THEN ;

: 4DUP 2DUP 2DUP ;
```

```
: 4DR  2DROP 2DROP ;

: FV   FL VARIABLE , ;          ( описание оператора "переменная с
                                  плавающей точкой")

: F-!  >R FCHS R> F+! ;

15E0 FC 15F                          3.1415927 FC PI ( число π)

: F>   F- F0> ;       : F<   F- F0< ;       : EX0  0 $EX ! ;

2.71828183 FC EE     0 0 FV XX    3E0 FC 3E      -1E0 FC -1E

CODE 2>R    S )+ RP -) MOV, S )+ RP -) MOV, NEXT, C;

CODE 2R>    RP )+ S -) MOV, RP )+ S -) MOV, NEXT, C;

CODE 2R     RP () S -) MOV, 2 RP I) S -) MOV, NEXT, C;

: SQRT FABS  2DUP 2/ '17600 AND '40000 OR 5 0 DO 2OVER 2OVER
       F/ F+ 2E F/ LOOP 2SWAP 2DROP ;    ( извлечение квадратного
                                  корня из числа с плавающей точкой)

: LN   EX0 -1 DPL ! DUP 128 / 127 AND FL LN2 F*
       2SWAP 127 AND 16384 OR 1E 2OVER F- 2SWAP
       1E F+ F/ 0 0 8 1
       DO 2OVER I 2* 1- >R R F** R> FL F/ F+
       LOOP 2SWAP 2DROP 2E F* F- ;

: EXP  EE 2OVER INT >R R F** 2SWAP EX0 R>
       -1 DPL ! FL F- 1E XX 2! 1E 8 1
       DO 2OVER F* I FL F/ 2DUP XX F+!
       LOOP 4DR XX 2@ F* ;

: SIN  EX0 -1 DPL ! 4DUP F* 2SWAP 2DUP 8 1
       DO XX 2! 2OVER F* I 2* DUP 1+ * FL F/
       32768 XOR 2DUP XX 2@ F+
       LOOP 2>R 4DR 2R> ;
```

Приложение 3

Программа "Плакатная печать"

(Текст вводится обычным образом, а распечатывается буквами высотой в 7 строчек)

```
: $ VARIABLE ; OCTAL                    ( таблица моделей символов)
( А) 3 $ PAT 5 , 11 , 17 , 21 , 21 , 21 ,
( Б) 37 , 20 , 20 , 36 , 21 , 21 , 36 ,
( В) 36 , 21 , 21 , 36 , 21 , 21 , 36 ,
( Г) 37 , 20 , 20 , 20 , 20 , 20 , 20 ,
( Д) 17 , 11 , 11 , 11 , 11 , 37 , 21 ,
( Е) 37 , 20 , 20 , 34 , 20 , 20 , 37 ,
( Ж) 25 , 25 , 16 , 4 , 16 , 25 , 25 ,
( З) 16 , 21 , 1 , 6 , 1 , 21 , 16 ,
( И) 21 , 23 , 23 , 25 , 31 , 31 , 21 ,
( Й) 25 , 25 , 21 , 23 , 25 , 31 , 21 ,
( К) 21 , 21 , 22 , 34 , 22 , 21 , 21 ,
( Л) 17 , 5 , 5 , 5 , 5 , 5 , 31 ,
( М) 21 , 33 , 33 , 25 , 25 , 21 , 21 ,
```

```
( Н) 21 , 21 , 21 , 37 , 21 , 21 , 21 ,
( О) 16 , 21 , 21 , 21 , 21 , 21 , 16 ,
( П) 37 , 21 , 21 , 21 , 21 , 21 , 21 ,
( Р) 36 , 21 , 21 , 36 , 20 , 20 , 20 ,
( С) 37 , 20 , 20 , 20 , 20 , 20 , 37 ,
( Т) 37 , 4 , 4 , 4 , 4 , 4 , 4 ,
( У) 21 , 21 , 21 , 37 , 1 , 1 , 37 ,
( Ф) 16 , 25 , 25 , 25 , 16 , 4 , 4 ,
( Х) 21 , 21 , 12 , 4 , 12 , 21 , 21 ,
( Ц) 22 , 22 , 22 , 22 , 22 , 37 , 1 ,
( Ч) 21 , 21 , 21 , 17 , 1 , 1 , 1 ,
( Ш) 25 , 25 , 25 , 25 , 25 , 25 , 37 , 40000 44 !
( Щ) 25 , 25 , 25 , 25 , 25 , 37 , 1 ,
( Ы) 21 , 21 , 21 , 35 , 25 , 25 , 35 ,
( Ь) 20 , 20 , 20 , 36 , 21 , 21 , 36 ,
( Ъ) 30 , 10 , 10 , 36 , 11 , 11 , 36 ,
( Э) 16 , 21 , 1 , 7 , 1 , 21 , 16 ,
( Ю) 27 , 25 , 25 , 35 , 25 , 25 , 27 ,
( Я) 17 , 21 , 21 , 17 , 3 , 11 , 21 ,
( 1) 4 , 4 , 4 , 4 , 4 , 4 , 4 ,
( 2) 16 , 21 , 1 , 2 , 4 , 10 , 37 ,
( 3) 16 , 21 , 1 , 6 , 1 , 21 , 16 ,
( 4) 21 , 21 , 21 , 17 , 1 , 1 , 1 ,
( 5) 37 , 20 , 20 , 36 , 1 , 21 , 36 ,
( 6) 16 , 21 , 20 , 36 , 21 , 21 , 36 ,
( 7) 37 , 1 , 1 , 2 , 4 , 10 , 20 ,
( 8) 16 , 21 , 21 , 16 , 21 , 21 , 16 ,
( 9) 16 , 21 , 21 , 17 , 1 , 21 , 16 ,
( 0) 16 , 21 , 21 , 21 , 21 , 21 , 16 ,
( .) 0 , 0 , 0 , 0 , 0 , 14 , 14 ,
( ,) 0 , 0 , 0 , 0 , 14 , 14 , 10 ,
( :) 14 , 14 , 0 , 0 , 0 , 14 , 14 ,
( ;) 14 , 14 , 0 , 0 , 14 , 14 , 10 ,
( !) 4 , 4 , 4 , 4 , 4 , 0 , 4 ,
( ?) 16 , 21 , 2 , 4 , 4 , 0 , 4 ,

DECIMAL

: STRING                              ( описание структур типа "строка")
        <BUILDS HERE BL WORD C@ 1+ ALLOT ALIGN
        DOES> COUNT ;

( Описание строки с именем ALF. При обращении к ALF в стек запи-
  сывается адрес начала строки и число элементов в ней)

STRING ALF АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЫЬЪЭЮЯ1234567890.,:;!?

0 $ BB 980 ALLOT              ( описание буфера, возможен вариант:
                  0 $ BB 0 BUFFER BB !, экономящий место в памяти)

: CBB BB 980 BLANKS ;                          ( очистка буфера BB)

0 $ BU 40 ALLOT                ( буфер      ввода исходных данных)

80 $ N#                0 $ LL

0 $ 00 0 , 0 , 0 , 0 , 0 , 0 ,                    ( модель пробела)

'40000 '44 !

: TT      BB 7 0       ( распечатка матрицы, записанной в буфер BB)
          DO CR ( 14 SPACES) DUP N# @ TYPE 140 +
          LOOP 0 LL ! ;

: WHO   ALF 0                    ( распознавание введенного символа)
          DO DUP I + C@ 3 PICK =
             IF 2DROP I 1 LEAVE          ( если символ узнан, в стек
                                         записывается его номер и 4)
```

```
            ELSE I 1+ I' =
                IF 2DROP 0    ( если символ не узнан)    THEN
            THEN
        LOOP ;

: TY        ( ввод нужного текста, подготовка матрицы для печати)
        BEGIN CBB ." > " BU 40 EXPECT     ( ввод исходной строки)
              BU 20 0
              DO DUP I + C@ DUP
                IF WHO        ( распознавание введенных символов)
                    IF 14 * PAT +
                    ELSE 00          ( ввод пробела, если символ не
                                       узнан)
                    THEN 7 0
                    DO DUP @ BB I 140 * + LL @ 7
                      * + R# ! 5 0        ( заполнение буфера BB)
                      DO DUP 16 AND
                         IF '173 R# @ C!     ( запись байта в BB)
                         THEN 1 R# +! ( обновление указателя) 2*
                      LOOP DROP 2+
                    LOOP 1 LL +! DROP
                ELSE LEAVE
                THEN
              LOOP TT          ( распечатка строки прописных букв)
              2DROP KEY DUP '145 = SWAP '105 = OR
                                          ( нажата клавиша <E>?)
        UNTIL ;

: PRINT   PRIN TY TTY ;          ( печать текста плакатным шрифтом)
```

Приложение 4

Программа тестирования блока памяти

```
( Блок выполнен в стандарте КАМАК. Положение модуля фиксируется в
  процессе диалога.)
: == CONSTANT ;                          : -- VARIABLE ;

0 == MEM                                 '164000 == CSR

0 -- RES

: RD CSR ! @ ;                    ( чтение данных из модуля КАМАК)
: WT CSR ! ! ;                   ( запись информации в модуль КАМАК)
: MAW MEM 17 WT ;                      ( запись в адресный регистр)
: MW MEM 16 WT ;                     ( запись кода в модуль памяти)
: MR MEM 0 RD ;                     ( чтение кода из модуля памяти)
: START ." MEMO BIN->" QUERY INTERPRET 32 * CSR + ' MEM ! ;
: ERR ." ERR ADR=" . ." CODE=" U. ;          ( сообщение об ошибке)
: RED ." READ=" U. CR ;
: MCLEAR 1024 0 DO I MAW 0 MW 0 MEM 2+ ! LOOP ;
: MTEST 0 MAW 1 15 0                          ( тест модуля памяти)
        DO 1024 0
```

```
        DO I MAW DUP 2DUP DUP MW MR DUP RES ! - 0=
           IF MEM 2+ 16 WT MEM 2+ 0 RD DUP RES ! -
              IF DUP I ." B " ERR RES @ RED
              THEN
           ELSE DUP I ." A " ERR RES @ RED
           THEN
        LOOP 2 *
     LOOP ;
```

Программа тестирования процессора с микропрограммным управлением

(В процессе тестирования с диска или вручную загружается микропрограмма, вводятся исходные коды в память данных, запускается микропрограмма и отображается результат. Предполагается, что аппаратура выполнена в стандарте КАМАК.)

```
'164000 == CSR   512 -- REND    0 -- PT   0 == MM   0 == PRO
     0 -- BLO     0 -- STADR   0 == DM   0 -- STAD
: NU QUERY INTERPRET ;
: MMR MM 0 RD ;                          ( операторы чтения-записи)
: MW DM 16 WT ;                           ( запись в память данных)
: MR DM 0 RD ;                            ( чтение из памяти данных)
: MAW DM 17 WT ;                        ( запись адреса в память данных)
: MWW DM 6 + 17 WT ;
: MWR DM 4 + 17 WT ;
: ERM ." ILL ADR" U. ABORT ;                    ( сообщение об ошибке)
: FADR ." FIRST ADR->"                         ( ввод начального адреса)
      NU DUP 0< OVER 1023 > OR
      IF ERM THEN ;
: BIN 32 * CSR + ;                    ( вычисление КАМАК-адреса модуля)
: SHOW DUP C@ . 1+ ;
: LEV TIO R> DROP ;
: DEL 1024 PT @ - ;
: GET QUERY N# -DUP
     IF 69 =
        IF LEV
        ELSE INTERPRET
        THEN
     THEN ;
: TAKE MWR ." OLD.LS=" MR 5 U.R ." MS=" M2R 5 U.R I' MWW
     ." NEW.LS=" GET N#
     IF MW
     THEN ." MS=" GET N#
     IF M2W
     THEN ;
: FILI BEGIN FADR 32767 SWAP ." <E>
           IF END/JMP" CR
```

```
                DO ." ADR=" I 5 U.R I TAKE
                LOOP ." NEW ADR? <'N'/CR>" KEY 78 =
        UNTIL KEY ;

: DELT R# @ DUP DEL >
        IF DROP DEL
        THEN DEL MINUS REND +! DUP MINUS R# +! ;

( Загрузка исходных данных)

: DLOD ." D-MEM BIN->" NU BIN ' DM ! FILI ;

: RECD DUP @ ." IC#=" . 2+ DUP @ DUP STAD ! ." START PC="
       DUP 1- STADR ! . 4 + DUP @ ." BYTE/IC=" DUP . ;

: PLUG 0 DO 2+ DUP SHOW SHOW SHOW C@ . CR DUP @
          1 STADR +! STADR @ MMAW MMW 2+ DUP @ MM 2+ ! 4
        +LOOP DROP ;

( Загрузка микропрограммы)

: MLOD ." M-MEM BIN-> " NU BIN ' MM ! ." BY HAND? <Y/RET> "
       KEY '131 =
       IF KEY MM ' DM ! FILI
       ELSE FILCH              ( загрузка микропрограммы из файла)
            1 BLO ! 1 BLOCK DUP DUP ." MAXR=" @ . 2+
            DUP ." COL=" @ . 2+ DUP ." ROW=" @ . 3 +
            ." IC DEPTH=" C@ . 520 + 520 PT ! CR RECD
            R# ! CR ." IC TEXT" CR OCTAL
            BEGIN DELT PLUG R# @

                  IF 0 0 PT ! 1 BLO +! REND @
                     IF BLO @ BLOCK
                     ELSE 512 REND ! 8 PT ! RECD
                     THEN
                  ELSE 1
                  THEN
            UNTIL DEFOL DECIMAL 512 REND !
       THEN ;
: GO STAD @ PRO ! ;
: RESULT ." RESULT MEMADR-> " NU DUP MAW 1+ MR ." RESLT="
        U. DM 2+ @ U. MAW ." [ADR+1]=" MR U. DM 2+ @ U. ;

: LODR ." PROCESSOR BIN-> " NU BIN ' PRO ! MLOD DLOD
       GO RESULT ;  ( пуск процессора и распечатка результата )
```

Приложение 5

Программа HELP

```
( Дисплей Электроника ИЭ15 или СМ7209; ОС RT-11 )
: $ VARIABLE ;              ( переопределение с целью экономии места)
0 $ CU              ( переменная, характеризующая положение курсора)
: FX 13 EMIT 22811 PAD ! PAD 2+ ! PAD 4 TYPE ; ( фиксация курсора
                                                 в нужной точке экрана)
: RU 14 EMIT ;                    ( переход к русскому алфавиту)
: LA 15 EMIT ;                   ( переход к латинскому алфавиту)
: IL 7 EMIT ;                                    ( звуковой сигнал)

( список экранов, где хранится справочная информация)
0 $ TS 61 , 28 , 50 , 23 , 51 , 24 , 46 , 48 , 27 , 26 , 25 ,
```

```
: NX CU @ + DUP 24 =                          ( позиционирование курсора)
     IF DROP 0
     THEN 0 MAX DUP CU ! 2/ 2 /MOD 8226 + SWAP 28 * 256 * +  FX ;

: EX FX 0 CU ! 27 EMIT 74 EMIT ;       ( гашение экрана и установка
                                        курсора в исходное состояние)

: HT 8233 EX RU ;

: .OU -DUP 0=                                 ( уход из программы HELP)
      IF DROP LEV 8224 EX
      THEN ;

: GO 8226 FX              ( прием, дешифровка и исполнение команд)
     BEGIN 0 KEY 27 =     ( начало анализа ESC-последовательности)
           IF KEY DUP 67 =
              IF DROP 2 NX                                ( если <→>)
              ELSE DUP 68 =
                   IF DROP -2 NX                          ( если <←>)
                   ELSE DUP 65 =
                        IF DROP -4 NX                     ( если <I>)
                        ELSE DUP 66 =
                             IF DROP 4 NX                 ( если <↓>)
                             ELSE 81 =                 ( если <HELP>)
                                  IF CU @ TS + @ .OU LOAD
                                     8226 FX ( возвращение курсо-

                                                ра в исходное положение)
                                  ELSE IL
                                  THEN
                             THEN
                        THEN
                   THEN
              THEN
           ELSE IL        ( звуковой сигнал при ошибочной команде)
           THEN
     UNTIL ;

: HELP 8224 EX RU ." ВЫБЕРИ и НАЖМИ" LA ." <HELP>:" CR
       CR RU ." УХОД                          РЕДАКТОР"
       CR    ." ОПЕРАТОРЫ УСЛОВИЯ             ЦИКЛА"
       CR    ." ОПЕРАЦИИ С ПАМЯТЬЮ            АРИФМЕТИКА"
       CR    ." СИНТАКСИС ФОРТ                УТИЛИТЫ"
       CR    ." ОПЕРАТОРЫ ФОРТ                РАБОТА СО СТЕКОМ"
       CR    ." ВОЗВРАТНЫЙ СТЕК               ВВОД/ВЫВОД"

       4096 36 !                                    ( установка JSW)
       GO LA 16384 36 !                  ( восстановление JSW) ;
```

Пример текстов справочных экранов

НТ ." Вход в РЕДАКТОР осуществляется путем ввода с пульта номера экрана." CR ." и имени редактора. Например:" LA ." 20 EDT" RU CR ." Перемещение курсора по экрану выполняется с помощью клавиш <ВК>" CR ." стрелок, направленных вверх, вниз, вправо, влево, а также клавиш <0> и <2>" CR ." <1> - служит для перемещения курсора на одно слово" CR ." <4> - задает направление перемещения курсора вперед, а <5> - назад " CR ." <6> - забой символа, на который указывает курсор" CR." <7> - переход к редактированию следу

ющего экрана" CR ." ДЧСВ <7> переход в режим выполнения команд" CR LA ." <EXIT>" RU ." = выход из редактора" CR LA ." <WIPE>" RU ." = стиранию экрана" CR LA ." <N EDT>" RU ." переход к редактированию экрана " LA ." <N>" RU CR ." <4> и <5> влияют на работу <0>, <1>, <2>, <7>, <8>, <9>" CR ." стрелка вверх двойная черта (СВДЧ) стирает строку начиная с позиции," CR ." указанной курсором."

KEY DROP HT ." <8> - обеспечивает вход в режим поиска нужной комбинации символов." CR ." но перед <8> следует нажать <двойная черта стрелка вниз> (ДЧСВ)," CR ." Нажатие клавиши <9> - " ." стирает слово. Для восстановления стертого слова нужно" CR ." нажать клавиши <ДЧСВ> и <9>" CR ." Комбинация <ДЧСВ> и <СВДЧ> восстанавливает стертую строку." CR LA ." ^W" RU ." служит для обновления текста экрана при сбоях" CR ." Курсор при этом возвращается в исходное положение." CR ." Вернуть курсор в исходное положение можно нажатием клавиш <ДЧСВ> и <5>." CR ." <,> и <3> служат для вырезания и вставки кусков текста." CR ." Начало вырезаемого текста отмечается курсором и нажатием клавиши <,>;" CR ." конец - курсором и <3>, последняя операция удаляет отмеченный кусок из текста." CR ." ввод вырезанного текста в место, указанное курсором, производится нажатием <ДЧСВ> и <3>" CR LA." <^L>" RU ." переносит текст правее курсора, на следующую строку." CR ." <ДЧСВ> и " LA ." <^L>" RU ." осуществляет обратную операцию."

Приложение 6

Программа подготовки автономной версии Форта

```
( Для гибкого 5"-диска типа "6022" или его аналога;  ОС RT-11)

: SIZETST 1024 9 * 256 - HERE U<        ( контроль объема словаря)
        IF ." TOO BIG" QUIT THEN ;
  SIZETST FORGET SIZETST
  LATEST 4 +ORIGIN ! HERE 20 +ORIGIN ! HERE 18 +ORIGIN !

: WRITE.INT 62 52 DO I 52 - 1024 * 512 + ( откуда)
            I BLOCK ( куда)
            1024 CMOVE UPDATE LOOP FLUSH ;
WRITE.INT 6 LOAD                          ( загрузка Форт-ассемблера)
ASSEMBLER DEFINITIONS OCTAL  ( начало описания операторов словаря
                             ASSEMBLER)
0 VARIABLE TI$
: *TIM 100 # R0 MOV
       *BEGIN 200 # R4 () BIS
```

```
            *BEGIN R4 () TSTB MI
            *UNTIL 200 # R4 () BIC R0 DEC LE
      *UNTIL PC RTS ;

: *INI 50000 # SP     MOV
     177130 # R4     MOV
       1000 # R5     MOV
         10 # R1     MOV
         24 # SP -) MOV
         SP () R3    MOV
      10040 # R4 () BIC ;

: *STEP   20 # R4 () BIS  PC TI$ @ JSR ;
: *T0 *BEGIN *STEP 4000 # R4 () BIT NE *UNTIL ;
: *WT *BEGIN R4 () TST MI *UNTIL ;
: *TRK 40 # R4 () BIS *BEGIN *STEP R3 DEC LE *UNTIL ;
: *TRB 40000 # R4 () BIC
       40000 # R4 () BIS *WT 177132 @# TST
       *WT 177132 @# R2 MOV ;
: *ER R1 DEC LE *IF HALT *THEN ;
: *READ *BEGIN *TRB R2 SP () CMP NE
                *IF *ER
                *ELSE 13 # R2 MOV SP -) CLR
                      *BEGIN 200 # R3 MOV   R0 CLR
                             *BEGIN  *WT 177132 @# R5 () MOV
                                     R5 )+ R0 ADD, R3 DEC, LE
                             *UNTIL *WT 177132 @# R0 CMP NE
                                     *IF *ER R2 CLR SP () INC
                                     *THEN R2 DEC LE
                      *UNTIL SP )+ TST EQ
                             *IF SP () INC *STEP *THEN
                *THEN  27 # SP () CMP LE
        *UNTIL 1000 @# JMP ;
137 45000 ! 45004 DP ! HERE TI$ ! *TIM HERE 45000 - 45002 !
*INI *T0 *TRK *READ 45000 0 0 WTS 45400 0 1 WTS    ( запись про-
                                                  граммы первичной загрузки)

FORGET WRITE.INT DECIMAL
```

Приложение 7

Программа чтения оглавления диска (для RT-11)

```
( Для дисковода PLX-45, ОС RT-11)
: ASC          ( преобразование одиночного кода RAD50 в ASCII-код)
      DUP
      IF DUP 27 =
         IF DROP 36
         ELSE DUP 27 <
            IF 64 +
            ELSE DUP 40 <
              IF 18 +
              ELSE DROP BL                    ( если код не узнан)
              THEN
            THEN
         THEN
      ELSE DROP 32                          ( если RAD50=0, пробел)
      THEN ;
```

```
: RASC
      0 DO
      DUP I 3 * + >R OVER I' 2* + @ 0 40 U/ SWAP ASC R
      2+ C! 40 /MOD SWAP ASC R 1+ C! ASC R> C!
        LOOP 2DROP ;

0 VARIABLE NM 4 ALLOT

: $R   NM 3 RASC NM 3 TYPE ;

: #L   DUP 8 + @ DUP 5 .R
: TAB  11 SPACES ;
: DAT  -DUP                          ( декодировка даты создания файла)
      IF 32 /MOD 32 /MOD SWAP 4 .R 1- 10
      45 EMIT * 7 - ' JAN + 3 TYPE 45 EMIT 72 + 2 .R
      ELSE TAB THEN ;
: $NM 2+ DUP $R 2+ DUP $R 46 EMIT 2+ $R ;
: DRL 0 >R 0 R# ! 0 CSP !
      BEGIN DUP @ DUP 2048 -
      WHILE R> 1+ >R DUP 1K <
        IF ." " < EMPTY > " DROP #L DROP TAB
        ELSE 1 R# +! SWAP DUP $NM #L CSP +! SWAP '102000 =
             IF 80 EMIT THEN DUP 12 + @ DAT
        THEN R 2 MOD
             IF 3 SPACES ELSE CR THEN 14 +
      REPEAT CR R# ? ." FILES " 212 CSP @ DUP .
      ." BLOCKS" CR - . ." FREE BLOCKS" LEV ;

: DIR FLUSH 11 ' SKIP >R R ! 1 BLOCK 10 + DRL 15 R> ! SP!
     'EMPTY-BUFFERS ;                              ( FIG-FORTH)
```

Приложение 8

Модель оператора **NUMBER**

```
( Преобразование счетной строки символов в число одинарной или
  двойной длины со знаком. Выходной формат приспособлен для
  последующего представления числа с плавающей точкой.)

: CROL DPL ! (NUMBER) DUP C@ BL - ;

: EXP DPL @ >R 0 0 ROT DUP C@ '105 - 0
     ?ERROR 1+ DUP C@ '55 =
     IF (NUMBER) C@ BL - 0 ?ERROR DROP MINUS
( (NUMBER) вспомогательный оператор со схемой изменения состоя-
  ния стека: 0 0 HERE -> N1 N2 HERE+K . Используется для преобра-
  зования символьной строки с адресом HERE в число без знака)
     ELSE DUP C@ '53 -
          IF 1- THEN (NUMBER) C@ BL - 0 ?ERROR DROP
     THEN EXPO ! 0 R> DPL ! ;

: NUMA 0 EXPO ! BASE @ >R 0 0 ROT 1+ DUP C@ DUP '53 =
      IF 10 BASE ! 0 >R DROP
      ELSE DUP '55 =
           IF 10 BASE ! 1 >R DROP
           ELSE 0 >R '47 =
                IF 8 BASE !
                ELSE 1-
                THEN
           THEN
      THEN CROL
```

```
      IF DUP C@ '56 =
         IF 0 CROL DPL @ ." DPL=" U.
            IF EXP THEN
         ELSE EXP
         THEN
      THEN DROP R>
      IF DMINUS THEN R> BASE ! ;
0 VARIABLE AA 20 ALLOT
: TT AA 1+ 16 EXPECT AA NUMBER ." RESL=" U. U. ." DPL=" DPL @
 . ." EXP=" EXPO @ . ;
```

Приложение 9
Форт-интерпретатор для персональной ЭВМ

Приведемный текст интерпретатора соответствует стандарту FIG-FORTH и ориентирован на персональную ЭВМ типа IBM PC (т.е. близок к Форт-79). Его можно адаптировать и к стандарту Форт-83. При его написании ставилась задача сохранения возможности использования библиотек и прикладных программ, реализованных для модифицированной версии FIG-FORTH в ИТЭФ. Интерпретатор имеет некоторые слова для работы с периферийным оборудованием (PORT, READ), а также операторы, предназначенные для представления информации на экране (SCL, PIX, LCL и т.д.). Данную версию можно рассматривать в качестве примера реализации интерпретатора на простом процессоре. Здесь сознательно не использованы многие команды процессора 80286 и некоторые возможности BIOS версии EGA AT для того, чтобы сделать эту программу приемлемой на ЭВМ более простой конфигурации. Интерпретатор работает как с монохромными (MG-150, HERCULES), так и с цветными (CGA, EGA) графическими адаптерами. В последнем случае экранный редактор и некоторые другие библиотеки предоставляют пользователю большие возможности. Воспользовавшись данным текстом, вы сможете модифицировать эту версию Форт в соответствии с вашими вкусами и привычками. Некоторые программы, приведенные в данной книге (например, программа, исполняющая мелодию "Подмосковные вечера"), написаны для данной версии интерпретатора. Программы составлена на ассемблере ЭВМ IBM PC и рассчитана на работу с виртуальным файлом FORTH.DAT на гибком диске "A". Если вас не устраивает, можно снять это ограничение, отредактировав FCB в конце текста или воспользовавшись другой более совершенной версией интерпретатора.

```
;                    FORTH ИТЭФ 1988 Семенов Ю.А.

 LINK=0
```

```
.LALL
;                              Макроопределения
; Описание примитивов и слов высокого уровня
HEAD  MACRO length,name,lchar,labl,code
LINK$=$
      DB  length
      IFNB <NAME>
      DB  NAME
      ENDIF
      DB  lchar
      DW  LINK
LINK=LINK$
labl     LABEL   FAR
      IFNB <code>
      DW  code
      ELSE
      DW  $+2
      ENDIF
      ENDM

NEXT MACRO    ; переход к исполнению следующего слова
      LODSW
      MOV  BX, AX
      JMP  [BX]
      ENDM

      TITLE   FORTH Interpreter

ARRAY    SEGMENT
$TIME    DW  0,0                        ; TIMER COUNTER
$DATE    DW  0                          ; DATE
; Сегмент стека и словарь пользователя размещен в конце текста
; интерпретатора. Для версии, записываемой в ПЗУ, их нужно раз-
; местить здесь.

DSKBUF   DW  3 DUP(0,512 DUP(0),0)   ; Экранные буферы
ENDBUF   DW  0

XTIB     DW  92 DUP(0)                  ; Входной буфер
XR0      DW  0,0                        ; Стек возвратов
XUP      DW  102 DUP(0)                 ; USER-область

$STI     DW  TASK-7Q ;                  ; Стартовая таблица
$US      DW  XUP      ;+2
$STK     DW  XS0      ;+4
$RS      DW  XR0      ;+6
         DW  XTIB     ;+8
         DW  37Q      ;+10
         DW  0        ;+12
         DW  XDP      ;+14
         DW  XDP      ;+16
         DW  XVOC     ;+18
$BUF     DW  DSKBUF   ;+20
         DW  ENDBUF   ;+22

    ASSUME  CS:ARRAY, DS:ARRAY, ES:ARRAY, SS:STACK

$INI       PROC    FAR
           JMP  ENT
```

```
; ** PRIMITIVES **

          HEAD    203Q,'IN',311Q,INIT                    ;INI
ENT:      MOV  CX, ARRAY
          MOV  DS, CX          ; Установка DX
          MOV  ES, CX
          MOV  BX, $US
          MOV  AX, $STI        ; Восстановление словаря
          LEA  SI, FORTH+6
          MOV  [SI], AX
          MOV  SI, $BUF
          MOV  CX, 1730        ; Установка счетчика
XXX:      MOV  WORD PTR [SI],0 ; Обнуление массивов
          ADD  SI, 2
          LOOP XXX

; INIT 'OFFSET, USE, PREV'
          MOV  CX, $BUF        ; TO 'USE'
          MOV  [BX]+72Q,CX
          MOV  [BX]+74Q,CX     ; ?
          MOV  CX, 14Q         ; Установка счетчика USER
          MOV  DI, $US         ; Запись адреса области USER
          ADD  DI, 6
          LEA  SI, $STK        ; Запись начального адреса
REP       MOVS WORD PTR ES:[DI],WORD PTR DS:[SI]
          MOV  BP, $RS         ; Установка начального значения
          MOV  DI, $US         ; указателя стека возвратов
          MOV  WORD PTR [DI+32Q],7 ; Установка цвета
          MOV  WORD PTR [DI+42Q],0 ; Сброс флага печати
          LEA  SI, GO$
          NEXT
GO$:      DW SPSTO,DEC,FORTH,DEFIN,ONE,LOAD
$INI      ENDP

; BX - WP                      BP - указатель стека возвратов
; SI - IP-регистр              SP - указатель стека параметров
; DI - указатель области USER

          HEAD 207Q,'EXECUT',305Q,EXEC                   ; EXECUTE
          POP  BX
          JMP  [BX]

          HEAD    203Q,'LI',324Q,LIT                     ; LIT
          LODSW
          PUSH AX
          NEXT

          HEAD    206Q,'TERMO',316Q,TERMON               ; TERMON
          POP  AX
          PUSH ES
          MOV  CX, 0
          MOV  ES, CX
          MOV  ES:417H,AX
          POP  ES
          NEXT

          HEAD    207Q,'?BRANC',310Q,ZBRAN               ; ?BRANCH
; Ветвление при нуле в стеке (FALSE)
          POP  AX
          CMP  AX, 0
          JE   CNT
```

```
        ADD  SI, 2
        NEXT

        HEAD    206Q,'BRANC',310Q,BRAN                  ; BRANCH
CNT:    ADD  SI, [SI]
        NEXT

        HEAD    204Q,'(DO',251Q,XDO                     ; (DO)
        POP  AX
        SUB  BP, 2
        POP  [BP]
        SUB  BP, 2
        MOV  WORD PTR [BP],AX
        NEXT

        HEAD    206Q,'(LOOP',251Q,XLOOP                 ; (LOOP)
; Приращение индекса цикла LOOP и может быть ветвление
        INC  WORD PTR [BP]
LOP:    MOV  AX, [BP]
        CMP  AX, 2[BP]
        JL   CNT
LV:     ADD  BP, 4
        ADD  SI, 2
        NEXT

        HEAD    207Q,'(+LOOP',251Q,XPLOO                ; (+LOOP)
        POP  AX
        ADD  [BP],   AX
        CMP  AX, 0
        JL   $LESS
        JMP  LOP
$LESS:  MOV  CX, [BP]  ; Работа с отрицательными приращениями
        CMP  [BP]+2, CX
        JLE  LV
        JMP  CNT

        HEAD    206Q,'(FIND',251Q,PFIND                 ; PFIND
; Адрес строки NFA => PFA длина TRUE/FALSE
        POP  AX
        POP  CX
        PUSH BP         ; сохранение содержимого регистров
        PUSH SI
        PUSH DI
        MOV  SI, CX
        SUB  BP, BP
        MOV  DI, AX
        MOV  DX,WORD PTR [SI]
        AND  DX, 77577Q
        CLD                     ; DF=0 (вперед)
FAST:   MOV  CX,WORD PTR [DI]
        AND  CX, 77477Q
        CMP  DX, CX
        JE   SLOW
MATCH:  CMP  WORD PTR [DI], 0
        JS   $SIG               ; BPL
        INC  DI
        JMP  MATCH
$SIG:   ADD  DI, 2
        CMP  WORD PTR [DI],0
        JE   FAIL
        MOV  DI,WORD PTR [DI]
        JMP  FAST
```

```
SLOW:   MOV  BP,WORD PTR [DI]
        MOV  BX, SI
        JMP  SLOW1
$LOOP:  INC  BX
        MOV  AX,WORD PTR [BX]
        MOV  CX,WORD PTR [DI]
        AND  CX, 77777Q
        CMP  AX, CX
        JNE  MATCH
SLOW1:  INC  DI
        TEST WORD PTR -1[DI],100000Q
        JE   $LOOP
        MOV  DX, BP
        ADD  DI, 5
        MOV  AX, DI
        POP  DI          ; восстановление содержимого регистров
        POP  SI
        POP  BP
        SUB  AX, 2
        PUSH AX
        AND  DX, 377Q          ; байт длины
        PUSH DX                ; в стек
        JMP  TRUE              ; Установка флага "Найдено"
FAIL:   POP  DI
        POP  SI
        POP  BP
        JMP  FALSE

        HEAD    205Q,'DIGI',324Q,DIGIT                  ; DIGIT
; ASCII-DIGIT BASE=>DIGIT-VALUE TRUE (FALSE)
        POP  AX       ; AX=BASE
        POP  CX
        SUB  CX, 60Q ; VALID DIGIT = ASCII-60
        JL   FALSE
        CMP  CX, 9    ; Если >9
        JLE  M09
        SUB  CX, 7
        CMP  CX, 10
        JL   FALSE
M09:    CMP  CX, AX   ; Если не меньше BASE, то ошибка
        JGE  FALSE
        PUSH CX       ; Запись цифры в стек
        JMP  TRUE     ; "Успешный" выход

;       ** Стандартные слова **

;       ** Условные операторы **

        HEAD    202Q,'0',275Q,ZEQU                      ; 0=
        POP  AX
        CMP  AX, 0
        JE   TRUE
FALSE:  SUB  AX, AX
PUT:    PUSH AX
        NEXT
TRUE:   MOV  AX, 1
        JMP  PUT

        HEAD    202Q,'0',276Q,ZGRET                     ; 0>
        POP  AX
        CMP  AX, 0
        JG   TRUE
```

```
        JMP   FALSE

        HEAD    202Q,'0',274Q,ZLESS                    ; 0<
        POP   AX
        CMP   AX, 0
        JS    TRUE          ; Если минус
        JMP   FALSE

        HEAD    201Q,,275Q,EQUAL                       ; =
        POP   AX
        POP   CX
        CMP   CX, AX
        JE    TRUE
        JMP   FALSE

        HEAD    202Q,'U',274Q,ULESS                    ; U<
        POP   AX
        POP   CX
        CMP   CX, AX
        JB    TRUE              ; для чисел без знака
        JMP   FALSE

        HEAD    201Q,,274Q,LESS                        ; <
        POP   AX
        POP   CX
        CMP   AX, CX
        JG    TRUE
        JMP   FALSE

        HEAD    201Q,,276Q,GREAT                       ; >
        POP   AX
        POP   CX
        CMP   AX, CX
        JL    TRUE
        JMP   FALSE

;       ******************

        HEAD    204Q,'ENC',314Q,ENCL                   ; ENCLOSE
        POP   AX          ; разделитель

        POP   CX          ; начальный адрес
        MOV   BX, CX
A:      CMP   BYTE PTR [BX], AL ;обход разделителей в начале
        JNE   NOTEQ
AAA:    INC   BX
        JMP   A
NOTEQ:  CMP   BYTE PTR [BX], 15Q
        JE    AAA
        CMP   BYTE PTR [BX], 12Q
        JE    AAA
        MOV   DX, BX              ; начало лексемы
        PUSH  DX
AA:     CMP   BYTE PTR [BX],  0
        JE    ZZZ                 ; если нуль
        CMP   BYTE PTR [BX], AL  ;не нуль, ищем конец лексемы
        JE    QW
        INC   BX
        JMP   AA
EQW:    MOV   AX, BX
        SUB   BX, DX
        PUSH  BX
```

```
        SUB  AX, CX
        INC  AX
        PUSH AX
        NEXT
ZZZ:    CMP  BX, DX
        JNE  EQW
        INC  BX
        JMP  EQW

;       ** Дисплей **

        HEAD    204Q,'PAG',305Q,$PAGE                   ; PAGE
; Установка активной страницы ( PAGE --)
        POP  AX
        MOV  AH, 5
        INT  16
        NEXT

        HEAD    203Q,'PI',330Q,PIX                      ; PIX
; COLCOD ROW COLUMN --> -     запись графической точки
        POP  CX                 ; колонка
        POP  BX                 ; строка
        POP  AX                 ; код цветности
        PUSH DX                 ; сохранение DX
        MOV  AH, 12
        MOV  DX, BX
        SUB  DH, DH
        INT  16                 ; запись графической точки
        POP  DX                 ; восстановление DX
        NEXT

        HEAD    204Q,'MOD',301Q,MODA                    ; MODA
; Изменение режима. ( M -> - )
        POP  AX
        SUB  AH, AH
        IN    16
        NEXT

        HEAD    203Q,'EM',311Q,EMI$                     ; EMI
        POP  CX             ; Число символов
        POP  AX             ; Символ
        PUSH CX             ; Сохранение содержимого CX
        PUSH AX             ; Сохранение содержимого AX
        MOV  BX, [DI+32Q]   ; Установка атрибута
        MOV  AH, 15         ; Чтение текущей страницы
        INT  16             ; в BH
        POP  AX             ; Восстановление содержимого AX
        MOV  AH, 9          ; Запись строки символов
        INT  16
        MOV  AH, 3          ; Чтение положения курсора
        INT  16
        POP  CX
        ADD  DL, CL
        MOV  AH, 2          ; Установка положения курсора
$EM:    INT  16
        CMP  WORD PTR [DI+42Q],0
        JNE  PRINT
OK:     NEXT

        HEAD    204Q,'EMI',324Q,EMIT,$EMIT              ; EMIT
$EMIT   LABEL   FAR
        POP  AX
```

```
ENT$:     PUSH AX
          MOV  AH, 15
          INT  16
          POP  AX
          MOV  AH, 14
          JMP  $EM
PRINT:    MOV  DX, 0  ; Установка номера принтера
          SUB  AH, AH
          INT  23
          TES  AH, 51Q
          JE   OK
ERR4:     MOV  AL, AH
          ADD  AL, 60Q
          MOV  AH, 14
          INT  16
          MOV  DX, OFFSET ERMES4
          MOV  AH, 9H
          INT  21H
          JMP  TYPE$
ERMES4:   DB   ' PRINTER ERROR $'

          HEAD 202Q,'R',303Q,RC                          ; RC
          MOV  AX, 13
          JMP  ENT$

          HEAD 202Q,'I',314Q,IL                          ; BELL
          MOV  AX, 7
          JMP  ENT$

          HEAD    204Q,'PRI',316Q,PRIN                ; PRINT-FLAG
          INC  WORD PTR [DI+42Q]
          MOV  DX, 0        ; Установка номера принтера
          MOV  AH, 1        ; Инициализация принтера
          INT  17H
          TEST AH, 51Q
          JNE  ERR4
          NEXT

          HEAD    203Q,'TT',331Q,TTY  ; TERMI-FLAG
TYPE$:    MOV  WORD PTR [DI+42Q],0
          NEXT

          HEAD    203Q,'SC',314Q,SCL                     ; SCL
; SCREEN CLEAR
          MOV  CX, 2048     ; Загрузка счетчика
          MOV  AH, 15
          INT  16           ; Установка текущей страницы
          SUB  DX, DX
          MOV  AH, 2        ; Курсор в исходное положение
          INT  16
          MOV  BL, 7
CLEAR:    MOV  AX, 0920H    ; Очистка экрана
          INT  16
          NEXT

          HEAD    203Q,'FI',330Q,FIX                     ; FIX
; Позиционирование курсора: COL ROW FIX
          MOV  AH, 15
          INT  10H   ; Запись текущего номера страницы в BX
          POP  DX
          MOV  DH, DL  ; строка
```

```
        POP  AX
        MOV  DL, AL  ; столбец
        MOV  AH, 2
        INT  10H     ; Фиксация положения курсора
        NEXT

        HEAD    204Q,'DSP',314Q,DSPL                    ; DSPL
; LOAD GRAFBUFFER (b adr DSPL)
        POP  BX  ; адрес
        POP  AX  ; байт
        PUSH ES  ; сохранение регистра сегмента
        MOV  CX, 0B000H ; GRAFBUF
        MOV  ES, CX
        MOV  BYTE PTR ES:[BX], AL
        POP  ES
        NEXT

;       ** Терминал **

        HEAD    203Q,'KE',331Q,KEY                      ; KEY
        SUB  AH, AH
        INT  22
        SUB  AH, AH
        PUSH AX
        NEXT

        HEAD    206Q,'EXPEC',324Q,EXPE                  ; EXPECT
        MOV  AH, 0AH
        POP  CX                     ; Число символов
        POP  BX                     ; Адрес буфера SS
        MOV  BYTE PTR [BX],  CL     ; Засылка ожидаемого числа
        MOV  DX, BX
        INT  21H

        MOV  AL, BYTE PTR [BX+1]
        SUB  AH, AH
        ADD  BX, AX
        MOV  WORD PTR [BX+2],0
        NEXT

        HEAD    205Q,'?TER',244Q,?TER$                  ; ?TER$
        PUSH ES
        SUB  CX, CX
        MOV  ES, CX
        OR   BYTE PTR ES:[417H], 40Q
TER:    MOV  AH, 1
        INT  22
        JE   TER
        SUB  AH, AH
        INT  22
        AND  BYTE PTR ES:[417H],337Q ; Сброс "NUM LOCK"
        POP  ES              ; Восстановление ES
        PUSH AX              ; Запись символа
        NEXT

; ** Ввод/Вывод **

        HEAD    204Q,'POR',324Q,PORT                    ; PORT
        POP  DX
        POP  AX
        OUT  DX, AL
        NEXT
```

```
HEAD    204Q,'REA',304Q,READ                    ; READ
POP  DX
IN   L, DX
SUB  AH, AH
PUSH AX
NEXT

** Арифметика **

HEAD    202Q,'1',253Q,ONEP                      ; 1+
POP  AX
INC  AX
PUSH AX
NEXT


HEAD    202Q,'2',253Q,TWOP                      ; 2+
POP  AX
ADD  AX, 2
PUSH AX
NEXT

HEAD    202Q,'1',255Q,ONEM                      ; 1-
POP  AX
DEC  AX
PUSH AX
NEXT

HEAD    202Q,'2',252Q,MUL2                      ; 2*
POP  AX
SAL  AX, 1
PUSH AX
NEXT

HEAD    202Q,'2',257Q,DIV2                      ; 2/
POP  AX
SAR  AX, 1
PUSH AX
NEXT

HEAD    202Q,'U',252Q,USTAR                     ; U*
POP  AX
POP  CX
MUL  CX
PUSH AX
PUSH DX
NEXT

HEAD    202Q,'U',257Q,USLAS                     ; U/
POP  CX          ; Делитель
POP  DX
POP  AX
DIV  CX
PUSH DX          ; Остаток
PUSH AX          ; Засылка результата
NEXT

HEAD    202Q,'M',252Q,MSTAR                     ; M*
POP  AX
POP  CX
IMUL CX
```

```
        PUSH AX
        PUSH DX
        NEXT

        HEAD    206Q,'DMINU',323Q,DMINU                 ; DMINUS
; Изменение знака числа двойной длины
        POP  AX
DMIN:   NEG  AX
        POP  CX
        NEG  CX
        SBB  AX, 0        ; sub with borrow
        PUSH CX
        PUSH AX
G$:     NEXT

        HEAD    204Q,'DAB',323Q,DABS                    ; DABS
        POP  AX
        CMP  AX, 0
        JS   DMIN
        PUSH AX
        NEXT

        HEAD    202Q,'M',257Q,MSLAS                     ; M/
        POP  CX           ; Делимое
        POP  DX
        POP  AX
        IDIV CX
        PUSH DX
        PUSH AX

        NEXT

        HEAD    201Q,,253Q,PLUS                         ; +
        POP  AX
        POP  CX
        ADD  AX, CX
        PUSH AX
        NEXT

        HEAD    201Q,,255Q,SUB                          ; -
        POP  CX
        POP  AX
        SUB  AX, CX
        PUSH AX
        NEXT

        HEAD    202Q,'D',253Q,DPLUS                     ; D+
        POP  AX
        POP  CX
        MOV  BX, SP
        ADD  SS:[BX+2],  CX
        ADC  SS:[BX],    AX
        NEXT

        HEAD    205Q,'MINU',323Q,MINUS                  ; MINUS
        POP  AX
        NEG  AX
        PUSH AX
        NEXT

        HEAD    204Q,'SWA',302Q,SWAB                    ; SWAB
        POP  AX
```

```
        XCHG AL, AH
        PUSH AX
        NEXT

        HEAD    203Q,'AB',323Q,ABS                      ; ABS
        POP  AX
        CMP  AX, 0
        JNS  AB
        NEG  AX
AB:     PUSH AX
        NEXT

        HEAD    203Q,'AN',304Q,$AND                     ; AND
        POP  AX
        POP  CX
        AND  AX, CX
ENDA:   PUSH AX
        NEXT

        HEAD    202Q,'O',322Q,$OR                       ; OR
        POP  AX
        POP  CX
        OR   AX, CX
        JMP  ENDA

        HEAD    203Q,'XO',322Q,$XOR                     ; XOR
        POP  AX

      POP  CX
      XOR  AX, CX
      JMP  ENDA

      HEAD    204Q,'S->',304Q,STOD                      ; S->D
      POP  AX
      PUSH AX
      MOV  CX, DX           ; Сохранение содержимого DX
      CWD
      PUS  DX
      MOV  DX, CX           ; Восстановление DX
      NEXT

      HEAD    203Q,'MI',316Q,MIN                        ; MIN
      POP  AX
      POP  CX
      CMP  CX, AX
      JL   DEEP
TOP:  PUSH AX
      NEXT
DEEP: PUSH CX
      NEXT

      HEAD    203Q,'MA',330Q,MAX                        ; MAX
      POP  AX
      POP  CX
      CMP  CX, AX
      JGE  DEEP
      JMP  TOP

      HEAD    203Q,'SP',300Q,SPAT                       ; SP@
      MOV  AX, SP
      JMP  ENDA
```

```
            HEAD    203Q,'SP',241Q,SPSTO                ; SP!
            MOV  SP, [DI]+6
            NEXT

            HEAD    203Q,'RP',241Q,RPSTO                ; RP!
            MOV  BP, $RS
            NEXT

            HEAD    202Q,';',323Q,SEMI                  ; ;S
            MOV  SI, [BP]
            ADD  BP, 2
            NEXT

;           ** Стек возвратов **

            HEAD    205Q,'LEAV',305Q,LEAVE              ; LEAVE
            MOV  AX, [BP]
            MOV  [BP]+2, AX
            NEXT

            HEAD    202Q,'>',322Q,TOR                   ; >R
            SUB  BP, 2
            POP  [BP]
            NEXT


            HEAD    202Q,'R',276Q,FROMR                 ; R>
            PUSH [BP]
            ADD  BP, 2
            NEXT

            HEAD    201Q,,311Q,I                        ; I
RR$:        PUSH [BP]
Z$:         NEXT

            HEAD    201Q,,322Q,R                        ; R
            JMP  RR$

            HEAD    202Q,'I',247Q,SRP                   ; I'
            PUSH 2[BP]
            JMP  Z$

            HEAD    203Q,'LE',326Q,LEV                  ; LEV
            ADD  BP, 2
            NEXT

;           ** Операции со стеком параметров **

            HEAD    204Q,'PIC',313Q,PICK                ; PICK
            POP  BX
            CMP  BX, 0
            JLE  $GO
            DEC  BX
            SAL  BX, 1
            ADD  BX, SP
            PUSH SS:[BX]
$GO:        NEXT

            HEAD    204Q,'OVE',322Q,OVER                ; OVER
            MOV  BX, SP
            PUSH SS:[BX]+2
            NEXT
```

```
        HEAD    204Q,'SWA',320Q,SWAP                    ; SWAP
        POP  CX
        POP  AX
        PUSH CX
        PUSH AX
        NEXT

        HEAD    205Q,'2SWA',320Q,DSWAP                  ; 2SWAP
        POP  AX
        POP  CX
        MOV  BX, SP
        MOV  DX,WORD PTR SS:[BX+2]
        PUSH DX
        MOV  DX,WORD PTR SS:[BX]
        PUSH DX
        MOV  WORD PTR SS:[BX],AX
        MOV  WORD PTR SS:[BX+2],CX
        NEXT

        HEAD    203Q,'DU',320Q,DUBL                     ; DUP
        POP  AX
DU:     PUSH AX
C$:     PUSH AX
        NEXT

        HEAD    204Q,'-DU',320Q,DDUP                    ; -DUP
        POP  AX
        CMP  AX, 0
        JNE  DU
        JMP  C$

        HEAD    204Q,'2DU',320Q,DUP2                    ; 2DUP
        POP  AX
        POP  CX
        PUSH CX
        PUSH AX
        PUSH CX
        PUSH AX
        NEXT

        HEAD    203Q,'RO',324Q,ROT                      ; ROT
        POP  AX
        POP  CX
        POP  BX
        PUSH CX
        PUSH AX
        PUSH BX
        NEXT

        HEAD    205Q,'2DRO',320Q,DDROP                  ; 2DROP
DRO:    ADD  SP, 4
        NEXT

        HEAD    204Q,'DRO',320Q,DROP                    ; DROP
DRP:    ADD  SP, 2
        NEXT

;       ** Работа с памятью **

        HEAD    205Q,'CMOV',305Q,CMOVE                  ; CMOVE
        POP  CX         ; Счетчик
```

```
        CMP  CX, 0
        JLE  DRO
        MOV  AX, DI      ; Сохранение содержимого DI,SI
        MOV  BX, SI
        POP  DI          ; Куда
        POP  SI          ; Откуда
REP     MOV  ES: BYTE PTR [DI], DS:[SI]
        MOV  DI, AX      ; Восстановление DI,SI
        MOV  SI, BX
        NEXT

        HEAD    204Q,'FIL',314Q,FILL                    ; FILL
        POP  AX          ; Символ
FLL:    POP  CX          ; Счетчик символов
        CMP  CX, 0
        JLE  DRP         ;
        POP  BX
$REPE:  MOV  BYTE PTR [BX],  AL
        INC  BX

        LOOP $REPE
        NEXT

        HEAD    205Q,'ERAS',305Q,ERASE                  ; ERASE
        SUB  AX, AX
        JMP  FLL

        HEAD    206Q,'BLANK',323Q,BLANK                 ; BLANKS
        MOV  AX, 32
        JMP  FLL

        HEAD    204Q,'HOL',304Q,HOLD                    ; HOLD
        DEC  WORD PTR [DI]+70Q
        POP  AX
        MOV  BX, [DI+70Q]
        MOV  BYTE PTR [BX],  AL
        NEXT

        HEAD    202Q,'+',241Q,PSTOR                     ; +!
        POP  BX
        POP  CX
        ADD  [BX],   CX
        NEXT

        HEAD    204Q,'TOG',314Q,TOGL                    ; TOGGLE
; адр-байта маска=>XOR в байт памяти
        POP  CX          ; Маска
        POP  BX
        XOR  BYTE PTR [BX],CL
        NEXT

        HEAD    201Q,,300Q,AT                           ; @
        POP  BX
        PUSH [BX]
        NEXT

        HEAD    202Q,'C',300Q,CAT                       ; C@
        POP  BX
        MOV  AL, BYTE PTR [BX]
        SUB  AH, AH
        PUSH AX
        NEXT
```

```
        HEAD    201Q,,241Q,STORE                        ; !
        POP  BX
        POP  AX
        MOV  [BX],   AX
        NEXT

        HEAD    202Q,'C',241Q,CSTOR                     ; C!
        POP  BX
        POP  AX
        MOV  BYTE PTR [BX],  AL
        NEXT

        HEAD    203Q,'NF',301Q,NFA                      ; NFA
        POP  BX
        SUB  BX, 5
        MOV  CX, -1

        JMP  TRV

        HEAD    203Q,'LF',301Q,LFA                      ; LFA
        POP  AX
        SUB  AX, 4
        PUSH AX
        NEXT

        HEAD    203Q,'CF',301Q,CFA                      ; CFA
        POP  AX
        SUB  AX, 2
        PUSH AX
        NEXT

        HEAD    204Q,'!CS',320Q,SCSP                    ; !CSP
        MOV  WORD PTR [DI]+64Q,SP
        NEXT

        HEAD    204Q,'HER',305Q,HERE                    ; HERE
        PUSH [DI]+22Q
        NEXT

        HEAD    205Q,'ALLO',324Q,ALLOT                  ; ALLOT
        POP  BX
        ADD  [DI]+22Q,BX
        NEXT

        HEAD    204Q,'TRA',326Q,TRAV                    ; TRAVERSE
; Движение вперед/назад вдоль имени переменной длины
        POP  CX       ; DELTA
        POP  BX
TRV:    ADD  BX, CX
        CMP  BYTE PTR [BX],  0
        JNS  TRV      ; JUMP IF POSITIVE
        PUSH BX
        NEXT

        HEAD    301Q,,333Q,LBRAC                        ; [
; Прекращение компил., начало исполн. (обнуление STATE)
        MOV  WORD PTR [DI]+54Q,0
        NEXT

        HEAD    301Q,,335Q,RBRAC                        ; ]
        MOV  WORD PTR [DI]+54Q,300Q  ;Начало компиляции
        NEXT
```

```
        HEAD    203Q,'HE',330Q,$HEX                     ; HEX
        MOV  WORD PTR [DI+56Q],16
        NEXT

        HEAD    207Q,'DECIMA',314Q,DEC                  ; DECIMAL
        MOV  WORD PTR [DI]+56Q,10
        NEXT

        HEAD    205Q,'OCTA',314Q,OCTAL                  ; OCTAL
        MOV  WORD PTR [DI]+56Q,  8
Y$:     NEXT

        HEAD    206Q,'LATES',324Q,LATES                 ; LATEST
        MOV  BX, [DI+52Q]
        PUSH [BX]
        NEXT

        HEAD    206Q,'-TRAI',314Q,DTRAI                 ; -TRAIL
        POP  CX
        POP  BX
        PUSH BX
        ADD  BX, CX
COMPA:  DEC  BX
        CMP  BYTE PTR [BX],  32  ; BLANK ?
        JNE  NO
        LOOP COMPA
NO:     PUSH CX
        NEXT

        HEAD    205Q,'UPPE',322Q,UPPER                  ; UPPER
        POP  CX
        POP  BX
$COMP:  CMP  BYTE PTR [BX],  141Q
        JL   OFLIM
        CMP  BYTE PTR [BX],  172Q
        JG   OFLIM
        AND  [BX],   177737Q
OFLIM:  INC  BX
        LOOP $COMP
        NEXT

        HEAD    213Q,'DEFINITION',323Q,DEFIN        ; DEFINITIONS
        MOV  AX, 50Q[DI]
        MOV  52Q[DI],AX
        NEXT

        HEAD    204Q,'+BU',306Q,PBUF                    ; +BUF
        MOV  BX, SP
        ADD  SS:[BX],1028
        MOV  AX, SS:[BX]
        CMP  30Q[DI],AX       ; LIMIT
        JNE  PB
        MOV  CX, 26Q[DI]      ; FIRST
        MOV  SS:[BX],CX
 PB:    PUSH SS:[BX]
        MOV  CX, 74Q[DI]
        SUB  SS:[BX-2],CX
        NEXT

        HEAD    206Q,'UPDAT',305Q,UPDAT                 ; UPDATE
        MOV  BX, 74Q[DI]
```

```
        OR   [BX],   100000Q
        NEXT

        HEAD    201Q,,330Q,X                        ; X
        POP  BX
        AND ,[BX],   77777Q
        PUSH [BX]
        NEXT

;       ** Управляющие слова **

        HEAD    301Q,,272Q,COLON,$COL               ; :
        DW QEXEC,SCSP,CURR,AT,CONT,STORE,CREAT
        DW RBRAC,PSCOD
$COL    LABEL   FAR
        ADD  BP, -2
        ADD  BX, 2
        MOV  [BP],   SI
        MOV  SI, BX
        NEXT

        HEAD    301Q,,273Q,SMI,$COL                 ; ;
        DW QCSP,COMP,SEMI,SMUG,LBRAC,SEMI

        HEAD    210Q,'CONSTAN',324Q,CON,$COL        ; CONSTANT
        DW CREAT,SMUG,COMMA,PSCOD
$CON    LABEL   FAR
        ADD  BX, 2
        PUSH [BX]
        NEXT

        HEAD    210Q,'VARIABL',305Q,VAR,$COL        ; VARIABLE
        DW CON,PSCOD
$VAR    LABEL   FAR
        ADD  BX, 2
        PUSH BX
        NEXT

        HEAD    204Q,'USE',322Q,USER,$COL           ; USER
        DW CON,PSCOD
$USE    LABEL   FAR
        ADD  BX, 2
        MOV  AX, [BX]
        ADD  AX, DI
        PUSH AX
        NEXT

        HEAD    205Q,'DOES',276Q,DOES,$COL          ; DOES>
        DW  FROMR,LATES,PFA,STORE,PSCOD
$DOE    LABEL   FAR
        SUB  BP,  2
        ADD  BX, 2
        MOV  [BP],   SI
        MOV  SI, [BX]
        ADD  BX, 2
        PUSH BX
        NEXT

;       ** Константы **

        HEAD    201Q,,260Q,ZERO,$CON                ; 0
        DW 0
```

```
HEAD    201Q,,261Q,ONE,$CON                     ; 1
DW 1

HEAD    201Q,,262Q,TWO,$CON                     ; 2
DW 2

HEAD    201Q,,263Q,THREE,$CON                   ; 3
DW 3

HEAD    201Q,,270Q,EIGHT,$CON                   ; 8
DW 8

HEAD    202Q,'B',314Q,BLAN,$CON                 ; BL
DW 40Q

HEAD    202Q,'C',314Q,$CL,$CON                  ; C/L
DW 64  ;CHAR# PER LINE

HEAD    202Q,'1',313Q,BBUF,$CON                 ; BBUF
DW 1024

** USER-паременные **

HEAD    202Q,'S',260Q,SZERO,$USE                ; S0
DW 6

HEAD    202Q,'R',260Q,RZERO,$USE                ; R0
DW 8

HEAD    203Q,'TI',302Q,TIB,$USE                 ; TIB
DW 12Q  ;Входной буфер терминала

HEAD    205Q,'WIDT',310Q,$WIDTH,$USE            ; WIDTH
DW 14Q

HEAD    207Q,'WARNIN',307Q,WARN,$USE            ; WARNING
DW 16Q

HEAD    205Q,'FENC',305Q,FENCE,$USE             ; FENCE
DW 20Q

HEAD    202Q,'D',320Q,DP,$USE                   ; DP
DW 22Q

HEAD    204Q,'VOC',314Q,VOCL,$USE               ; VOCL
DW 24Q

HEAD    205Q,'FIRS',324Q,FIRST,$USE             ; FIRST
DW 26Q

HEAD    205Q,'LIMI',324Q,LIMIT,$USE             ; LIMIT
DW 30Q

HEAD    203Q,'AT',322Q,ATR,$USE                 ; ATR
DW 32Q          ;Атрибут

HEAD    202Q,'P',307Q,PG,$USE                   ; PG
DW 34Q          ; Текущая страница

HEAD    203Q,'BL',313Q,BLK,$USE                 ; BLK
DW 36Q
```

```
HEAD    202Q,'I',316Q,IN,$USE                   ; IN
DW 40Q

HEAD    203Q,'PN',324Q,PNT,$USE                 ; PNT
DW 42Q  ,       ;Флаг-печати

HEAD    203Q,'SC',322Q,SCR,$USE                 ; SCR
DW 44Q

HEAD    206Q,'OFFSE',324Q,OFSET,$USE            ; OFFSET
DW 46Q

HEAD    207Q,'CONTEX',324Q,CONT,$USE            ; CONTEXT
DW 50Q

HEAD    207Q,'CURREN',324Q,CURR,$USE            ; CURRENT
DW 52Q

HEAD    205Q,'STAT',305Q,STATE,$USE             ; STATE
DW 54Q ;0=> Исполнение

HEAD    204Q,'BAS',305Q,BASE,$USE               ; BASE
DW 56Q

HEAD    203Q,'DP',314Q,DPL,$USE                 ; DPL
DW 60Q

HEAD    203Q,'CS',320Q,CSP,$USE                 ; CSP
DW 64Q

HEAD    202Q,'R',243Q,RNUM,$USE                 ; R#
DW 66Q

HEAD    203Q,'HL',304Q,HLD,$USE                 ; HLD
DW 70Q

HEAD    203Q,'US',305Q,USE,$USE                 ; USE
DW 72Q

HEAD    204Q,'PRE',326Q,PREV,$USE               ; PREV
DW 74Q

HEAD    203Q,'$E',330Q,$EX,$USE                 ; EXP
DW 76Q

HEAD    203Q,'ER',302Q,ERB,$USE                 ; ERB
DW 100Q

** Слова высокого уровня **

HEAD    205Q,'?TER',315Q,?TERM,$COL             ; ?TERM
DW ?TER$,LIT,377Q,$AND,SEMI

HEAD    202Q,'C',322Q,CR,$COL                   ; CR
DW LIT,15Q,EMIT,LIT,12Q,EMIT,SEMI

HEAD    201Q,,254Q,COMMA,$COL                   ; ,
DW HERE,STORE,TWO,ALLOT,SEMI

HEAD    202Q,'C',254Q,CCOM,$COL                 ; C,
DW HERE,CSTOR,ONE,ALLOT,SEMI
```

```
HEAD    205Q,'SPAC',305Q,SPACE,$COL          ; SPACE
        DW BLAN,EMIT,SEMI

        HEAD    203Q,'PF',301Q,PFA,$COL              ; PFA
        DW ONE,TRAV,LIT,5,PLUS,SEMI

        HEAD    203Q,'?E',322Q,QERR,$COL             ; ?ER
        DW SWAP,ZBRAN,TTT-$,ERROR,SEMI
TTT:    DW DROP,SEMI

        HEAD    205Q,'?COM',320Q,QCOMP,$COL          ; ?COMP
        DW STATE,AT,ZEQU,LIT,21Q,QERR,SEMI

        HEAD    205Q,'?EXE',303Q,QEXEC,$COL          ; ?EXEC
        DW STATE,AT,LIT,22Q,QERR,SEMI

        HEAD    205Q,'?PAI',322Q,QPAIR,$COL          ; ?PAIR
        DW SUB,LIT,23Q,QERR,SEMI

        HEAD    204Q,'?CS',320Q,QCSP,$COL            ; ?CSP
        DW SPAT,CSP,AT,SUB,LIT,24Q,QERR,SEMI

        HEAD    205Q,'?LOA',304Q,QLOAD,$COL          ; ?LOAD
        DW BLK,AT,ZEQU,LIT,26Q,QERR,SEMI

        HEAD    207Q,'COMPIL',305Q,COMP,$COL         ; COMPILE
; Компиляция исполнительного адреса, следующего за оператором
        DW QCOMP,I,FROMR,TWOP,TOR,AT,COMMA,SEMI

        HEAD    204Q,'SMU',307Q,SMUG,$COL            ; SMUDGE
        DW LATES,BLAN,TOGL,SEMI

        HEAD    207Q,'(;CODE',251Q,PSCOD,$COL        ; (;CODE)
        DW FROMR,LATES,PFA,CFA,STORE,SEMI

        HEAD    207Q,'<BUILD',323Q,BUILD,$COL        ; <BUILDS
        DW ZERO,CON,SEMI

        HEAD    205Q,'COUN',324Q,COUNT,$COL          ; COUNT
        DW DUBL,ONEP,SWAP,CAT,SEMI

        HEAD    204Q,'TYP',305Q,$TYPE,$COL           ; TYPE
        DW DDUP,ZBRAN,TC1-$,ZERO,XDO
TC0:    DW DUBL,I,PLUS,CAT,LIT,177Q,$AND
        DW ONE,EMI$,XLOOP,TC0-$
TC1:    DW DROP,SEMI

        HEAD    204Q,'(."',251Q,PDOTQ,$COL           ; (.")
        DW I,COUNT,DUBL,ONEP
        DW FROMR,PLUS,TOR,$TYPE,SEMI

        HEAD    302Q,'."',242Q,DOTQ,$COL             ; ."
        DW LIT,34,STATE,AT,ZBRAN,XT-$
        DW COMP,PDOTQ,$WORD,HERE,CAT,ONEP
        DW ALLOT,SEMI
XT:     DW $WORD,HERE,COUNT,$TYPE,SEMI

        HEAD    205Q,'QUER',331Q,QUERY,$COL          ; QUERY
        DW TIB,AT,CFA,LIT,120Q,EXPE,ZERO,IN,STORE,CR,SEMI
        HEAD    301Q,,200Q,NULL,$COL                 ; NULL
        DW BLK,AT,ZBRAN,NUL-$
        DW ONE,BLK,PSTOR,ZERO,IN,STORE,QEXEC
```

```
NUL:        DW LEV,SEMI

            HEAD    203Q,'PA',304Q,PAD,$COL                 ; PAD
            DW HERE,LIT,104Q,PLUS,SEMI

            HEAD    204Q,'WOR',304Q,$WORD,$COL              ; WORD
            DW BLK,AT,DDUP,ZBRAN,WD1-$,BLOCK,BRAN,WD2-$
WD1:        DW TIB,AT
WD2:        DW IN,AT,PLUS,SWAP,ENCL,HERE,BLAN,BLANK,IN
            DW PSTOR,TOR,I,HERE,CSTOR
            DW HERE,ONEP,FROMR,CMOVE,SEMI

            HEAD    210Q,'(NUMBER',251Q,PNUMB,$COL          ; (NUMBER)
BN:         DW ONEP,TOR,I,CAT,BASE,AT,DIGIT,ZBRAN,MMO-$
            DW SWAP,BASE,AT,USTAR,DROP,ROT,BASE,AT,USTAR
            DW DPLUS,DPL,AT,ONEP,ZBRAN,BN1-$,ONE,DPL,PSTOR
BN1:        DW FROMR,BRAN,BN-$
MMO:        DW FROMR,SEMI

            HEAD    205Q,'-FIN',304Q,DFIND,$COL             ; -FIND
            DW BLAN,$WORD,HERE,COUNT,UPPER,HERE,CONT,AT,AT
            DW PFIND,DDUP,ZEQU,ZBRAN,FIN-$,HERE,LATES,PFIND
FIN:        DW SEMI

            HEAD    206Q,'NUMBE',322Q,NUMB,$COL             ; NUMBER
            DW ZERO,$EX,STORE,BASE,AT,ZERO,ROT
            DW ONEP,DUBL,CAT,DUBL,LIT,53Q,EQUAL
            DW ZBRAN,NH1-$,DEC,DROP,BRAN,NH4-$
NH1:        DW DUBL,LIT,55Q,EQUAL,ZBRAN,NH2-$,DROP,DEC
            DW SWAP,DROP,ONE,SWAP,BRAN,NH4-$
NH2:        DW LIT,47Q,EQUAL,ZBRAN,NH3-$,OCTAL,ONEP
NH3:        DW ONEM
NH4:        DW LIT,-1,DPL,STORE,ZERO,ZERO,ROT,PNUMB,DUBL
            DW CAT,BLAN,SUB,ZBRAN,NH6-$
            DW DUBL,CAT,LIT,56Q,EQUAL,ZBRAN,EXP-$
            DW ZERO,DPL,STORE,PNUMB,DUBL,CAT,BLAN,SUB,ZBRAN,NH6-$
EXP:        DW DPL,AT,SWAP,DUBL,CAT,LIT
            DW 105Q,EQUAL,ZBRAN,ER1-$,ONEP,DUBL,CAT
            DW LIT,55Q,EQUAL,ZBRAN,NEMI-$,ONE,BRAN,NH0-$
NEMI:       DW DUBL,CAT,LIT,53Q,SUB,ZBRAN,PLU-$,ONEM
PLU:        DW ZERO
NH0:        DW SWAP,ZERO,ZERO,ROT,PNUMB,CAT,BLAN,EQUAL,ZBRAN,ER-$
            DW DROP,SWAP,ZBRAN,NH5-$,MINUS
NH5:        DW $EX,STORE,DPL,STORE,ZERO
NH6:        DW DROP,ROT,ZBRAN,NH7-$,DMINU
NH7:        DW ROT,BASE,STORE,SEMI
ER:         DW DDROP,DROP
ER1:        DW DDROP,DDROP,DROP,ZERO,ERROR,SEMI

            HEAD    205Q,'ERRO',322Q,ERROR,$COL             ; ERROR
            DW HERE,COUNT,$TYPE,PDOTQ
            DB 3,' ? '
            DW ERB,AT,ZBRAN,XER-$
            DW ZERO,ERB,STORE,DROP,SEMI
XER:        DW MESS,SPSTO,QUIT

            HEAD    203Q,'ID',256Q,IDDOT,$COL               ; ID.
            DW COUNT,LIT,37Q,$AND,$TYPE,SPACE,SEMI

            HEAD    206Q,'CREAT',305Q,CREAT,$COL            ; CREATE
            DW DFIND,ZBRAN,CRE-$,DROP,TWOP,NFA,IDDOT
            DW LIT,4,MESS
```

```
CRE:    DW HERE,DUBL,CAT,$WIDTH,AT,MIN,ONEP,ALLOT
        DW DUBL,LIT,240Q,TOGL,HERE,ONEM
        DW LIT,200Q,TOGL,LATES,COMMA,CURR,AT,STORE
        DW HERE,TWOP,COMMA,SEMI

        HEAD    311Q,'[COMPILE',335Q,BCOM,$COL      ; [COMPILE]
        DW DFIND,ZEQU,ZERO,QERR,DROP,COMMA,SEMI

        HEAD    307Q,'LITERA',314Q,LITER,$COL       ; LITERAL
        DW STATE,AT,ZBRAN,LIL-$,COMP,LIT,COMMA
LIL:    DW SEMI

        HEAD    310Q,'DLITERA',314Q,DLITE,$COL      ; DLITERAL
        DW STATE,AT,ZBRAN,DLI-$,SWAP,LITER,LITER
DLI:    DW SEMI

        HEAD    206Q,'?STAC',313Q,QSTAC,$COL        ; ?STACK
        DW SZERO,AT,CFA,SPAT,ULESS,ONE,QERR
        DW LIT,-200Q,SPAT,ULESS,TWO
        DW QERR,SEMI

        HEAD    211Q,'INTERPRE',324Q,INTER,$COL     ; INTERPRET
IT1:    DW DFIND,ZBRAN,IT3-$,STATE,AT,LESS
        DW ZBRAN,IT2-$,COMMA,BRAN,IT5-$
IT2:    DW EXEC,BRAN,IT5-$
IT3:    DW HERE,NUMB,DPL,AT,ONEP,ZBRAN,IT4-$,DLITE,BRAN,IT5-$
IT4:    DW DROP,LITER
IT5:    DW QSTAC,BRAN,IT1-$

        HEAD    211Q,'IMMEDIAT',305Q,IMMED,$COL     ; IMMEDIATE
        DW LATES,$CL,TOGL,SEMI

        HEAD 212Q,'VOCABULAR',331Q,VOCAB,$COL       ; VOCABULARY
        DW BUILD,LIT,120201Q,COMMA,CURR,AT,CFA,COMMA
        DW HERE,VOCL,AT,COMMA,VOCL,STORE,DOES
DOVOC   LABEL   FAR
        DW TWOP,CONT,STORE,SEMI

        HEAD    301Q,,250Q,PAREN,$COL               ; (
        DW LIT,51Q,$WORD,SEMI

        HEAD    204Q,'QUI',324Q,QUIT,$COL           ; QUIT
        DW ZERO,BLK,STORE,LBRAC
QUI:    DW RPSTO,CR,QUERY,INTER,STATE,AT
        DW ZEQU,ZBRAN,QUI-$,PDOTQ
        DB 3,' OK'
        DW BRAN,QUI-$

        HEAD    205Q,'ABOR',324Q,ABORT,$COL         ; ABORT
        DW SPSTO,DEC,CR,PDOTQ
        DB 17,'FORTH-PC IS HERE '
        DW FORTH,DEFIN,QUIT


        HEAD    202Q,'S',256Q,SPOT,$COL             ; S.
        DW DUBL,UDOT,SEMI

        HEAD    201Q,,252Q,STAR,$COL                ; *
        DW MSTAR,DROP,SEMI

        HEAD    204Q,'/MO',304Q,SLMOD,$COL          ; /MOD
        DW TOR,STOD,FROMR,MSLAS,SEMI
```

```
        HEAD    201Q,,257Q,SLASH,$COL                   ; /
        DW SLMOD,SWAP,DROP,SEMI

        HEAD    203Q,'MO',304Q,$MOD,$COL                ; MOD
        DW SLMOD,DROP,SEMI

        HEAD    205Q,'*/MO',304Q,SSMOD,$COL             ;. */MOD
        DW TOR,MSTAR,FROMR,MSLAS,SEMI

        HEAD    202Q,'*',257Q,SSLA,$COL                 ; */
        DW SSMOD,SWAP,DROP,SEMI

        HEAD    205Q,'M/MO',304Q,MSMOD,$COL             ; M/MOD
        DW TOR,ZERO,I,USLAS,FROMR,SWAP,TOR,USLAS
$MO:    DW FROMR,SEMI

        HEAD    215Q,'EMPTY-BUFFER',323Q,MTBUF,$COL ;EMPTY-BUFF
        DW FIRST,AT,LIT,3084,ERASE,SEMI

        HEAD    205Q,'FLUS',310Q,FLUSH,$COL             ; FLUSH
        DW LIMIT,AT,FIRST,AT,XDO
FLU:    DW I,AT,ZLESS,ZBRAN,FL1-$
        DW I,TWOP,I,X,ZERO,RW
FL1:    DW LIT,1028,XPLOO,FLU-$,MTBUF,SEMI

        HEAD    206Q,'BUFFE',322Q,BUFFE,$COL            ; BUFFER
        DW USE,AT,TOR,I
BR1:    DW PBUF,ZBRAN,BR1-$,USE,STORE
        DW I,AT,ZLESS,ZBRAN,BR2-$
        DW I,TWOP,I,X,ZERO,RW
BR2:    DW I,STORE,I,PREV,STORE,FROMR,TWOP,SEMI

        HEAD    205Q,'BLOc',313Q,BLOCK,$COL             ; BLOCK
        DW OFSET,AT,PLUS,TOR
        DW PREV,AT,DUBL,X,I,SUB,ZBRAN,BLC-$
BLO:    DW PBUF,ZEQU,ZBRAN,BCK-$
        DW DROP,I,BUFFE,DUBL,I,ONE,RW,CFA
BCK:    DW DUBL,X,I,SUB,ZEQU
        DW ZBRAN,BLO-$,DUBL,PREV,STORE
BLC:    DW FROMR,DROP,TWOP,SEMI

        HEAD    205Q,'.LIN',305Q,DLINE,$COL             ; .LINE
        DW TOR,$CL,BBUF,SSMOD,FROMR,PLUS,BLOCK
        DW PLUS,$CL,DTRAI,$TYPE,SEMI

        HEAD    207Q,'MESSAG',305Q,MESS,$COL            ; MESSAGE
        DW IN,AT,RNUM,STORE,WARN,AT,ZBRAN,MS1-$
        DW DDUP,ZBRAN,MES-$,LIT,4,OFSET,AT,SUB,DLINE
MES:    DW SEMI
MS1:    DW PDOTQ

        DB 6,'MSG # '
        DW $DOT,SEMI

        HEAD    204Q,'LOA',304Q,LOAD,$COL               ; LOAD
        DW BLK,AT,IN,AT,ZERO,IN,STORE,ROT,BLK
        DW STORE,INTER,IN,STORE,BLK,STORE,SEMI

        HEAD    303Q,'--',276Q,ARROW,$COL               ; -->
        DW QLOAD,ZERO,IN,STORE,ONE
        DW BLK,PSTOR,SEMI
```

```
        HEAD    301Q,,247Q,TICK,$COL                    ; '
        DW DFIND,ZEQU,ZERO,QERR,DROP,TWOP,LITER,SEMI

        HEAD    206Q,'FORGE',324Q,FORGE,$COL            ; FORGET
        DW CURR,AT,CONT,AT,SUB,LIT,30Q,QERR,TICK
        DW DUBL,FENCE,AT,ULESS,LIT,25Q,QERR,DUBL,NFA
        DW DP,STORE,LFA,AT,CONT,AT,STORE,SEMI

        HEAD    204Q,'BAC',313Q,BACK,$COL               ; BACK
        DW HERE,SUB,COMMA,SEMI

        HEAD    305Q,'BEGI',316Q,BEGIN,$COL             ; BEGIN
        DW QCOMP,HERE,ONE,SEMI

        HEAD    304Q,'THE',316Q,THEN,$COL               ; THEN
        DW QCOMP,TWO,QPAIR,HERE,OVER,SUB,SWAP,STORE,SEMI

        HEAD    302Q,'D',317Q,$DO,$COL                  ; DO
        DW COMP,XDO,HERE,THREE,SEMI

        HEAD    304Q,'LOO',320Q,LOOP,$COL               ; LOOP
        DW THREE,QPAIR,COMP,XLOOP,BACK,SEMI

        HEAD    305Q,'+LOO',320Q,PLOOP,$COL             ; +LOOP
        DW THREE,QPAIR,COMP,XPLOO,BACK,SEMI

        HEAD    305Q,'UNTI',314Q,UNTIL,$COL             ; UNTIL
        DW ONE,QPAIR,COMP,ZBRAN,BACK,SEMI

        HEAD    306Q,'REPEA',324Q,REPEAT,$COL           ; REPEAT
        DW ROT,ONE,QPAIR,ROT,COMP,BRAN,BACK
        DW THEN,SEMI

        HEAD    302Q,'I',306Q,$IF,$COL                  ; IF
        DW COMP,ZBRAN,HERE,ZERO,COMMA,TWO,SEMI

        HEAD    304Q,'ELS',305Q,$ELSE,$COL              ; ELSE
        DW TWO,QPAIR,COMP,BRAN,HERE,ZERO,COMMA
        DW SWAP,TWO,THEN,TWO,SEMI

        HEAD    305Q,'WHIL',305Q,WHILE,$COL             ; WHILE
        DW $IF,SEMI

        HEAD    206Q,'SPACE',323Q,SPACS,$COL            ; SPACES
        DW ZERO,MAX,DDUP,ZBRAN,SP1-$,ZERO,XDO
SPA:    DW SPACE,XLOOP,SPA-$
SP1:    DW SEMI

        HEAD    202Q,'<',243Q,BDIGS,$COL                ; <#
        DW PAD,HLD,STORE,SEMI

        HEAD    202Q,'#',276Q,EDIGS,$COL                ; #>
        DW DDROP,HLD,AT,PAD,OVER,SUB,SEMI

        HEAD    204Q,'SIG',316Q,SIGN,$COL               ; SIGN
        DW ROT,ZLESS,ZBRAN,SIG-$,LIT,55Q,HOLD
SIG:    DW SEMI

        HEAD    201Q,,243Q,DIG,$COL                     ; #
        DW BASE,AT,MSMOD,ROT,LIT,11Q,OVER,LESS
        DW ZBRAN,DIGI-$,LIT,7,PLUS
DIGI:   DW LIT,60Q,PLUS,HOLD,SEMI
```

```
        HEAD    202Q,'#',323Q,DIGS,$COL                 ; #S
DIS:    DW DIG,DUP2,$OR,ZEQU,ZBRAN,DIS-$,SEMI

        HEAD    203Q,'D.',322Q,DDOTR,$COL               ; D.R
        DW TOR,SWAP,OVER,DABS,BDIGS,DIGS,SIGN,EDIGS
        DW FROMR,OVER,SUB,SPACS,$TYPE,SEMI

        HEAD    202Q,'.',322Q,DOTR,$COL                 ; .R
        DW TOR,STOD,FROMR,DDOTR,SEMI

        HEAD    203Q,'U.',322Q,UDOTR,$COL               ; U.R
        DW ZERO,SWAP,DDOTR,SEMI

        HEAD    202Q,'D',256Q,DDOT,$COL                 ; D.
        DW ZERO,DDOTR,SPACE,SEMI

        HEAD    201Q,,256Q,$DOT,$COL                    ; .
        DW STOD,DDOT,SEMI

        HEAD    201Q,,277Q,QUEST,$COL                   ; ?
        DW AT,$DOT,SEMI

        HEAD    202Q,'U',256Q,UDOT,$COL                 ; U.
        DW ZERO,DDOT,SEMI

;       ** Вспомогательные процедуры **

        HEAD    204Q,'LIS',324Q,$LIST,$COL              ; LIST
        DW DEC,CR,DUBL,SCR,STORE,PDOTQ
        DB 3,'S# '
        DW $DOT,LIT,20Q,ZERO,XDO
LSTI:   DW CR,I,THREE,DOTR,SPACE
        DW I,SCR,AT,DLINE,XLOOP,LSTI-$,CR,SEMI

        HEAD    205Q,'INDE',330Q,INDEX,$COL             ; INDEX
        DW ONEP,SWAP,XDO
INDX:   DW CR,I,THREE,DOTR,SPACE,ZERO,I,DLINE
        DW XLOOP,INDX-$,SEMI

        HEAD    204Q,'TRI',317Q,TRIO,$COL               ; TRIO
        DW LIT,14Q,EMIT
        DW THREE,OVER,PLUS,SWAP,XDO
TRI:    DW I,$LIST,XLOOP,TRI-$,SEMI
        HEAD    205Q,'VLIS',324Q,VLIST,$COL             ; VLIST
        DW CONT,AT,AT
VL0:    DW CR,THREE,ZERO,XDO
VL1:    DW DUBL,IDDOT,LIT,15Q,OVER,CAT,LIT,37Q,$AND,SUB
        DW SPACS,PFA,DUBL,LIT,6,UDOTR,SPACE,LIT,41Q
        DW EMIT,SPACE,LFA,AT,DUBL,ZEQU,ZBRAN,VL2-$
        DW LEAVE
VL2:    DW XLOOP,VL1-$,DDUP,ZEQU,ZBRAN,VL0-$,SEMI

        HEAD    203Q,'LC',314Q,LCL,$COL                 ; LCL
        DW ZERO,ZERO,FIX,$CL,SPACS,RC,SEMI

        HEAD    202Q,'H',324Q,HT,$COL                   ; HT
        DW ZERO,ZERO,FIX,SEMI

        HEAD    204Q,'COP',331Q,COPY,$COL               ; COPY
        DW SWAP,BLOCK,CFA,STORE,UPDAT,FLUSH,SEMI
```

```
        HEAD    202Q,'T',331Q,TY$$,$COL                 ; TY
        DW ZERO,XDO
TY4:    DW I,EIGHT,$MOD,ZEQU,ZBRAN,TY5-$
        DW CR,DUBL,LIT,7,UDOTR
TY5:    DW DUBL,AT,LIT,7,UDOTR,TWOP,XLOOP,TY4-$,SEMI

        HEAD    205Q,'DEPT',310Q,DEPTH,$COL             ; DEPTH
        DW SZERO,AT,SPAT,TWOP,SUB,DIV2,SEMI

        HEAD    204Q,'DUM',320Q,DUMP,$COL               ; DUMP
        DW ZERO,XDO
DMP:    DW DUBL,LIT,7,UDOTR,SPACE,EIGHT,ZERO,XDO
DP1:    DW DUBL,I,PLUS,CAT,LIT,5,DOTR,XLOOP,DP1-$
        DW LIT,4,SPACS,EIGHT,ZERO,XDO
DP2:    DW DUBL,I,PLUS,CAT,LIT,177Q,$AND,DUBL,BLAN
        DW LESS,ZBRAN,DP3-$,DROP,LIT,56Q
DP3:    DW EMIT,XLOOP,DP2-$,CR,EIGHT,PLUS,EIGHT
        DW XPLOO,DMP-$,DROP,SEMI
        ;
        HEAD    204Q,'SWA',323Q,SWAS,$COL               ; SWAS
        DW TOR,BLOCK,CFA,DUBL,AT,I,BLOCK,CFA,STORE,UPDAT
        DW FROMR,LIT,100000Q,$OR,SWAP,STORE,FLUSH,SEMI

        HEAD    203Q,'ST',331Q,STY,$COL                 ; STY
        DW DEPTH,DDUP,ZBRAN,STY3-$,ZERO,XDO
STY1:   DW I,EIGHT,$MOD,ZEQU,ZBRAN,STY2-$,CR
STY2:   DW I,ONEP,PICK,LIT,7,UDOTR,XLOOP,STY1-$
STY3:   DW SEMI

        HEAD    202Q,'O',256Q,ODOT,$COL                 ; O.
        DW BASE,AT,OVER,OCTAL,UDOT,BASE,STORE,SEMI

        HEAD    203Q,'R/',327Q,RW                       ; R/W
; Открытие файла
        MOV  DX, OFFSET FCB
        MOV  AH, 0FH
        INT  21H
        CMP  AL, 0FFH
        JE   ERR0
        POP  BX  ; R/W - флаг
        POP  AX  ; Номер блока
        POP  DX  ; Адрес буфера
        DEC  AX
        MOV  CL, 3
        SAL  AX, CL  ; (BLOCK#-1)*8
        MOV  RANDREC,AX
        MOV  RANDREC+2,0
        CMP  BX, 0
        JNE  RED
; WRITE
        MOV  BX, DX
        MOV  CX, 8       ; Номер записи
WR:     MOV  DX, BX
        MOV  AH,1AH      ; Запись адреса буфера
        INT  21H
        MOV  DX,OFFSET FCB
        MOV  AH, 22H     ; Запись RECORD
        INT  21H
        CMP  AL, 0
        JNE  ERR1
        INC  RANDREC     ; Коррекция адреса буфера
        ADD  BX, 80H
```

```
            LOOP WR
            JMP  OUT
RED:        MOV  CX, 8
            MOV  BX, DX
RD:         MOV  DX, BX
            MOV  AH, 1AH     ; Запись адреса буфера
            INT , 21H
            MOV  DX, OFFSET FCB
            MOV  AH, 21H
            INT  21H
            CMP  AL, 0
            JNE  ERR3
            INC  RANDREC     ; Коррекция адреса
            ADD  BX, 80H
            LOO  RD
OUT:        MOV  DX, OFFSET FCB  ; Закрытие файла
            MOV  AH, 10H
            INT  21H
            CMP  AL, 0
            JNE  ERR2
EXIT:       NEXT
ERR0:       MOV  DX, OFFSET ERMES0
            JMP  DONE
ERR1:       MOV  DX, OFFSET ERMES1
            JMP  DONE
ERR2:       MOV  DX, OFFSET ERMES2
            JMP  DONE
ERR3:       MOV  DX, OFFSET ERMES3
DONE:       MOV  AH, 9H
            INT  21H
            JMP  EXIT

            HEAD    305Q,';COD',305Q,SEMIC,$COL          ; ;CODE
            DW QCSP,COMP,PSCOD,LBRAC,SMUG,SEMI

            HEAD    305Q,'FORT',310Q,FORTH,$DOE          ; FORTH
            DW DOVOC.120201Q,TASK-7
XVOC        LABEL   FAR
            DW 0

            HEAD    204Q,'TAS',313Q,TASK,$COL            ; TASK
            DW SEMI

FCB         LABEL WORD
DRIVE       DB  1
FN          DB  'FORTH   '
EXT         DB  'DAT'
CURBLK      DW  0            ; Относительное начало файла
RECSIZE     DW  80H
FILESIZE    DW  5000,0
DATE        DB  0,0
            DB  0,0,0,0,0, 0,0,0,0,0
CURREC      DB  0
RANDREC     DW  0,0
ERMES0      DB  'ERR OPENING FILE$'
ERMES1      DB  'ERR WRITING FILE$'
ERMES2      DB  'ERR CLOSING FILE$'
ERMES3      DB  'ERR READING FILE$'

XDP     DW  16000 DUP(0)            ; DICTIONARY

STACK   SEGMENT STACK               ; Стек параметров
```

```
        DW   64 DUP (?)
XSO     LABEL   WORD
        DW   0,0    ; STACK

STACK   ENDS

ARRAY     ENDS
END       $INI
```

Приложение 10

Слово-декомпилятор

(Применимо для структур словаря, описанного в книге. Работает со словами типа "двоеточие".)

```
: DECOD 0 2 !       ( ячейка с абсолютным адресом 2 используется в
                      качестве рабочей [счетчик слов])
        [COMPILE] '                    ( определение PFA слова  XXX )
        DUP CFA @       ( определение адреса программы, на который
                          указывает CFA слова "XXX")
        ' :                                ( в стеке PFA слова ":")
        18 + =                               ( Это $COL [DOCOL] ?)
        IF                                  ( структура : NN ...;)
           58 EMIT SPACE                ( распечатка на экране  ":")
           DUP NFA          ( определение адреса имени слова  XXX )
           ID.                                      ( печать имени)
           BEGIN 0 0 !     ( ячейка с абсолютным адресом "0" также
            используется в качестве рабочей. Это плохо, но так не
            хочется заводить новые переменные в словаре)
                DUP @ 2+      ( вычисление PFA очередного слова в
                              списке-описании слова  XXX )
                DUP ' LIT =                      ( оператор LIT?)
                IF 1 0 !        ( если да, запись 1 в "0"-ячейку)
                ELSE DUP ' (.") =                    ( это (.") ? )
                     IF 2 0 !                ( если да, то запись 2 в
                                               "0"-ячейку)
                     THEN
                THEN NFA ID.       ( нахождение и распечатка имени
                                     очередного слова из списка-описания "XXX")
                     1 2 +!        ( добавление 1 к счетчику слов)
                     2+ 0 @ 1 =             ( содержимое "0"-ячейки
                                              равно 1 ?)
                     OVER @ ABS  512 < OR ( может быть, это одна
                                           из команд ветвления ?)
                     IF DUP @ .      ( распечатка цифры, следующей
                                       за командой)
                        2+           ( переход к следующему слову)
                     THEN 0 @ 2 = ( последнее слово было (.") ?)
                     IF DUP COUNT TYPE          ( распечатка строки
                                                  символов)
                        DUP C@ EVEN +    ( вычисление указателя с
                                           учетом длины строки)
                        SPACE                    ( Введение пробела)
                     THEN DUP @ 2+ ' ;S OVER =  ( очередное сло-
                                                  во ";S" ? )
                          SWAP ' QUIT = OR          ( "QUIT" ? )
                          2 @ 10 MOD 0= IF CR THEN
                            ( ввод кода перевода строки, если на
                                 строке уже отображено 10 слов)
           UNTIL 59 EMIT                    ( распечатка символа ";")
        ELSE ." PRIMITIVE"          ( распечатка производится каждый
                                      раз, когда встретилось нечто отличное от : NNN...;)
        THEN CR ;
```

СПИСОК ЛИТЕРАТУРЫ

1. **Brodie L.** Staring FORTH. — Prentice-Hall, 1981.
2. **Vickers S.** FORTH Pocket Guide. — Pitman Press, 1984.
3. **FORTH** inc. Using FORTH. — FORTH Inc., 1979.
4. **Scanlon L. J.** FORTH Programming. — H. W. Sams, Indianapolis, 1983.
5. **Salman W. P.** FORTH. — Macmillan Publishers, Computer Science Series, 1984.
6. **Haydon** All About FORTH. — Mountain View Press, 1983.
7. **Hicks S. M.** FORTH: A Cost Saving Approach to Software Development. — Wescon /Los Angeles, 1978.
8. **Hogan T.** Discover FORTH: Learning and Programming the FORTH. — Osborn: McGraw-Hill, 1982.
9. **Katzan H.** Invitation to FORTH. — Petrocelli Books, 1981.
10. **Knecht K. B.** Introduction to FORTH. — H. W. Sams, 1982.
11. **Loeliger R. G.** Threaded Interpretive Languages. — Byte Books, 1981.
12. **Ting C. H.** System Guide to FORTH. — Offete Enterprise Inc., 1983.
13. **Winfield A.** Complete FORTH. — Sigma Technical Press, 1983.
14. **Kelly Mahlon G., Spies N.** FORTH. A Text and Reference. — Prentice-Hall, 1986.
15. **Tello E.** PolyFORTH and PC/FORTH//BYTE. — 1984. — Vol. 9, N 12. — P. 301—313.
16. **Walker R. V., Rather E. D.** PolyFORTH II: Reference Manual. — S. L. FORTH Inc., 1983. — 524 p.
17. **Ragsdale W. F.** FIG-FORTH Installation Manual: Classary. — San Carlos: FORTH Interest Group, 1980. — 61 p.
18. **FORTH-83** Standard. — Mountain View (USA): FORTH Standard Team, 1983. — 82 p.
19. **Reynolds A. J.** Advanced FORTH. — Wilmslow: Sigma Press Publishers. — 1986.
20. **Rather E., Brodie L.** Using FORTH. — FORTH Inc., 1980.
21. **DECUS** FORTH Programming System for PDP-11. — 1975. (DECUS, № 11—232).
22. **Семенов Ю. А.** Язык четвертого поколения FORTH. — Препринт. — М., 1984. — (ИТЭФ, № 30).
23. **FORTH** Extensibity. Or How to Write a Compiler in 25 Words or Less. Kim Harris. — Byte Publishers Inc., 1980.
24. **Вульф А.** Операционные системы реального времени в русле развития вычислительной техники //Электроника. — 1985. — № 17. — С. 46—56.
25. **MVP-FORTH** PADS. — Mountain View Press Inc., 1983.
26. **MVP-FORTH** EXPERT System. — Mountain View Ptess Inc., 1984.
27. **Семенов Ю. А.** Управляющий язык FORTH: Рекомендации по программированию. — Препринт. — М., 1984. — (ИТЭФ, № 63).
28. **Семенов Ю. А.** Системный пакет программ для FORTH. — Препринт. — М., 1985. — (ИТЭФ, № 72).
29. **Семенов Ю.А.** Экранный редактор для системы FORTH. — Препринт. — М., 1985. — (ИТЭФ, № 73).
30. **Семенов Ю. А.** Новая версия системы FORTH. — Препринт. — М., 1986. — (ИТЭФ, № 25).
31. **Семенов Ю. А.** Программные средства ЭВМ "Электроника-60" для работы с микропроцессорами //12-й Международ. симп. по ядерной электронике. — Дубна, 1985.
32. **Диалоговые** микропроцессорные системы /Под ред. Н. П. Брусенцова и А. М. Шаумана. — М.: Изд-во МГУ, 1986. — 148 с.

33. **Семенов Ю. А.** О возможности FORTH при автоматизации физического эксперимента //4-й Всесоюз. семинар по автоматизации научных исследований в ядерной физике и смежных областях. — Протвино, 1986.
34. **Семенов Ю. А.** Программа-справочник по цифровым интегральным схемам на языке Форт. — Препринт. — М., 1987. — (ИТЭФ, № 97).
35. **Баранов С. Н., Ноздрунов Н. Р.** Язык Форт и его реализации. — Л.: Машиностроение, 1988. — (Сер. ЭВМ в производстве).
36. **Баранов С. Н.** Система Форт-ЕС. —Препринт. — Л., 1986. — (ЛИИАН, № 1).
37. **Демидович Б. П., Марон И. А.** Основы вычислительной математики. — М.: Наука, 1966.
38. **RT-11** Software Support Manual. — DES Maynard Massachusetts, 1980.
39. **Семенов Ю. А.** Многозадачная версия Форт. — Препринт. — М., 1988. — (ИТЭФ, № 167).
40. **Барни К.** Плата Форт-процессора для персонального компьютера PC IBM //Электроника. — 1985, — № 17. — С. 46.
41. **Каррен К.** Новое изделие фирмы HARRIS, соединяющее в себе PISK-архитектуру и язык Форт //Электроника. — 1987. — № 11. — С. 6.
42. **ГОСТ 26.201—80.** Единая система стандартов приборостроения. Система КАМАК. Крейт и сменные блоки. Требования к конструкции и интерфейсу.
43. **OEM** Boards and System Handbook /Intel. — Santa Clara, 1987.
44. **Семенов Ю. А.** Ассемблер для микропрограммирования и его реализация на ЭВМ "Электроника-60". — Препринт. — М., 1984. — (ИТЭФ, № 154).
45. **PASHA** — an Approach to Computer-aided Hardware Debugging /R. C. Agostini, H. V. Walz, D. B. Gustavson, L. Barker. — SLAC-PUB-4766 (REV), 1988, Nov.
46. **Астановский Ф. Г., Ломунов В. Н.** Интерпретирующие системы программирования и их аппаратная поддержка. Процессор, ориентированный на язык Форт. — Таллинн: Ин-т кибернетики АН ЭССР, 1988.
47. **PDP-11** Peripherals Handbook. — DEC, 1984.

Ю. А. Семенов

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ФОРТ

Москва «Радио и связь»
1991