ВВЕДЕНИЕ
В ADVANCED FORTH
И ФОРТ-МАШИНЫ

ADVANCED FORTH

System. Machine. Much more.

Ред. 2.02 2020 г.

Введение в Advanced Forth и Форт-машины. Самоучитель начального уровня.

Редакция 2.02, ноябрь 2020 г.
Номер документа: TB00010202RU

Оглавление

Иллюстрации

Таблицы

1. Введение

В 1958-1959 г.г. в МГУ им. М.В. Ломоносова под руководством Н.П.Брусенцова была разработана и построена вычислительная машина "Сетунь", обладавшая рядом конструктивных особенностей, отличавших ее от ЭВМ "традиционной" архитектуры. Кроме использования "троичной" системы хранения чисел со знаком, это была стековая машина с постфиксной нотацией операндов.

Стек – особым образом организованная область памяти, предназначенная для хранения данных. Стек не требует присвоения имен переменным, также не требуется непосредственной адресации данных в командах машины. Стековая машина поддерживает использование множества таких областей памяти на аппаратном уровне.

Способ записи математических выражений в виде "x y +" называется постфиксной нотацией, поскольку символ операции (оператор) стоит после чисел (операндов). Постфиксная нотация известна также как обратная польская нотация (ОПН). Такое представление дает массу преимуществ при механизации вычислений: не требуются скобки, порядок действий определяется порядком следования операторов. ОПН является идеальным способом представления математических и логических выражений для обработки стековой машиной.

Машины "Сетунь" серийно производились в 1960-х, а к 1970 году была разработана следующая модель "Сетунь-70", в которой к указанным особенностям добавились развитое структурированное программирование с постфиксными процедурами и диалоговая среда программирования.

На западе также проявляли интерес к такого рода машинам. Стек и постфиксная нотация оказались очень удобными и востребованными в условиях ограниченности аппаратных ресурсов, что привело к попыткам создать универсальный язык программирования на их основе. Некоторые из таких языков просуществовали довольно долго, например POP-2 (в СССР - ПОПЛАН для БЭСМ-6).

Приблизительно в то же время компания Hewlett-Packard создает один из первых программируемых калькуляторов HP 9100A (1968 г.), использующий обратную польскую нотацию. Являясь инженерным чудом для своего времени, 9100A стал родоначальником модельного ряда калькуляторов HP.

В 1968-1970 г.г. при разработке программного обеспечения для радиотелескопа Национальной радиоастрономической обсерватории (США) была создана программа под названием FORTH. Автор программы, Чарльз Мур, впервые предложил упорядоченный список слов-процедур и детально описал язык программирования с постфиксной нотацией для стековой машины.

Принимая во внимание, что работы в этом направлении проводились и другими исследователями, Ч. Муру удалось создать самую сбалансированную и законченную концепцию системы. К 1971 г. идеи Ч. Мура оформились в коммерческий продукт, известный как язык программирования Forth. Была создана компания FORTH Inc., успешно действующая и в настоящее время.

Популярность нового языка стремительно росла. Уже к концу 1970-х годов имелись версии практически для всех существующих аппаратных платформ, а в 1979 г. был принят первый официальный стандарт языка.

Язык Forth с легкостью расширялся и адаптировался к новой аппаратуре. Один из ярких примеров применения Forth того времени – программное обеспечение компьютера Apple II. Ключевые прикладные программы для этого персонального компьютера были созданы с использованием адаптированной версии Forth, а специально разработанный "графический" диалект GraFORTH обеспечил Apple II анимированной графикой.

К началу 1980-х идеи, заложенные в ЭВМ "Сетунь", вернулись в СССР в виде популярного языка программирования Forth. В нашей стране чаще использовалось название Форт, а к середине 1980-х годов "русский" Форт превратился в самостоятельное ответвление стековых машин и систем программирования с постфиксной нотацией и применялся, в основном, для создания инструментальных средств выпускаемых в стране процессоров.

Примерно в это время такие машины стали называть просто Форт-машинами, а среду программирования языка Форт, объединенную с операционным ядром – Форт-системой.

С развитием технологии производства интегральных схем появилась возможность реализовать Форт-машину на аппаратном уровне, в виде специализированного процессора. В начале 1980-х Чарлз Мур разрабатывает и патентует архитектуры микропроцессоров с мультистековой организацией, а в 1983 г. создает компанию Novix Inc. Новая фирма вскоре выпускает процессор NC4000, лицензию на который приобретает Harris Semiconductor. На основе лицензии Harris производит Форт-процессор RTX2000, который длительное время используется NASA в вычислительных комплексах космических аппаратов.

В СССР с 1980 г. работы по Форт-системам получают поддержку на уровне Государственного комитета по науке и технике, в том числе ведется разработка специализированного Форт-процессора. В 1983 г. был принят новый стандарт Forth-83, на который ориентировались отечественные разработчики программного обеспечения.

К середине 1980-х в нашей стране существовали разновидности Форт-систем для всех производимых архитектур: от микропроцессора серии К580 до ЭВМ БЭСМ-6 и суперЭВМ "Эльбрус".

К середине 1990-х годов умами завладел Интернет, внимание специалистов сосредоточилось на задачах мультимедиа, web-дизайна, интернет-коммерции. Бурное развитие мобильной связи отодвинуло на второй план средства программирования для научно-прикладных задач и промышленного применения, к которым в большой степени относится Форт. Стоит отметить, что Форт мог быть успешно применен в качестве языка скриптов и управления активным содержимым web-страниц, но данную нишу сегодня занимают иные программные средства.

Несмотря на столь значительные перемены в компьютерной индустрии, Форт продолжает активно применяться. Язык PostScript, компьютер NeXT, архитектура PowerPC и многие другие значимые разработки не обошлись без его участия.

В 1994 году был принят официальный стандарт языка Форт ANSI X3.215-1994, а в 1997 году этот документ был также адаптирован как международный стандарт ISO.

Энтузиасты из NASA подсчитали в 2003 г., что системы на основе Форта, как программные, так и аппаратные, применялись (и продолжают использоваться) более чем в 60 космических миссиях и связанных проектах (в том числе и на космодроме Байконур).

Современные тренды – "Индустрия 4.0", роботика, искусственный интеллект, - требуют создания компактного и надежного программного обеспечения, обеспечивающего требуемое быстродействие и производительность. При этом все более жесткие требования предъявляются к срокам разработки программ.

Можно предположить, что основная конкуренция среди разработчиков и производителей средств вычислительной техники в ближайшие годы развернется в области новых технологий, альтернативных компьютерных архитектур, алгоритмов, программного обеспечения.

Вместе с тем хорошо известно, что эффективность программного обеспечения тем выше, чем ниже уровень используемого языка программирования.

Именно поэтому интерес к Форт-машинам и Форт-системам возрождается вновь. Форт – единственный язык программирования, имеющий качество языка высокого уровня, но при этом максимально приближенный к уровню машинного кода. Форт снова становится актуальным и востребованным.

Это пособие поможет вам быстро освоить программирование на Форте и начать использовать в своих разработках новый класс интеллектуальных устройств — Форт-машины.

Данный курс предназначен для пользователей, не имеющих какой-либо подготовки. По мере освоения учебного материала вводного курса необходимо дополнительно обращаться к специальной документации.

Если вы уже имели возможность поработать с какой-либо Форт-системой, это пособие необходимо изучить для понимания различия между исходным Фортом и Advanced Forth System (AFS).

Если вы новичок в этой области, информацию о языке Форт, истории его создания, существующих разработках Форт-систем можно найти в Интернете. Ссылки на некоторые интересные страницы имеются на нашем сайте в разделе поддержки пользователей (см. “Контакты и полезные ссылки” на стр. 9).

Наиболее популярными публикациями по "классическому" языку Форт считаются статьи и книги Лео Броуди, например, "Thinking FORTH. A Language and Philosophy for Solving Problems". Одна из книг была переведена на русский язык: "Начальный курс программирования на языке Форт" (Л. Броуди, Москва, "Финансы и статистика", 1990).

Из отечественных изданий представляет интерес книга "Язык Форт и его реализации" (С.Н. Баранов, Н.Р. Ноздрунов, Ленинград, издательство "Машиностроение", 1988).

Также рекомендуем книгу "FORTH - A Text and Reference". Книга переведена на русский язык (М. Келли, Н. Спайс. "Язык программирования Форт", Москва, издательство "Радио и связь", 1993).

1.1. Что такое Advanced Forth?

Многолетняя практика разработки и применения Форт-систем показала, что прямое следование устаревшему стандарту не позволяет реализовать всю мощь Форта на новой аппаратуре.

"Продвинутый Форт" (Advanced Forth) появился в процессе разработки современной Форт-системы AFS (Advanced Forth System). Эта система объединяет высокопроизводительное операционное ядро, драйверы устройств целевой платформы, интерактивную среду программирования, средства межмашинного обмена данными и многое другое. Все части AFS оптимизированы для эффективного выполнения метакода Advanced Forth.

Язык программирования Advanced Forth является результатом развития прогрессивных идей, заложенных в классическом Форте, но при этом он избавлен от анахронизмов официального стандарта.

Мы будем использовать название "Форт", когда речь идет о языке программирования, кроме случаев, когда надо указать на различие между классическим Фортом и Advanced Forth. Вместо полного наименования Advanced Forth может быть использовано сокращение AF, соответственно вместо слов Форт-система — AFS.

1.2. Что такое Форт-система?

Как было сказано выше, Advanced Forth System, или AFS, это программный комплекс, который является и операционной системой, и средой разработки программ, и транслятором языка программирования, и средством интерактивного управления.

За годы развития AFS получила множество вариантов реализации: система поддерживает процессоры с различной архитектурой, обеспечивает параллельное выполнение на уровне нитей и процессов, устанавливается в однокристальные микроконтроллеры и компьютеры.

Все это многообразие подчиняется строгому закону. Любой вариант AFS собирается из заранее отлаженных модулей, оптимизированных для использования на конкретной аппаратуре.

Состав системы и ее возможности определяются доступными ресурсами целевой аппаратной платформы. Если в компьютерах с большими объемами памяти возможности AFS практически не ограничены, то в однокристальных контроллерах с небольшой встроенной памятью используются специальные сборки AFS, обеспечивающие высокую эффективность при выполнении ограниченного круга задач.

Для простоты ориентирования в вариантах исполнения AFS было введено понятие "интеллектуального уровня" системы. Чем ниже значение уровня, тем меньше возможностей предоставляется прикладному программному обеспечению. Разумеется, системы с "низким уровнем интеллекта" требуют меньший объем памяти для размещения и могут быть установлены на менее дорогостоящих аппаратных платформах.

"Уровень интеллекта" AFS обозначается от L0 (Level-0) до L3 (Level-3)

Системы нижних уровней предназначены для встраивания в микроконтроллеры различного класса, системы с высоким "уровнем интеллекта" предназначены для построения мультипроцессорных и многомашинных комплексов.

Более детальную информацию о различиях AFS разных уровней содержит Приложение А.

1.3. Что такое Форт-машина?

Форт-машина — это универсальная вычислительная машина, интегрированная с Форт-системой, архитектура которой оптимизирована для эффективного выполнения метакода языка Форт. Благодаря простоте и гибкости, такая машина может быть реализована самыми разными способами: аппаратным Форт-процессором или на ПЛИС, на специально разработанном компьютере или адаптированной ЭВМ массового производства, в однокристальном микроконтроллере.

Различные реализации, использующие AFS, составляют семейство Форт-машин под общим названием AFM (Advanced Forth Machine).

Как и в случае с AFS, для AFM введены классификаторы модельного ряда, отвечающие за уровень сложности Форт-машины. Модели AFM обозначаются M0, M1, M2 и т д.

Современные RISC-процессоры являются удобной аппаратной базой для создания высокоэффективной Форт-машины. Особенности RISC-архитектуры позволяют реализовать ядро Форт-системы с минимальным объемом машинного кода.

В настоящее время в семействе AFM используются, в основном, однокристальные микроконтроллеры с ядром ARM различных модификаций, для каждой из которых разработана и оптимизирована своя версия AFS.

Форт-машины на базе однокристальных микроконтроллеров занимают нижний уровень иерархии AFM (модели M0 – M9). Такие Форт-машины поставляются в виде

запрограммированных микросхем, защищенных от считывания и модификации программ AFS. Для удобства разработки и мелкосерийного производства изделий микросхемы монтируются на печатные платы модулей.

В зависимости от числа используемых выводов микроконтроллера и размеров дополнительных схем, модули AFM выпускаются в нескольких исполнениях – AFMnano, AFMmicro, AFMstandard. Изготавливаются также специальные изделия на базе Форт-машин.

На рисунке представлены разнообразные конструктивные исполнения Форт-машин AFM.

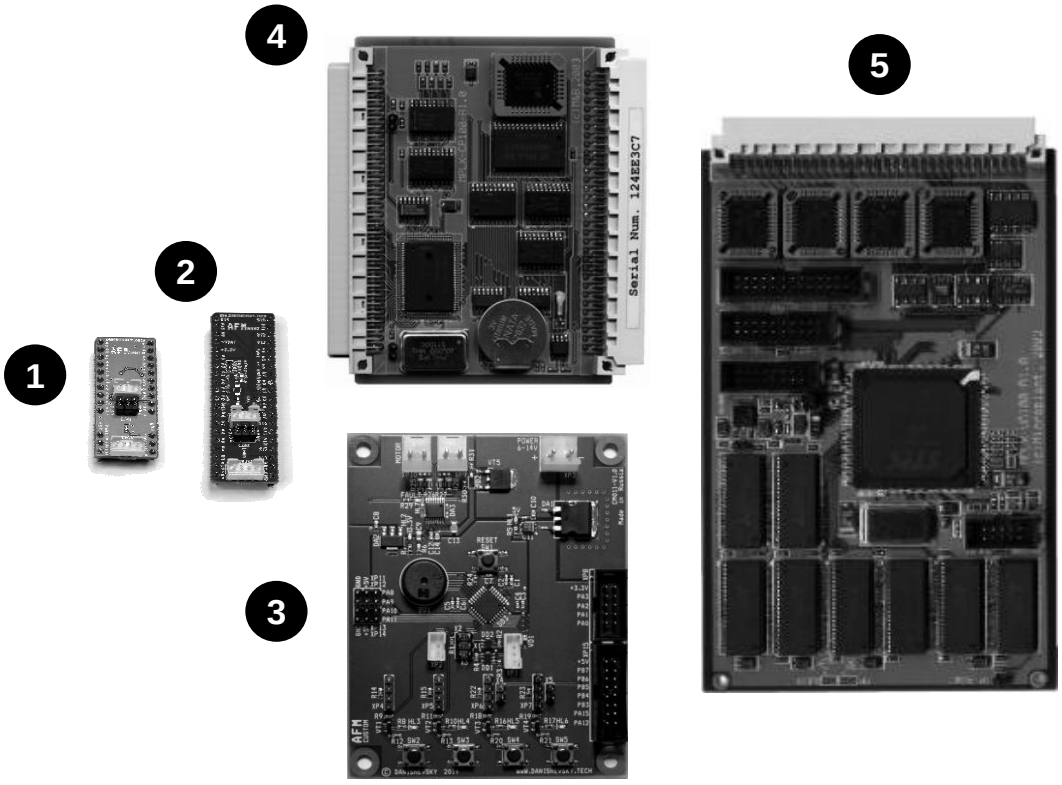


Рисунок 1.1 Различные исполнения Форт-машин AFM

Цифрами 1 и 2 отмечены модули AFMnano M0.10 и M2.10 соответственно. Это самые простые и доступные варианты Форт-машин.

Под номером 3 представлена Форт-машина AFMnano-M1.10 (микросхема в центре), вокруг которой построена специализированная управляющая плата AFMcustom-M1.10. На плате установлены стабилизаторы напряжения питания, драйвер двигателей постоянного тока, схемы сопряжения с датчиками, звуковая сигнализация и другие вспомогательные схемы. Такие Форт-машины, доработанные (кастомизированные) для решения определенного круга задач выпускаются в серии AFMcustom.

Номер 4 на рисунке – Форт-машина класса AFMstandard. Это более сложная система, построенная на мощном процессоре, имеющая большой объем памяти и высокое быстродействие. Такие Форт-машины работают под управлением AFS Level-3.

Под номером 5 представлен Форт-компьютер – мощная вычислительная система, включающая видеографическую подсистему и каналы связи. Плата выполнена в формате

Еврокарты и может быть использована как одноплатный компьютер либо в составе большого вычислительного комплекса.

1.4. Краткий обзор вводного курса

Представленный материал позволяет самостоятельно изучить язык программирования Advanced Forth и приобрести навыки работы с интерактивной системой программирования AFS, познакомиться с основами применения Форт-машин. Изложение ведется по принципу "от простого к сложному".

Начальные разделы представляют собой краткий ознакомительный курс языка Форт и основ представления информации в вычислительной технике, что позволяет подготовиться к изучению более серьезных вопросов пользователям, не имеющим опыта программирования ЭВМ.

Не пропускайте эти разделы, даже если вы профессиональный программист. В тексте вам будут встречаться ссылки на возможности языка Advanced Forth, аналогов которым вы не найдете в других языках программирования (включая стандартный Форт).

Лучший способ изучать AFS и язык Форт – выполнять приводимые в тексте примеры в настоящей Форт-машине. Минимально необходимый для работы комплект оборудования для начинающих называется "Старт AFM-0".

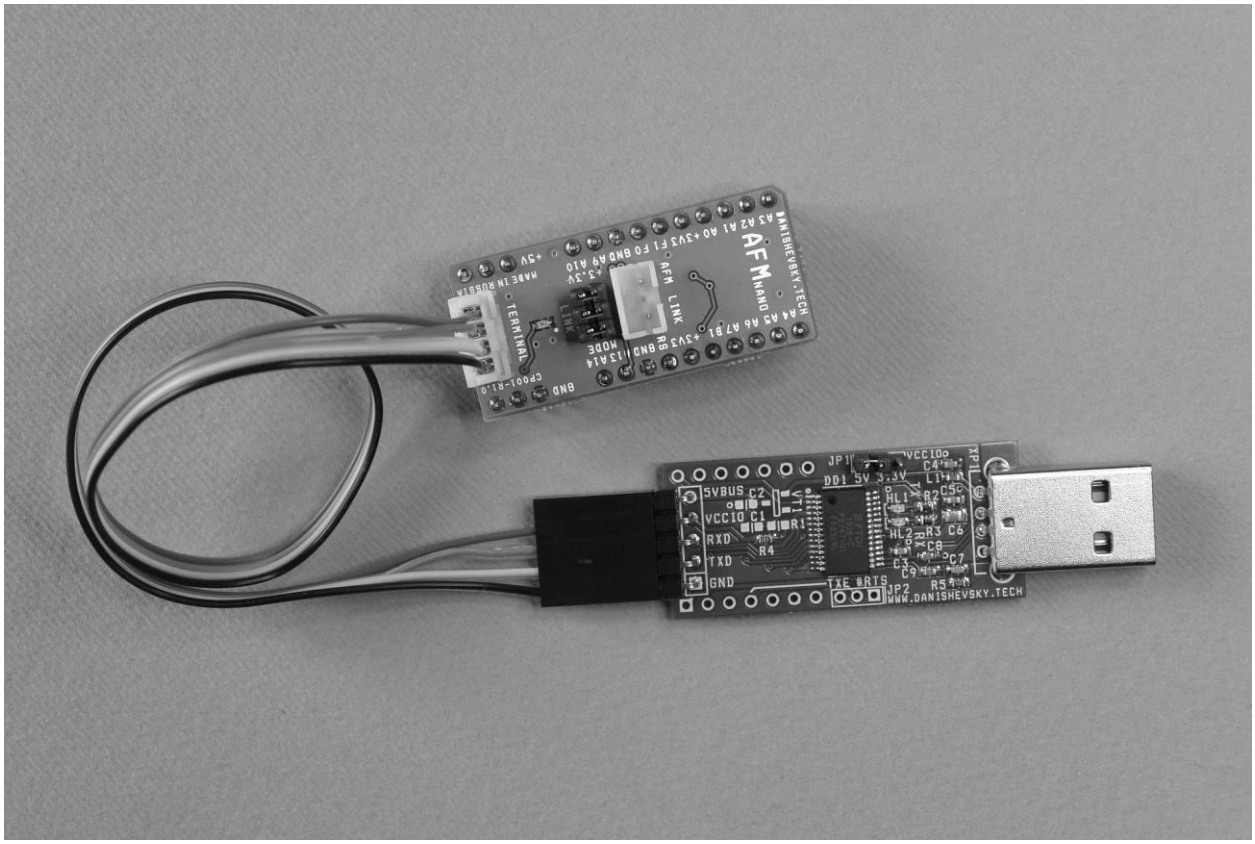


Рисунок 1.2 Комплект "Старт AFM-0"

На рисунке 1.2 представлен комплект "Старт AFM-0" с модулем AFMnano-M0 и конвертером для подключения к терминалу через USB. В качестве терминала используется любой персональный компьютер.

Мы рекомендуем использовать этот комплект для первого знакомства с возможностями Форт-систем.

Во второй части книги обсуждаются вопросы практического применения модуля Форт-машины, рассматриваются встроенные периферийные устройства и приводятся примеры программирования.

При рассмотрении примеров программирования встроенных устройств Форт-машины мы будем ориентироваться на модель AFMnano-M0.

Более сложные модули, например, AFMnano-M2, имеет смысл приобретать на следующем этапе, если вы планируете углубленное изучение AFS/AFM и дальнейшее практическое применение Форт-машин в своих разработках.

Если вы уже имеете опыт работы с микроконтроллерами и простыми комплектами на их основе, например, для моделей роботов, вам может быть интересна плата AFMcustom-M1.10 (см. Рисунок 1.1[№3]).

1.5. Замечания по оформлению текста

В тексте используются следующие способы представления символов, терминов и специальных слов языка программирования Форт:

наклонным шрифтом выделяются термины и важные пояснения, на которые следует обратить внимание;

терминальным шрифтом набраны слова языка и фрагменты текста программ, команды и другой текст, вводимые пользователем с клавиатуры, а также сообщения, выводимые на экран системой.

Жирным шрифтом отмечается информация, касающаяся отличия Advanced Forth от существующих стандартов Форт.

1.6. Контакты и полезные ссылки

Для получения актуальной информации и обмена мнениями о технологии AFS/AFM и другим продуктам от компании Danishevsky Technology, присоединяйтесь к нам в соцсетях:

"Вконтакте": https://vk.com/danishevsky.tech

"Facebook": https://www.facebook.com/Danishevsky.Tech

Для получения обновленной документации и другой полезной информации обращайтесь к нашим сайтам:

Поддержка пользователей: http://support.danishevsky.technology

Главная страница в Сети: http://www.danishevsky.tech

Онлайн-магазин: https://dt-store.shop/

Замечания по тексту вводного курса присылайте по адресу: Link@Danishevsky.tech

или пишите в комментариях к соответствующим постам в соцсетях.

2. Первое знакомство

Мы будем на практике изучать программирование на языке Advanced Forth в самой простой и доступной Форт-машине семейства AFM.

С самых первых страниц вам понадобится модуль AFMnano. Для общения с интерактивной системой программирования AFS, интегрированной в модуль, необходим последовательный терминал и кабель для подключения. Обычно это программный терминал для Windows или Unix (Linux), работающий с эмуляцией последовательного порта на USB. Преобразователь UART–USB входит в комплект для начинающих, при необходимости его можно приобрести отдельно.

К модулю прилагается описание и руководство по подключению к терминалу. Изучите прилагаемые документы и настройте ваш терминал для подключения (если под рукой нет документации, воспользуйтесь советами по настройке терминала далее по тексту).

Документацию на модули AFM, справочные руководства AFS и различные рекомендации можно также найти на нашем сайте в разделе поддержки пользователей.

2.1. Настройка программы терминала

Для работы с AFS/AFM подойдет любая программа терминала. Если вы пользуетесь UNIX (Linux) или MacOS то у вас, скорее всего, уже имеется нужное программное обеспечение.

Пользователям Windows рекомендуем свободно распространяемую программу Tera Term. Программа легко устанавливается и настраивается, адрес для загрузки дистрибутива и руководство по настройке вы найдете на нашем сайте.

Мы не будем здесь подробно разбирать все настройки Tera Term, остановимся лишь на главных параметрах, обеспечивающих работу с AFS/AFM.

В меню программы откройте пункт "Настройка" и найдите пункт "COM-порт…" (см. рисунок).

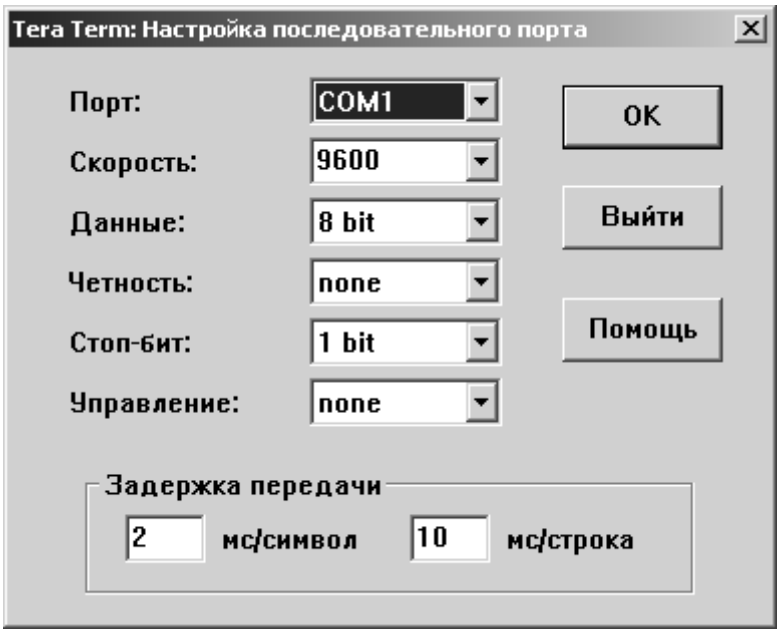


Рисунок 2.1 Настройка последовательного порта

В открывшемся окне выберите COM-порт, который вы будете использовать для связи с модулем AFM. Установите параметры канала связи: скорость 9600 bps, данные 8 бит, четность – нет, стоп-бит 1, управление - нет (как на рисунке).

Установите параметры "Задержка передачи". Вы должны задать задержку передачи строки 10 мс (не менее). Задайте задержку передачи символа 2 мс (не менее).

Если вы не установите эти значения, возникнут проблемы с передачей текста от модуля AFM в терминал. Дело в том, что модули AFM используют полудуплексный однопроводный интерфейс, т.е. канал связи, в котором данные передаются только в одном направлении в выбранный момент времени.

Вы можете копировать строки программ из любого текстового файла в терминал, и AFS будет воспринимать информацию так же, как вводимый с клавиатуры текст. Если же в ответ возвращаются случайные символы или часто возникают ошибки, это говорит о том, что полудуплексный канал не успевает переключаться с приема на передачу. Если время задержки передачи установлено правильно, такая проблема не возникает.

Закройте окно параметров COM-порта кнопкой OK. В меню "Настройка" найдите пункт "Терминал…" (см. рисунок).

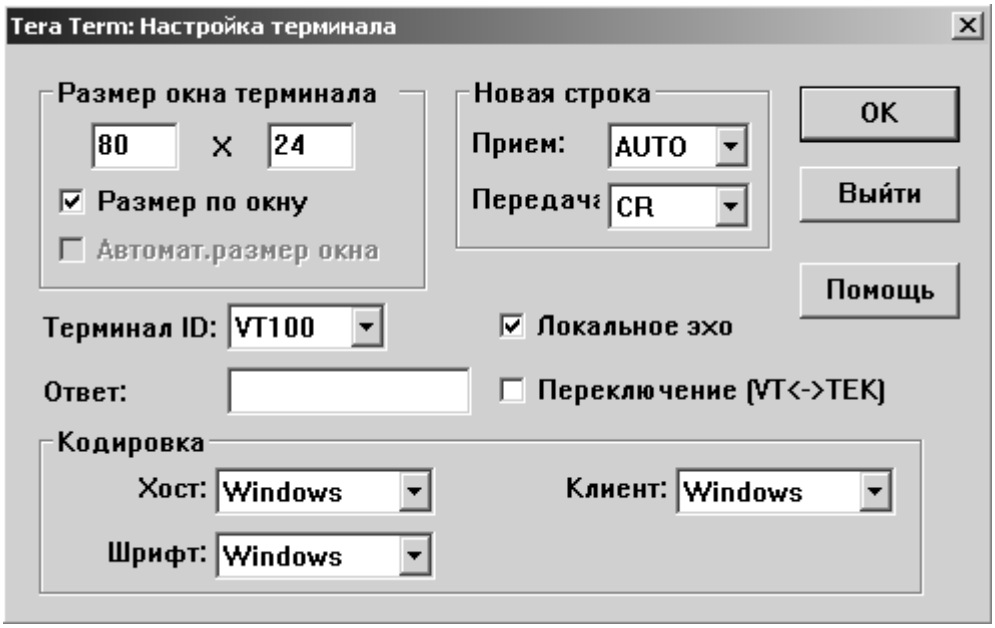


Рисунок 2.2 Настройка параметров терминала

Обязательно отметьте пункт "Локальное эхо" – это позволит видеть вам набираемые символы в полудуплексном канале AFM-Link. Другие настройки можно выставить, как показано на рисунке.

Правильно заданные параметры надо сохранить, чтобы они загружались при запуске терминала автоматически.

Найдите в меню "Настройка" пункт "Сохранить настройки…". Программа Tera Term предлагает записать настройки в свой файл по умолчанию, что и требуется для нормальной работы. Просто нажмите кнопку "Сохранить" и подтвердите перезапись существующего файла.

Указанные настройки должны быть установлены обязательно, независимо от предпочитаемого вами программного обеспечения. Большинство терминалов имеют

схожие пункты меню, вам не составит труда найти соответствующие разделы в используемой программе.

Возможно также изменение настроек режимов работы текстовой консоли AFS, связанных с передачей данных в последовательный терминал. На начальном этапе изучения Форт-системы лучше использовать установки по умолчанию, но если возникают проблемы с работой текстовой консоли, изучите также настройки на стр. 130.

2.2. Включаем Форт-машину

Проверьте установку перемычек на модуле AFM. Они должны быть установлены для работы в интерактивном режиме программирования AFS, как это указано в прилагаемом к модулю описании.

Подключите кабель AFM-Link к модулю AFM и порту USB вашего компьютера. Убедитесь, что эмуляция последовательного порта работает, а номер порта правильно задан в настройках программы терминала. Проверьте настройки скорости и другие параметры порта.

Запустите программу терминала и установите связь с Форт-машиной.

Если установить связь не удается, обратитесь к документации и еще раз проверьте все настройки.

2.2.1. Вход в сеанс интерактивного программирования AFS

При запуске AFS выдает на экран сообщение копирайта и данные о конфигурации AFM/AFS:

```
(c) DANISHEVSKY 2006-2020
AFMnanoM0.10 AFS16L0R1.10
```

Сообщение может отличаться в зависимости от модели AFM и версии AFS. Просто нажмите клавишу ввода (Enter) и на экране появится слово **Ok** .

Некоторые терминалы не показывают первое сообщение, выдаваемое Форт-машиной при подключении. Если на экране ничего нет, нажмите клавишу Enter – должно появиться слово **Ok** . Это означает, что сеанс интерактивного программирования AFS уже идет.

Если на экране вы видите что-то другое, для входа в сеанс могут потребоваться дополнительные действия.

2.2.2. Очистка памяти Форт-машины

Если вы пользуетесь модулем AFM, в котором уже сохранена программа пользователя или последний сеанс программирования, вам потребуется очистить память Форт-машины.

Если вы видите один из вариантов меню запуска

```
MEMORY OCCUPIED BY AFS SESSION
1= ERASE PROGRAM MEMORY
2= CONTINUE AFS SESSION
```

или

```
MEMORY OCCUPIED BY USER PROGRAM
1= ERASE PROGRAM MEMORY
2= COPY USER PROGRAM
```

выберите пункт 1.

Система запросит подтверждение:

```
ARE YOU SURE? [Y/N]:
```

Нажмите клавишу **Y**. Память будет очищена, произойдет перезагрузка Форт-системы и вы увидите сообщение

```
(c) DANISHEVSKY 2006-2020
AFMnanoM0.10 AFS16L0R1.10
```

Нажмите клавишу Enter – должно появиться слово **Ok** .

С этого момента мы будем считать, что все настроено правильно и вы готовы к программированию вашей Форт-машины.

2.3. Простейшие операции

Наберите на клавиатуре

```
12 345 +
```

и нажмите клавишу ввода (Enter).

На экране вы должны увидеть сообщение "Ok", означающее, что система нормально восприняла введенную информацию и готова к продолжению ввода данных. На первый взгляд ничего не произошло, но в действительности вы ввели в систему числа 12 и 345, сложили их, но не увидели результат, потому что не попросили вывести его на экран.

Если на экране возникло сообщение об ошибке, повторите ввод более внимательно: между числами и знаком сложения обязательно должны быть пробелы, а после плюса должна быть нажата клавиша ввода.

Теперь введите

```
.
```

(то есть просто наберите символ "точка" и нажмите клавишу ввода).

Вы увидите

```
357 Ok
```

Очевидно, что это результат сложения ранее введенных чисел. Теперь вы знаете, что для вывода результата вычислений используется **.** ("точка"). Слово "Ok" выводится интерактивной системой программирования, когда она успешно завершает обработку введенной информации и ожидает продолжения диалога с пользователем.

Первое, что следует усвоить при работе с языком программирования Форт, это одно существенное отличие от всех остальных языков программирования: *каждое вводимое слово должно отделяться от соседних пробелом*.

Все, что вы вводите с клавиатуры, в Форте является либо числом, либо *словом языка*. В предыдущих примерах **+** (плюс) и **.** (точка) были *словами языка Форт*.

Система воспринимает текст так же, как человек, читающий книгу - отсутствие пробела между двумя словами может превратить их в новое слово, с собственным значением, меняющим смысл всего предложения.

Ввод текста всегда завершается нажатием клавиши ввода (Enter), только после завершения ввода строки система приступает к ее "чтению" и анализу. В дальнейшем слова "нажмите клавишу ввода" в тексте будут опускаться.

Итак, вы только что сложили два числа и увидели результат. Теперь введите

```
6 7 * .
```

и вы получите результат

```
42 Ok
```

Символ ***** - это слово операции умножения. А теперь введите более длинную строку:

```
36 164 + 2 * .
```

На экране должно получиться

```
400 Ok
```

Давайте проанализируем действия системы. Сначала были введены числа 36 и 164, выполнено их сложение, что дало 200, а затем система получила число 2 и команду умножения. Результат умножения суммы (200) на 2 был выведен на экран.

Проверим другие арифметические действия.

Введите

```
50 5 / .
```

и вы увидите

```
10 Ok
```

Аналогично ввод строки

```
50 5 - .
```

дает в результате

```
45 Ok
```

Если ввести

```
5 50 - .
```

то результат будет выглядеть как

```
-45 Ok
```

из чего следует, что система принимает во внимание порядок, в котором вводятся числа.

Теперь введите

```
5 50 / .
```

и вы получите

```
0 Ok
```

Результат совершенно правильный, поскольку система работает с целыми числами, а 5/50 представляет собой дробь, значение которой округляется до 0.

Вы только что использовали интерактивную среду Форт-системы для выполнения арифметических действий сложения, вычитания, умножения и деления. Для продолжения изучения ее возможностей необходимо рассмотреть то, что пока не было видно на экране.

2.4. Стек параметров

Что происходит с текстом, который вы ввели с клавиатуры? Очевидно, что система каким-то образом выделяет из него числа и слова, являющиеся для нее командами. Детальное изучение этого процесса оставим на потом, а сейчас разберемся с тем, куда попадают введенные числа и откуда берется результат.

Числа из введенной строки помещаются в стек. *Стек* – особым образом организованная область памяти, предназначенная для хранения данных.

Представьте стек как сложенные аккуратной стопкой кирпичи, на гранях которых нанесены числа.

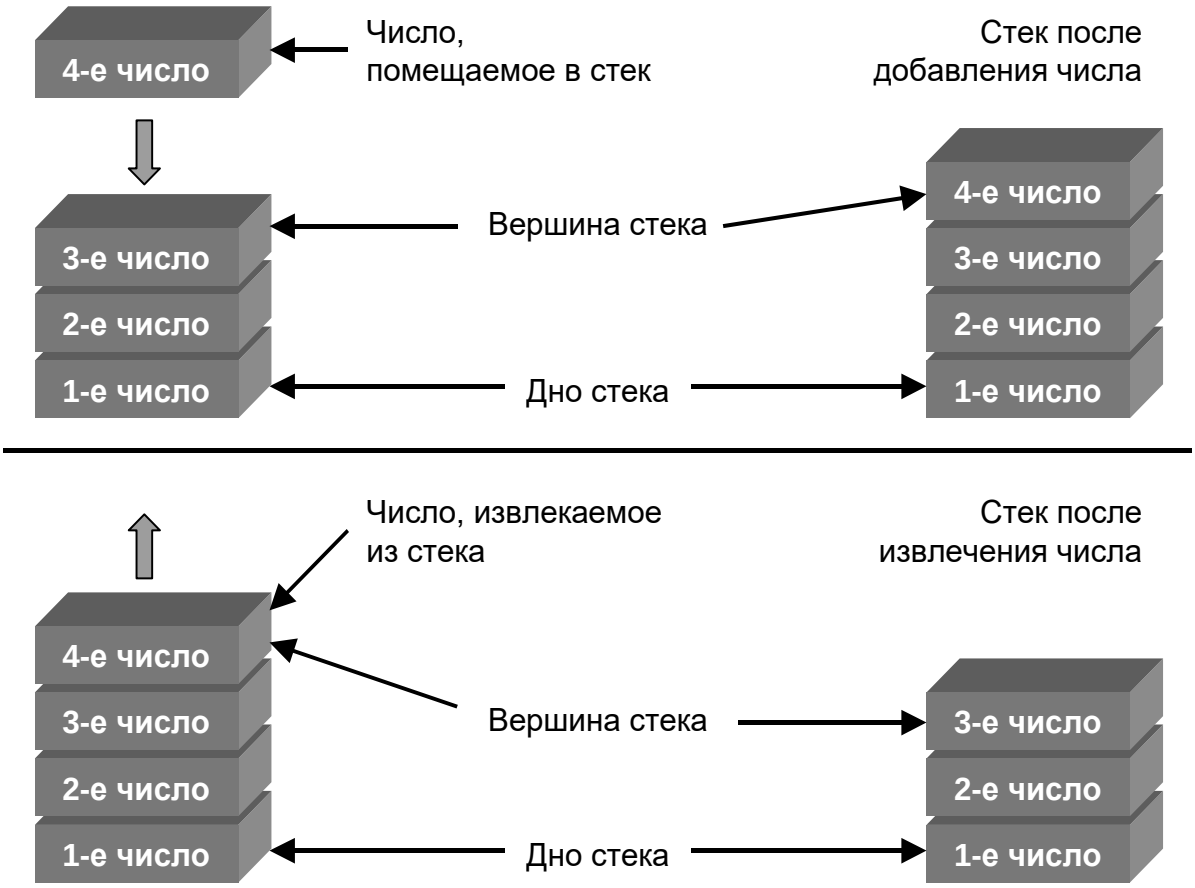


Рисунок 2.3 Стек параметров

Данные, помещаемые в стек, автоматически размещаются аппаратурой в памяти в том порядке, в каком они поступают, а извлекаются в обратном порядке.

Другими словами, доступным для извлечения из стека всегда является последнее введенное в него число. Такой порядок размещения данных известен как «первым пришел – последним вышел».

Последнее введенное в стек число называют *вершиной стека*, а самое первое – *дном стека* (про последнее введенное число также говорят, что оно *лежит на вершине стека*).

Стек Форт-программы, в котором размещаются данные для арифметических и других операций, называется *стеком параметров*. Кроме стека параметров в системе существует и другие стеки специального назначения.

Интенсивное использование стека - это наиболее существенное отличие языка Форт от других языков программирования.

Начнем с заполнения стека числами (Рисунок 2.3). В верхней части рисунка мы изобразили процесс помещения числа в стек.

Когда вы вводите текст, представляющий число, завершая его нажатием клавиши ввода, система помещает это число в стек, аналогично тому, как кирпич кладется на вершину стопки для хранения.

Если до этого в стеке уже были какие-то числа, они остаются на месте. В нижней части рисунка изображено состояние стека после извлечения числа.

Когда в тексте встречается **.** (то есть команда вывода числа), число извлекается из стека так же, как кирпич снимается с вершины стопки для использования (в данном случае это делается, чтобы вывести число). Лежащие ниже числа остаются на своих местах без каких либо изменений.

Заметьте, что извлекая числа из стека, в какой-то момент мы заберем последнее число. Что будет, если после этого попытаться извлечь из стека еще одно число?

Если бы это были кирпичи, лежащие стопкой на земле, мы бы просто не смогли это сделать (нет больше кирпичей). Но стек компьютера – это область памяти, выше и ниже которой есть другие ячейки, занятые какими-то данными.

Попытка выборки из пустого стека приводит к тому, что вершина стека на какое-то время оказывается ниже его дна. Если в этом состоянии в стек поступит новое число, оно окажется записанным в память, не принадлежащую стеку, что скорее всего приведет к повреждению программы.

То же самое происходит при заполнении стека выше предельно допустимого уровня – очередное число записывается не в стек, а в область памяти программы и повреждает ее. Происходит переполнение стека.

Форт-система отслеживает, насколько это возможно, нарушение границ стека. Если вы получили сообщение

```
P-STACK DESTROYING
```

("Разрушение стека параметров"),

то это означает, что вы что-то делаете неправильно. Частое появление этого сообщения несет потенциальную опасность повреждения важных участков памяти, поэтому лучше выключить Форт-машину и начать сеанс программирования заново.

Вернемся к работе со стеком. Когда вы вводите

```
12 345 +
```

числа 12 и 345 кладутся в стек, а операция **+** извлекает их из стека, складывает и полученную сумму снова кладет в стек. Все перемещения чисел-кирпичей при выполнении сложения изображены на следующем рисунке.

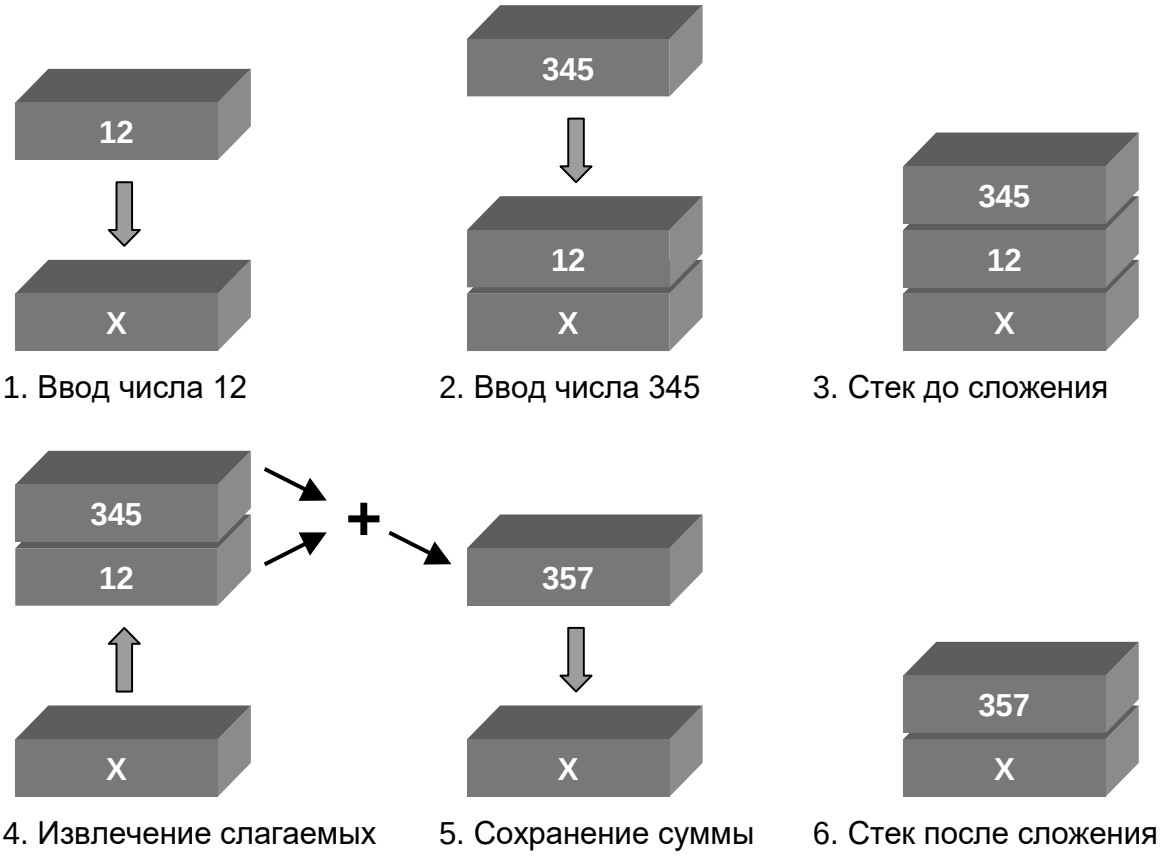


Рисунок 2.4 Сложение чисел с использованием стека

Символом X обозначены значения, которые присутствуют в стеке на момент выполнения операции сложения. Это могут быть ранее введенные числа или результаты выполнения предшествующих слов.

Используемая языком программирования Форт система записи выражений вида "x y +" называется *постфиксной нотацией*, поскольку символ операции (оператор) стоит после чисел (операндов). Постфиксная нотация известна также как *обратная польская нотация*.

Когда ранее вы ввели строку

```
36 164 + 2 * .
```

результат был равен 400. Система извлекла из стека числа 164 и 36, сложила их и поместила сумму на вершину стека. Затем в строке обнаружилась двойка, которая также была помещена в стек. Перед выполнением оператора умножения стек содержит два числа - 200 и 2, что и дает в результате 400.

Выражение с обратной нотацией

```
2 36 164 + *
```

соответствует алгебраическому выражению 2*(36+164) и дает тот же результат, что и приведенный выше пример.

2.5. Первый шаг к программированию

До сих пор вы использовали систему как калькулятор и практически не сталкивались с какими-либо действиями, непосредственно представляющими процесс программирования.

Существует старинная традиция при обучении программированию: первая программа должна вывести на экран фразу "Hello, World!".

Введите следующий текст:

```
: Hello ." Hello, World! " ;
```

Не забудьте вставлять пробелы между словами Форта! Если вы набрали текст без ошибок, на экране появится "Ok" и больше ничего. Как было сказано ранее, это означает нормальное усвоение информации системой и готовность к дальнейшему вводу.

Введите

```
Hello
```

и вы получите

```
Hello, World! Ok
```

Вы только что определили (описали) слово Hello на языке Форт. Строго говоря, это слово можно рассматривать как крошечную программу: ее действие состоит в том, чтобы вывести на терминал текстовую строку.

Мы можем сохранить эту программу в памяти Форт-машины как программу пользователя, это несложно, переключить модуль AFM в автономный режим и получать "Hello, World!" на терминале при каждом включении или нажатии кнопки сброса.

Как вы уже знаете, интерактивная система программирования в буквальном смысле читает то, что вы вводите с клавиатуры. Для того чтобы понять, что именно требуется сделать, система анализирует последовательность слов и выполняет соответствующие действия.

Язык Форт, как и те языки, на которых общаются люди, включает в себя некоторый набор слов, имеющих определенный смысл, и правила, позволяющие соединять эти слова для передачи смысла более сложных понятий.

Когда вы встречаете в тексте слово, значение которого вам неизвестно, вы используете толковый словарь. В словаре данному слову соответствует статья, текст которой передает смысловое содержание нового слова с помощью более простых, более распространенных и потому более понятных слов.

Приведенный пример с толковым словарем в точности соответствует тому, как Форт-система ведет себя при анализе текста. В системе имеется *словарь Форт*. На момент "рождения" в словаре Форт определены слова, значение которых может быть задано только на самом глубоком уровне - на уровне аппаратуры исполнения программ.

Эти слова называются *примитивами языка*, или *атомными словами Форта*, к ним относятся **+**, **-**, *****, **/**, **:**, **;** и многие другие.

Общение с пользователем позволяет пополнить словарный запас системы путем определения новых слов и внесения их в словарь.

По своей сути интерактивное программирование Форт-системы сводится к последовательному изложению более сложных понятий с использованием ранее определенных слов. В этом и есть фундаментальное отличие языка Форт от всех остальных.

2.5.1. Создание новых слов. Определение через двоеточие

Разберемся с последними введенными в систему данными.

Строка

```
: Hello ." Hello, World! " ;
```

представляет собой определение нового слова для словаря Форт. Как система читает строку и что при этом происходит? Первое слово **:** (двоеточие) указывает системе на начало определения нового слова (как заголовок статьи толкового словаря). Само слово следует за двоеточием, в данном случае это **Hello**.

Слово, следующее за двоеточием, называется *именем определения* или *именем статьи*. Далее следует то, что в толковом словаре является текстом статьи, а в словаре Форт - описанием действия нового слова.

Слово **."** ("точка-кавычка") - это атомное слово, уже существующее в словаре. Его задача — выделить из потока ввода строку текста, завершенную символом "кавычка", сохранить строку в памяти и вывести на терминал, когда слово **Hello** будет выполняться.

Слово **."** выводит на экран текст, начинающийся сразу за отделяющим его пробелом и заканчивающийся кавычкой. В нашем определении между восклицательным знаком и завершающей текст кавычкой имеется пробел, хотя можно обойтись и без него. Пробел здесь "для красоты", так как кавычка не является словом языка, а лишь используется словом **."** для нахождения конца строки. После кавычки пробел как всегда необходим для выделения слова **;** .

Последний символ **;** (точка с запятой) указывает на то, что определение закончено и новое слово может быть использовано.

Еще раз обратим внимание на то, что в качестве слова в языке Форт может выступать даже один символ, а все слова должны обязательно отделяться друг от друга пробелами. То, что мы привыкли считать знаками препинания, например, двоеточие, точка, запятая и другие символы, с точки зрения Форта может быть словом, имеющим вполне определенный смысл.

Слова языка Форт можно использовать двояко: либо как команды, которые должны быть немедленно выполнены, примерами этого были наши арифметические упражнения, либо для описания новых слов, добавляемых в словарь.

Процесс анализа системой текстовой строки и немедленного выполнения встреченных слов называется *интерпретацией*. Добавление в словарь определения нового слова языка называется *компиляцией*.

Программа на языке Форт создается путем составления определений новых слов, для которых используются ранее определенные слова, пока не будет определено главное слово, то есть то слово, которое нужно ввести, чтобы начать выполнение всей программы.

Большинство новых слов языка Форт создается в виде так называемых *определений через двоеточие*, потому что их описание начинается с **:** и заканчивается **;** (точкой с запятой).

Хотя слово Hello в буквальном смысле представляет собой настоящую программу, она слишком проста и не позволяет увидеть все преимущества предлагаемого метода программирования.

2.6. Словарь пользователя и сеанс программирования

Когда вы включаете Форт-машину в интерактивном режиме, запускается сеанс программирования AFS. В памяти выделяется место для вашей программы и новых слов.

Слова в системе хранятся в двух словарях. Первый – это системный словарь Форт, второй – словарь пользователя, куда попадают все определенные вами слова.

2.6.1. Слово WORDS

Время от времени возникает необходимость посмотреть, какие слова уже известны системе. Для вывода списка всех слов, определенных в Форт-системе на текущий момент, используется слово **WORDS** .

Слово **WORDS** сначала показывает все слова, созданные пользователем, причем от последнего слова к первому, затем все слова системного словаря по алфавиту.

Введите **WORDS** и вы увидите словарь пользователя:

```
USER DICTIONARY

HELLO                                |----

PRESS 'ENTER' TO CONTINUE
```

Мы видим слово **HELLO**, которое мы только что создали, и его атрибуты (четыре прочерка справа), а также предложение нажать клавишу Enter для продолжения. С атрибутами слов мы разберемся позже.

Система ожидает нажатия клавиши Enter для продолжения вывода. Когда вы это сделаете, начнется вывод системного словаря:

```
SYSTEM DICTIONARY

!                                    |S---
#                                    |S---
#>                                   |S---
#S                                   |S---
'                                    |S---
*                                    |S---
```

Каждому слову словаря соответствует своя строка, справа выводятся атрибуты слова. Обратите внимание, появился атрибут S, что означает принадлежность слова к системному словарю.

Словарь выводится на экран страницами по 20 строк (мы не стали здесь перепечатывать весь экран), в конце выводится предложение нажать Enter для продолжения вывода.

Если вы уже нашли интересующее слово, можно прекратить вывод словаря, нажав вместо Enter комбинацию клавиш Ctrl+Z. Система вернется в интерактивный режим и напишет "Ok".

Если выводится словарь пользователя, занимающий более одной страницы, комбинация клавиш Ctrl+Z прекратит его вывод и начнется вывод системного словаря.

Слова системного словаря выводятся в алфавитном порядке за одним исключением: все слова управления устройствами ввода-вывода сгруппированы в конце словаря.

Заметим, что все слова выводятся заглавными буквами, независимо от того, как вы написали слово при создании. Для языка Форт регистр букв в словах не имеет значения, при записи в словарь и поиске слов все символы приводятся к верхнему регистру.

Таким образом, слова **`Hello`**, **`HELLO`** и **`hello`** – для Форт-системы одно и то же слово.

Будьте внимательны: кроме слова **`WORDS`** *существует слово* **`WORD`** *, одна пропущенная буква может привести к неожиданному результату!*

2.6.2. Слово SMUDGE

Бывает, что уже после определения слова вы понимаете, что нужно было сделать все иначе, или просто видите опечатку в тексте. Причин избавиться от неудачного слова может быть много.

Чтобы удалить слово из словаря, используйте слово **`SMUDGE`** . Например, если вы хотите избавиться от слова **`Hello`** , введите

```
SMUDGE Hello
```

и вы больше никогда не увидите его в словаре. Это легко проверить с помощью слова **`WORDS`** . Теперь вы можете определить новое слово **`Hello`** .

Слово **`SMUDGE`** удаляет только слова пользователя. Если слово системное (в атрибутах присутствует S), попытка стереть его вызовет ошибку и вы увидите сообщение

```
CAN'T ERASE SYSTEM WORD: . . .
```

и слово, которое пытались удалить.

2.6.3. Слово SAVE-SESSION

Программы в Форте разрабатываются последовательным добавлением новых слов в словарь.

Вы можете тестировать различные варианты каждого следующего слова, удаляя неудачные и продвигаясь к цели методом проб и ошибок.

Чтобы результаты вашего труда не пропали даром, AFS предоставляет возможность сохранения проделанной работы во время сеанса программирования.

Введите

```
SAVE-SESSION
```

и система сохранит в постоянной памяти словарь пользователя, метакод созданных слов, переменные программы и другую информацию.

Если после этого вы выполните слово, приводящее к краху системы, после перезапуска AFS предложит вам восстановить сохраненный сеанс программирования (см. далее).

Точка сохранения остается в постоянной памяти Форт-машины до тех пор, пока вы не добавите хотя бы одно новое слово. Система видит изменение состояния словаря и удаляет предыдущую сохраненную информацию. Если в этот момент вы отключите питание, все данные будут потеряны и следующий сеанс начнется "с чистого листа".

2.6.4. Слово BYE

Слово BYE завершает текущий сеанс программирования с сохранением его результатов.

Поскольку вы попрощались с Форт-системой, если сохранение сеанса прошло успешно, AFS выходит в меню запуска. Теперь можно отключить питание Форт-машины.

При следующем включении вы снова увидите меню запуска:

```
MEMORY OCCUPIED BY AFS SESSION
1= ERASE PROGRAM MEMORY
2= CONTINUE AFS SESSION
```

Вам предлагается на выбор два варианта – очистить память и начать новый сеанс программирования, либо вернуться к сохраненному сеансу и продолжить работу с того места, где вы остановились накануне.

Для выбора введите 1 или 2 соответственно.

Сеанс программирования сохраняется только если вы закончили работу словом **BYE** или выполнили слово **SAVE-SESSION** перед выключением. Если вы просто выключили питание, часть данных в памяти машины безвозвратно теряется. При старте AFS обнаруживает такое нарушение целостности программы и очищает всю память Форт-машины, чтобы начать работу с исходного состояния.

2.7. Более сложный пример программирования

Введите текст

```
: square DUP * ;
```

Это определение нового слова **square** (квадрат). В определении появилось новое слово **DUP**, которое делает дубликат числа, лежащего на вершине стека.

Если вернуться к аналогии с кирпичами, при выполнении слова **DUP** кирпич с вершины стека "клонируется" и его брат-близнец кладется поверх него. Таким образом, в стеке становится одним числом больше, причем на вершине оказывается пара идентичных чисел. Действие слова **DUP** наглядно представлено на рисунке.

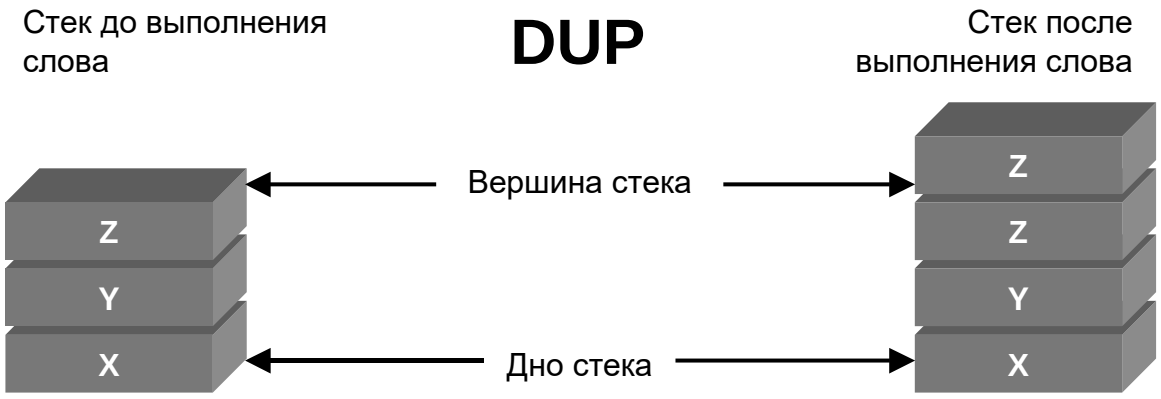


Рисунок 2.5 Действие слова DUP

Введите

```
8 square .
```

и вы получите

```
64 Ok
```

Очевидно, слово **.** (точка) выводит результат, а вот откуда он берется ?

Посмотрите на рисунок.

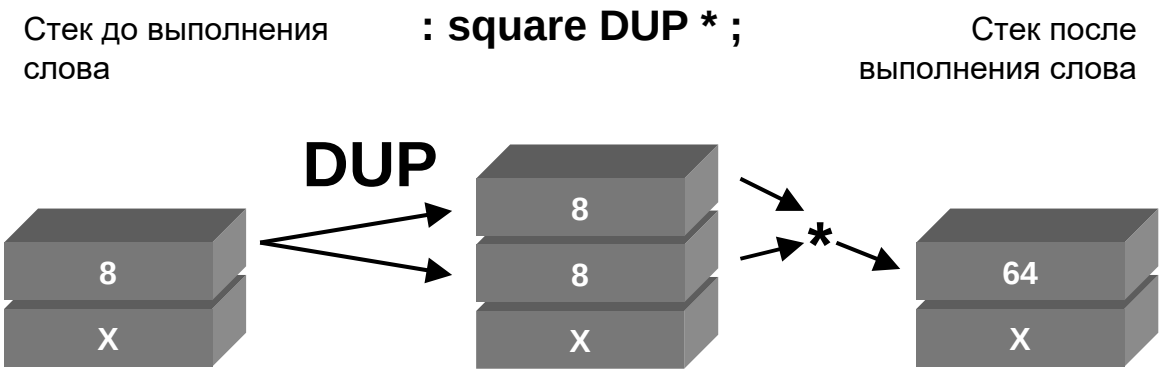


Рисунок 2.6 Выполнение слова square

Слово **square** делает копию числа 8 в стеке, оставляя 8 и 8, а затем перемножает эти два одинаковых числа, получая 64.

Теперь введите

```
: cube DUP square * ;
```

и вы получите слово для вычисления куба числа, использующее только что определенное слово, вычисляющее квадрат. Как оно работает? Очень просто. Помещенное в стек число

дублируется словом **DUP** из определения в слове **cube**. Копия числа на вершине стека возводится в квадрат словом **square** и в стеке оказывается два числа - исходное значение и его квадрат. Слово ***** из определения в слове **cube** перемножает их и оставляет на вершине стека результат, который и является кубом исходного числа. Убедиться в том, что все происходит именно так, вы можете, введя

```
3 cube .
```

и получив результат

```
27 Ok
```

2.8. Простая программа

В завершении первого знакомства с программированием на языке Форт напишем небольшую программу, содержащую некоторые ключевые элементы языка, требующую определения более чем двух слов и дающую более наглядный результат, чем простое возведение числа в куб.

Создадим программу, выводящую гистограмму из символов "звездочка".

Построение рисунков из символов - хорошая практическая задача для начинающих программистов. Для ее решения не требуется углубленного знания аппаратуры, что позволяет сосредоточиться на тонкостях изучаемого языка программирования.

Для начала напишем слово, которое будет выводить на экран строку "звездочек" заданной длины, но для этого надо знать, как вывести один символ. Вы уже знаете, что числа выводятся с помощью слова **.** (точка), но сейчас вам потребуется выводить не число, а текст.

В языке Форт предусмотрено несколько слов для вывода текста. Воспользуемся уже известным нам словом **."** (точка и кавычка, без пробела между ними) и с его помощью определим слово **star**, которое будет выводить на экран одну "звездочку":

```
: star ." *" ;
```

В данном случае между звездочкой и последней кавычкой пробел не нужен, иначе он тоже будет выведен на экран. После кавычки пробел как всегда необходим для отделения слова **;** .

Проверьте новое слово. Введите

```
star
```

и на экране должно появиться

```
*Ok
```

Теперь можно использовать слово **star** для изображения линии из "звездочек". Для этого нам потребуется повторить выполнение слова star некоторое количество раз.

Многократное выполнение набора инструкций, или *зацикливание программы*, очень важная возможность любого языка программирования.

В языке Форт предусмотрено несколько способов организации такого рода действий. Подробно они будут рассмотрены позже, а пока мы в двух словах обсудим принцип организации *цикла* в программе (см. Рисунок 2.7).

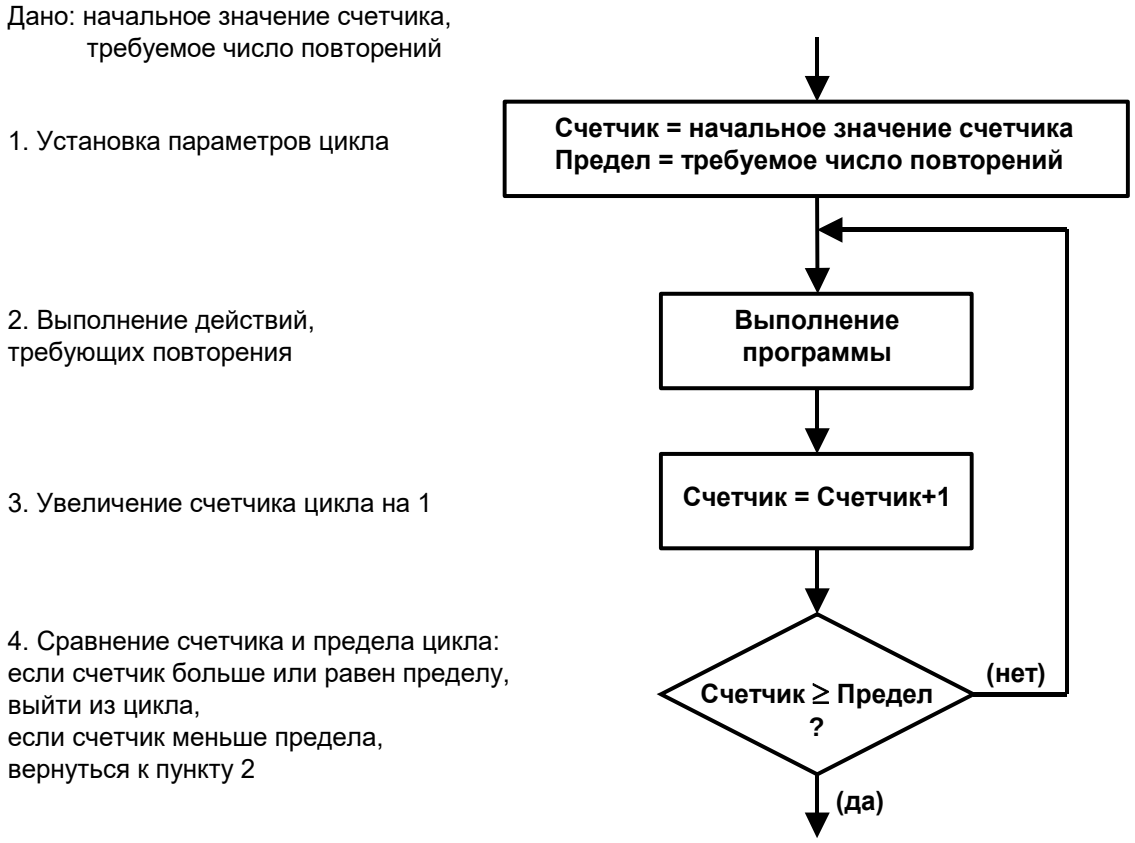


Рисунок 2.7 Организация цикла в программе

Очевидно, что для выполнения какого-либо действия некоторое количество раз требуется *счетчик* повторений. Каждый раз, когда повторяемое действие выполняется, счетчик должен увеличиваться на 1. Для того, чтобы понять, когда необходимо прекратить повторения и заняться чем-то еще, программа должна помнить требуемое число повторов. Это число называется *пределом* цикла. После увеличения счетчика повторений программа сравнивает его с пределом цикла. Если счетчик стал равен пределу или превысил его, выполнение цикла прекращается и программа переходит к выполнению следующих действий.

Если вы уже изучили Рисунок 2.7, то наверняка заметили, что там присутствует еще один параметр – *начальное значение счетчика* цикла. Для правильного счета необходимо установить счетчик в известное значение, обычно в ноль, но часто бывает полезным начинать счет с другого числа.

Итак, вы знаете о принципе организации цикла. Настало время применить эти знания в вашей программе.

Введите следующий текст :

```
: 10stars 10 0 DO star LOOP ;
```

Слово **10stars** - это имя определяемого слова, поэтому после 10 пробел не требуется.

Введите

```
10stars
```

и на экране появится

```
**********Ok
```

То есть 10 "звездочек", после которых выведено сообщение о готовности системы. Новые слова **DO** и **LOOP**, которые появились в определении **10stars**, используются для повторного выполнения слов, находящихся между ними, другими словами, они организуют цикл. Давайте поищем известные нам параметры цикла.

Мы изобразили схему выполнения слов из определения **10stars** на рисунке.

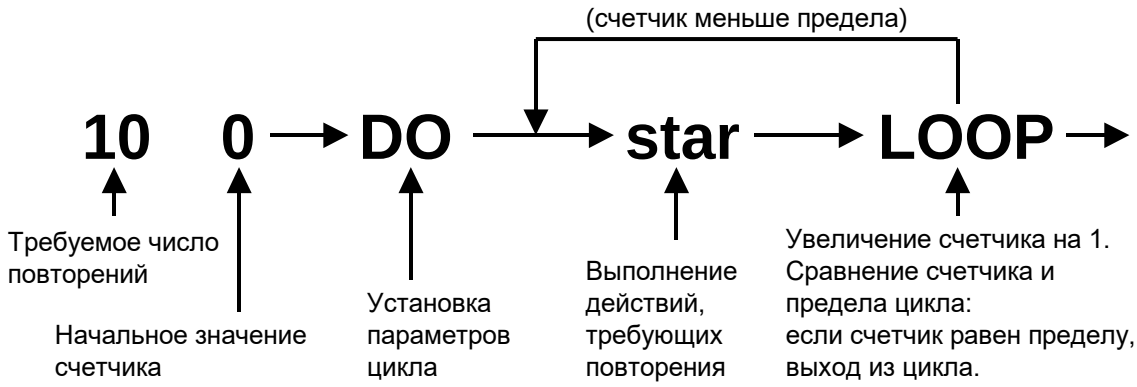


Рисунок 2.8 Цикл в определении слова

Перед словом **DO** в стек помещаются два числа. Слово **DO** извлекает число с вершины стека, в нашем случае это 0, и использует его для установки начального значения счетчика цикла. Счетчик цикла хранится внутри системы, где именно – сейчас не важно, позже мы подробно все объясним.

Затем слово **DO** извлекает из стека следующее число, в нашем определении это 10. Как вы уже догадались, это требуемое число повторений или предел цикла. Предел также сохраняется внутри системы. Таким образом слово **DO** используется для инициализации параметров цикла, необходимых слову **LOOP**.

После того, как слово **DO** выполнило свою работу, выполняются слова между **DO** и **LOOP**. В нашем определении это единственное слово **star**, но на самом деле их может быть сколько угодно. Когда приходит очередь слова **LOOP**, оно увеличивает счетчик цикла и сравнивает его новое значение с пределом цикла. Если счетчик меньше предела, программа вернется к слову, следующему за **DO**. Если счетчик стал равен пределу или превысил его, выполнение перейдет к слову, стоящему после **LOOP**.

Слово **10stars**, на самом деле, определено только для знакомства с новыми словами для организации цикла. Слово выводит строку из 10 символов "*****", а нам необходимо

выводить линии различной длины, в зависимости от числа, находящегося на вершине стека. Другими словами, нам нужно изменить определение таким образом, чтобы предел цикла задавался не внутри определения, а мы сами помещали его в стек перед выполнением слова.

Определим еще одно слово:

```
: stars 0 DO star LOOP NL ;
```

Теперь в определении присутствует только начальное значение счетчика цикла (0), а вот предел цикла вам необходимо помещать в стек перед каждым выполнением слова **stars** ("звездочки").

Если вы введете

```
20 stars
```

то увидите

```
********************
Ok
```

Обратите внимание, что "Ok" теперь выводится на следующей строке. Это результат действия слова **NL** из определения **stars**, которое производит перевод строки и возврат экранного курсора в начало строки. Без слова **NL** все строчки "звездочек" будут выстроены в ряд, а не одна под другой.

Замечание для программистов, знакомых с классическим Фортом. Слово NL ("new line") словаря Advanced Forth соответствует слову CR стандартного Форта. В Advanced Forth слово CR имеет другое значение (см. стр. 81).

Вы можете поэкспериментировать, вводя различные числа и слово **stars**. Наверняка вы обратите внимание на тот факт, что при задании числа, большего 80 вывод "звездочек" продолжается на следующих строках. Это нормальная реакция системы на вывод строк, длина которых превышает ширину экрана текстового терминала.

На экране стандартного текстового терминала обычно помещается 25 строк по 80 символов в строке. После вывода каждого символа курсор сдвигается на одно знакоместо вправо. Достигнув крайней правой позиции, курсор переносится в начало следующей строки, а если строка была последней, кроме того происходит сдвиг изображения на одну строку вверх.

Кстати, по этой причине вывод ровно 80 "звездочек" приводит к пропуску одной строки - ведь после вывода восьмидесятого символа "*****" курсор оказывается в начале следующей строки, а в определении **stars** вывод завершается выполнением слова **NL**, которое переводит строку еще раз.

Как можно ограничить длину строки звездочек? Очевидно, перед вызовом **stars** необходимо проверить число в стеке и заменить его, если оно больше 79.

На самом деле надо произвести два независимых действия: *сравнение* чисел и выполнение по результату сравнения, то есть *ветвление* программы. Сначала рассмотрим сравнение.

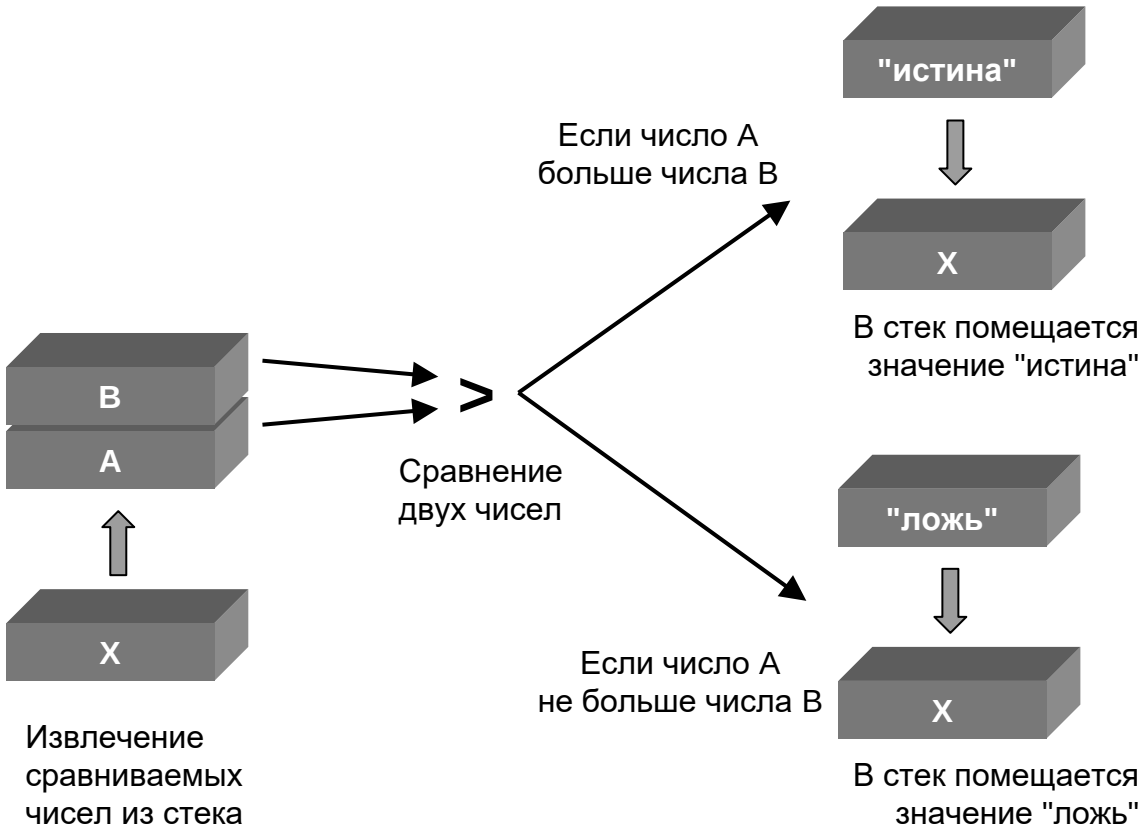


Рисунок 2.9 Сравнение чисел

Как мы выяснили, необходимо узнать, больше число в стеке, чем 79 или нет. Для такого сравнения используем слово **>** (математический знак "больше"). Слово **>** является одним из *операторов сравнения*.

Язык Форт предоставляет большой выбор таких операторов, позже мы более внимательно изучим их.

При выполнении слова **>** из стека удаляются два числа и помещается одно (результат). Результат сравнения представляется не числом, а специальными значениями, соответствующими логическим понятиям "истина" или "ложь". На рисунке вы можете увидеть работу оператора сравнения **>**.

Значения, соответствующие понятиям "истина" и "ложь" называются *логическими флагами* или просто *флагами*. Часто их так и называют – *флаг "истина"* и *флаг "ложь"*.

Если операторы сравнения оставляют в стеке флаг, значит этот флаг должно использовать какое-то слово языка. Логические флаги используются для организации ветвления программы. В данном случае будет применено слово **`IF`**, которое используется вместе со словом **`THEN`**.

Посмотрите на рисунок.

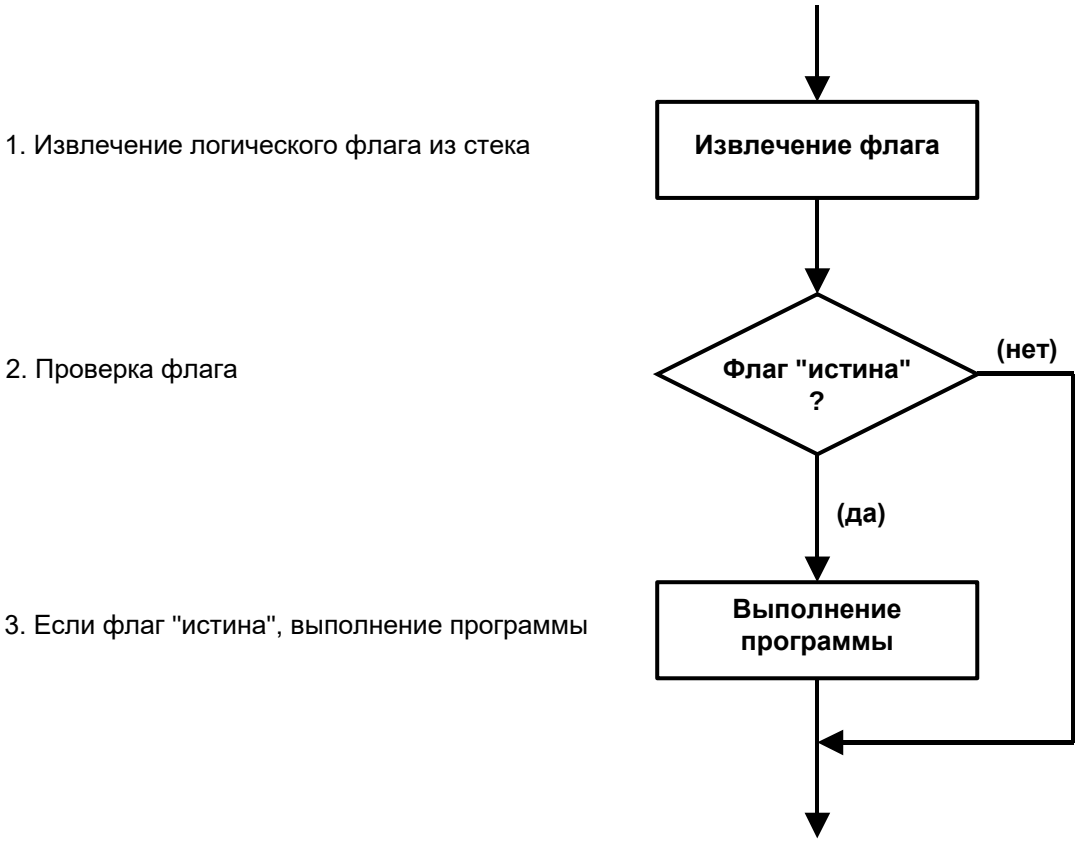


Рисунок 2.10 Организация ветвления программы

Если слово **IF** извлекает из стека значение “истина", то выполняются те слова определения, которые находятся между словами **IF** и **THEN**. Если слово **IF** обнаруживает в стеке значение "ложь", выполнение продолжается со слова, следующего за **THEN**. Таким образом, слово **THEN** ровным счетом ничего не делает – с его помощью слово **IF** определяет точку перехода.

Слова **IF** и **THEN** относятся к группе слов, называемых *операторами ветвления*. Есть и другие способы разветвления программы, но данная конструкция является основной.

Ветвление - это важное свойство любого языка программирования, поскольку оно обеспечивает гибкость и удобство программирования задач любой сложности. Операторам ветвления будет посвящен отдельный раздел.

Теперь, когда вам известны слова для сравнения и ветвления программы, осталось объединить их и использовать для ограничения длины выводимой строки "звездочек".

В определении слова, которое делает это, будет использовано слово **DROP**, которое вам еще не встречалось. Это слово просто удаляет число, находящееся на вершине стека, как показано на следующем рисунке.

Рисунок 2.11 Действие слова DROP

Теперь вы можете ввести определение слова

```
: limit79 DUP 79 > IF DROP 79 THEN ;
```

Рассмотрим, что делают слова, которые впервые появились в этом примере. На рисунке мы изобразили схему выполнения слова **limit79**.

Рисунок 2.12 Ветвление в определении слова

Перед вызовом в стек помещается некоторое число. Слово **DUP** вам уже знакомо, оно создает копию числа на вершине стека. Как вы помните, операторы сравнения извлекают из стека два числа, именно поэтому мы сначала сделали копию проверяемого значения с помощью **DUP**.

Если число в стеке меньше или равно 79, то выражение **79 >** помещает в стек флаг "ложь", которое слово **IF** воспринимает как указание перейти на слово, следующее после **THEN**, на этом выполнение слова **limit79** заканчивается.

Если число больше 79, то в стек помещается флаг "истина" и исполняются слова между **IF** и **THEN**, оставшееся в стеке исходное значение удаляется с помощью слова **DROP** и вместо него помещается число 79.

Таким образом, используя слово **limit79** перед вызовом слова **stars** вы получаете строку "звездочек" заданной длины, либо строку, занимающую всю ширину экрана:

```
15 limit79 stars
```

выведет 15 "звездочек", а

```
150 limit79 stars
```

выведет 79 "звездочек".

Теперь нам нужно напечатать несколько строк "звездочек" различной длины. Самое, простое, что можно для этого сделать, это поместить в стек значения для вывода строк и организовать еще один цикл с помощью слов **DO** и **LOOP**. Вот определение слова, которое строит гистограмму из 5 строк:

```
: test-hist 5 0 DO limit79 stars LOOP ;
```

Если ввести следующий текст

```
5 10 15 20 25 test-hist
```

вы увидите гистограмму

```
*************************
*******************
***************
**********
*****
```

Существенным недостатком слова **test-hist** является то, что работать оно будет только при наличии в стеке ровно 5 чисел. Решить эту проблему можно, вынеся из определения значение предела цикла, как мы сделали это раньше, определяя слово **stars**. Тогда на вершине стека должно быть количество чисел, помещенных в стек.

Это не очень хороший вариант решения задачи, так как в нем нет гибкости и он не защищен от ошибок. К счастью, в языке Форт есть слово **DEPTH** (глубина), которое помещает на вершину стека количество чисел в стеке. Например, если ввести

```
1 2 3 4 5 DEPTH .
```

вы увидите

```
5 Ok
```

на экране, а сами числа по-прежнему находятся в стеке.

Теперь вы можете определить слово **hist**, которое будет строить гистограмму чисел, находящихся в стеке, независимо от их количества:

```
: hist DEPTH 0 DO limit79 stars LOOP ;
```

Итак, введите

```
8 10 12 14 10 8 hist
```

Получаем гистограмму:

```
********
**********
**************
************
**********
********
```

На этом можете считать ваше первое знакомство с языком Форт состоявшимся.

Чтобы окончательно завершить написание самой первой программы и облегчить понимание материала последующих разделов, добавим важные штрихи к тексту составленной вами программы и придадим ей законченный вид.

2.9. Комментарии в тексте программы

К тексту программы желательно иметь *комментарии*. Некоторые программы становятся полной загадкой без пояснений. Если в исходный текст необходимо внести изменения по прошествии некоторого времени, без комментариев это может быть затруднено.

Комментарии в Advanced Forth отличаются от канонических комментариев Форта. Подробные разъяснения приведены в конце раздела.

В языке Advanced Forth используются два вида комментариев.

2.9.1. Слово \\

Первый вид комментария – строка, начинающаяся словом **** . Если вы введете

```
\\ Это строка комментария
```

то система просто ответит “Ok” и ничего не произойдет. Слово **** сообщает системе о необходимости игнорировать весь текст до конца строки. Комментарии, занимающие всю строку, обычно используются в тексте программ как заголовки и пояснения к словам:

```
\\ Слово, вычисляющее квадрат числа на вершине стека
: square DUP * ;
```

2.9.2. Слово \

Второй вид комментария – текст, заключенный между двумя символами **** . Слово **** используются для ввода в текст программы комментариев такого вида:

```
\ Это текст комментария\
```

Обратите внимание на пробелы: после первой косой черты пробел должен быть обязательно, так как **** является словом языка AF, а вот в конце комментария пробел не обязателен, второй символ **** используется только в качестве признака завершения текста комментария (аналогично кавычке в тексте для слова **."**).

Если вы введете этот текст, система отреагирует на него так же, как и в предыдущем случае, за одним исключением: текст, следующий за комментарием до конца строки будет интерпретироваться системой как обычно.

Такой вид комментария удобно использовать внутри определения слова:

```
: square \ (n1 -- n2)\ DUP * ;
```

Комментарий, который вы видите в этом примере, называется *диаграммой состояния стека* или *стековой диаграммой*. Такие диаграммы будут широко использоваться в тексте следующих разделов, в описании слов Форта, поэтому разберемся с ними подробнее.

Диаграмма состояния стека заключена в круглые скобки и состоит из двух частей: слева от символов -- находится текст, представляющий состояние стека, необходимое для выполнения слова, справа – текст, представляющий состояние стека после выполнения слова. Иными словами, в диаграмме состояния стека приводится условное обозначение входных параметров и результата.

Круглые скобки и параметры внутри не являются словами AF, поэтому диаграммы состояния стека в тексте программы должны оформляться как комментарии.

Прочитаем диаграмму слева направо. Условное обозначение n1 соответствует одному целому числу со знаком. Это число берется с вершины стека при выполнении слова square. Условное обозначение n2 также соответствует целому числу со знаком, которое является результатом, который помещается на вершину стека словом square. Индексы 1 и 2 показывают, что числа не равны.

Значения, находящиеся в стеке до выполнения слова, называются *входными или принимаемыми параметрами* слова. Значения, остающиеся в стеке после выполнения слова, называются *выходными или возвращаемыми параметрами* слова. Соответственно, описывая действие слова, говорят что оно *принимает* какие-либо значения и *возвращает* или *оставляет* в стеке результат. Эти термины будут часто встречаться в описании различных слов языка, поэтому их стоит запомнить.

Стековая диаграмма может отражать увеличение или уменьшение количества чисел в стеке после выполнения слова.

Например, диаграмма состояния стека в описании DUP (x -- x x) показывает, что в результате выполнения слова DUP в стеке становится на одно число больше, а на вершине находятся два одинаковых числа, равных исходному. Условное обозначение x соответствует любому числу.

Мы не поместили диаграмму в комментарий, потому что это не текст программы. В текстах стандартов описания слов Форта приводятся именно в таком виде.

Такие описания, которые не являются определениями слов и не требуют ввода в систему, мы будем называть *заголовками слов*. Заголовки используются в документации и комментариях программ для лучшего понимания того, какие данные требуются слову

перед выполнением и какие результаты оно возвращает. Также они используются в высокоуровневых версиях AFS как заголовки статей в справочной системе словаря Форт.

Такое наглядное представление очень полезно, и мы рекомендуем вам сразу привыкнуть к записи стековых диаграмм в определениях ваших слов.

Мы настоятельно рекомендуем использовать круглые скобки только для стековых диаграмм в определениях слов, заключая их в комментарии. В самих диаграммах пользоваться условными обозначениями, принятыми стандартом языка Форт.

Соблюдение данной рекомендации позволит вам использовать возможности AFS более высокого уровня при обработке исходных текстов ваших старых программ.

Описание DROP (x --) показывает, что слово **DROP** удаляет одно число с вершины стека, и если кроме него в стеке больше ничего не было, стек становится пустым.

Последнее замечание о стековых диаграммах: если в правой или левой части диаграммы имеется больше одного числа, *крайнее левое значение соответствует дну стека, а крайнее правое – вершине*.

Например, в диаграмме состояния стека слова Test (n1 n2 n3 n4 -- n5 n6 n7) число n1 помещено в стек первым и находится на дне, а число n4 помещено в стек последним и находится на вершине.

После выполнения слова **Test** числа n1-n4 удаляются из стека и вместо них в стеке оказываются числа n5, n6 и n7, причем число n5 помещается в стек первым, а число n7 – последним. Соответственно, после выполнения слова **Test** на вершине стека оказывается n7.

В диаграммах состояния стека используются различные условные обозначения, соответствующие значениям, находящимся в стеке. Эти обозначения будут вводится по мере изучения языка Форт в следующих разделах.

Теперь мы можем представить, как должен выглядеть текст составленной вами программы (вводить в систему его не надо):

```
\\ Программа построения гистограммы из символов "*"
\\ Слово для вывода одной "звездочки"
: star \ ( -- )\ ." *" ;
\\ Слово для вывода строки "звездочек" заданной длины
: stars \ (n -- )\ 0 DO star LOOP NL ;
\\ Слово для ограничения длины строки шириной экрана
: limit79 \ (n -- n|79)\ DUP 79 > IF DROP 79 THEN ;
\\ Главное слово – для запуска программы ввести в стек
\\ числа для отображения и выполнить hist
: hist \ (.. n -- )\ DEPTH 0 DO limit79 stars LOOP ;
```

Обратите внимание на комментарий к слову **limit79** : обозначение n|79 указывает на то, что в результате выполнения слова на вершине стека либо остается исходное число, либо возвращается число 79. В комментарии к слову **hist** обозначение .. n указывает на необходимость наличия в стеке произвольного количества чисел (но не менее одного).

У вас наверняка возник вопрос: для чего нужны комментарии, если текст программы нигде не сохраняется? Это не так. На самом деле текст программы можно составить в любом текстовом редакторе и затем предоставить его в виде обычного файла для компиляции Форт-системой.

Сейчас мы используем самую простую систему, "уровень интеллекта" которой не позволяет записывать файлы в Форт-машину, но вы можете передавать текстовый файл целиком или отдельными фрагментами через терминал.

В более интеллектуальных системах, вводимый вами текст автоматически запоминается в текстовом файле, который вы сможете отредактировать и использовать снова, а составленная вами программа может быть сохранена в виде исполняемого файла AFS.

В нашей учебной AFS уровня L0 комментарии между двух символов \ так и остаются просто комментариями, но в AFS более высокого уровня они могут помещаться в словарь Форт вместе с определением слова. Когда вам нужно вспомнить, что делает какое-либо слово, вы используете специальный сервис, который показывает вам эти комментарии, сохраненные в словаре.

Все эти возможности доступны в более мощных системах, но принимать их во внимание надо уже сейчас и оформлять программы соответствующим образом.

Для реализации новых возможностей в Advanced Forth пришлось отказаться от совместимости с классическим Фортом. В частности, комментарии в стандартном Форте выглядят иначе.

Комментарий до конца строки начинается словом **** :

```
\ Это комментарий в стандартном Форте
```

Комментарий внутри текста определения заключается в круглые скобки:

```
: square ( Это комментарий в стандартном Форте ) dup * ;
```

Помните об этом, читая литературу по языку Форт.

2.10. Загрузка исходного текста программы из файла

Создавая программу на Форте вы можете проверять ее слово за словом - нет необходимости компилировать весь исходный текст единовременно, как это происходит в большинстве других языков программирования.

Разумеется, исходный текст программы на Форте тоже где-то должен храниться.

Вы можете использовать любой текстовый редактор, например, "Блокнот" для редактирования файла, содержащего исходный текст программы.

Практически все программы терминала позволяют передавать текстовый файл полностью или копировать его части для передачи.

С помощью имеющегося у вас текстового редактора создайте файл HIST.AF и скопируйте в него текст последней программы (с предыдущей страницы). Вы можете задать любое другое имя файла, но для исходных текстов Advanced Forth рекомендуем расширение .AF .

Если вы используете терминал Tera Term, для передачи текстового файла выберите пункт меню "Файл / Передать файл…" (см. Рисунок 2.13).

В открывшемся окне выберите файл HIST.AF и нажмите кнопку "Открыть". Программа терминала начнет передачу файла в AFS.

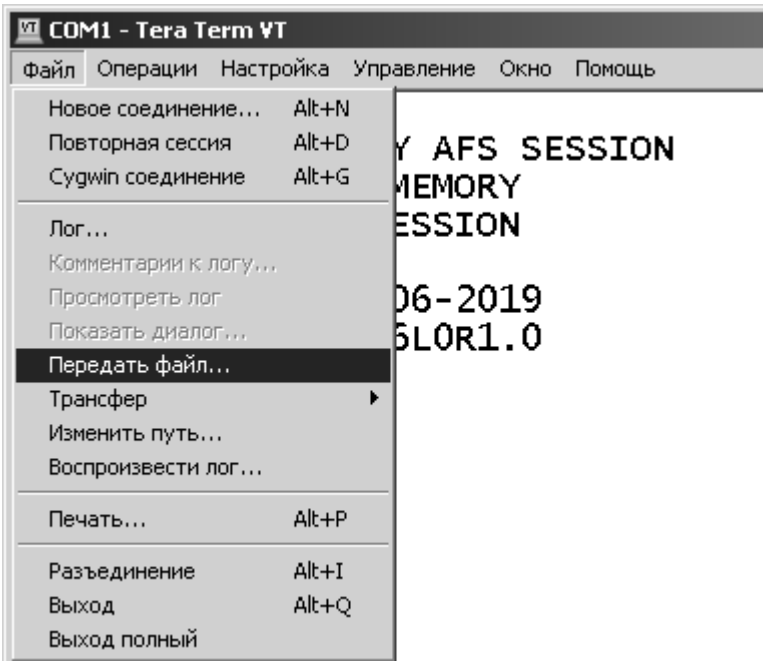


Рисунок 2.13 Передача тестового файла в Tera Term

Поскольку интерактивная среда отвечает "Ok" на каждую введенную без ошибок строку, вы увидите столько сообщений "Ok", сколько строк было в текстовом файле. В некоторых случаях эти сообщения могут быть показаны терминалом после текста переданного файла.

Сообщения об ошибках поступают от AFS сразу после обнаружения, но текстовый файл при этом продолжает передаваться, что может привести к возникновению новых ошибок. По этой причине имеет смысл загружать только файлы уже проверенных исходных текстов.

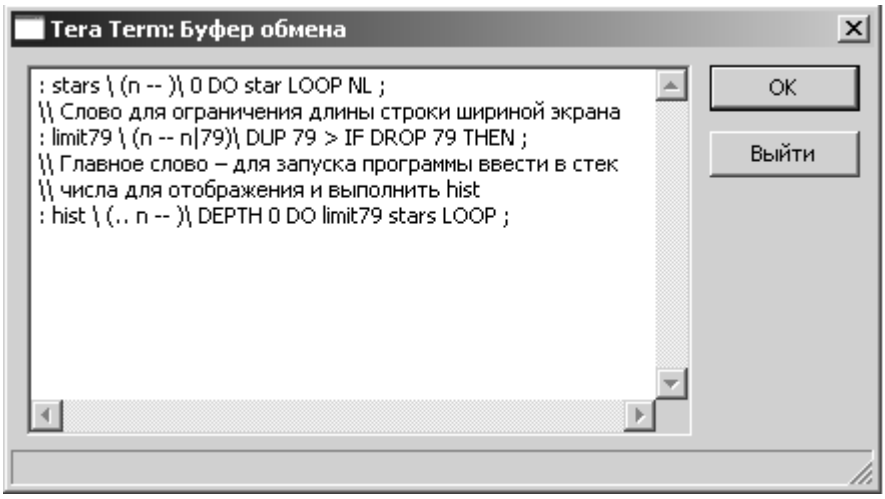


Рисунок 2.14 Редактирование буфера обмена в Tera Term

Для отладки отдельных слов и загрузки частей исходного текста программы воспользуйтесь копированием текста: выделите фрагмент в открытом файле и скопируйте его через Ctrl-C или меню программы редактирования. Вы также можете копировать ранее набранный текст из окна терминала.

Для отправки фрагмента текста в Tera Term просто поместите курсор в окно терминала и нажмите правую кнопку мыши. Если текст короткий (1 строка), он будет немедленно передан в AFS. Если текст состоит из нескольких строк, откроется окно редактирования буфера обмена. Вы можете отредактировать текст перед отправкой (см. Рисунок 2.14).

В других программах передача текста в окне терминала может потребовать нажатия клавиш Ctrl-V или выбора соответствующей команды из меню.

Если вы новичок в использовании терминала, рекомендуем изучить все возможности имеющейся в вашем распоряжении программы. Многие функции могут оказаться полезными при работе с AFS и Форт-машинами.

Например, многие программы терминала (в том числе и Tera Term) имеют функцию записи лога. Весь введенный вами текст и ответы Форт-системы будут автоматически сохраняться в текстовом лог-файле. Вы можете легко создать файл исходного текста вашей программы, обработав лог-файл в текстовом редакторе.

2.10.1. "Тихий" режим. Слово QUIET-MODE

Если при передаче текстового файла вы не хотите видеть сообщение "Ok" после каждой строки, включите "тихий" режим обмена с интерактивной системой программирования. В этом режиме AFS сообщает только об ошибках, сообщение "Ok" не выводится.

Для управления используйте слово QUIET-MODE (x --).

Для включения "тихого" режима поместите в стек число 1 и выполните **QUIET-MODE**. Чтобы вернуться к нормальной реакции системы на ввод текста, перед выполнением слова поместите в стек ноль.

При создании файла с исходным текстом программы добавьте

```
1 QUIET-MODE
```

в первой строке, а

```
0 QUIET-MODE
```

– в последней строке файла.

Текст нашей первой программы в файле будет выглядеть так:

```
1 QUIET-MODE \\ Файл HIST.AF
\\ Программа построения гистограммы из символов "*"
\\ Слово для вывода одной "звездочки"
: star \ ( -- )\ ." *" ;
\\ Слово для вывода строки "звездочек" заданной длины
: stars \ (n -- )\ 0 DO star LOOP NL ;
\\ Слово для ограничения длины строки шириной экрана
: limit79 \ (n -- n|79)\ DUP 79 > IF DROP 79 THEN ;
\\ Главное слово – для запуска программы ввести в стек
\\ числа для отображения и выполнить hist
: hist \ (.. n -- )\ DEPTH 0 DO limit79 stars LOOP ;
0 QUIET-MODE
```

При загрузке через терминал вы увидите "Ok" только один раз – по окончании передачи текста.

2.11. Автономное выполнение программы

Мы только что создали полноценную программу на языке Форт и можем выполнить ее в интерактивном режиме. Однако, конечной целью разработки программы является ее автономная работа в Форт-машине без участия оператора. Другими словами, программа должна выполняться сразу после включения питания.

2.11.1. Сохранение программы пользователя. Слово SAVE-PROGRAM

Прежде чем сохранить нашу программу для автономного выполнения, надо в нее кое-что добавить.

При запуске в автономном режиме AFS начинает выполнение программы пользователя с последнего слова, определенного через двоеточие.

В нашей программе таким словом будет **hist** , но начинать выполнение с него нельзя, поскольку ему требуются данные в стеке параметров (при старте системы стек пуст).

Создадим еще одно слово, которое будет выполнять **hist** в цикле и предоставлять ему необходимые параметры.

Мы используем два еще незнакомых вам слова **BEGIN** и **AGAIN** . Слова, размещенные между ними, будут выполняться снова и снова, без конца.

Позже мы рассмотрим использование этих слов, а пока определим слово **Wave** :

```
: Wave BEGIN 5 10 15 20 25 20 15 10 hist AGAIN ;
```

Выполнять это слово не надо! Оно никогда не завершается и вам придется перезагружать систему, чтобы начать все заново.

Слово **Wave** просто размещает в стеке параметры для слова **hist** и выполняет его. Повторение создает эффект "волны" из "зведочек" на экране.

Теперь можно сохранить программу для автономного выполнения – при старте будет автоматически выполнять последнее слово, то есть Wave .

Введите

```
SAVE-PROGRAM
```

Как только вы нажмете Enter, Форт-система запишет вашу программу в постоянную память и выполнит другие необходимые действия, что приведет к завершению сеанса интерактивного программирования.

Вы увидите системное меню:

```
MEMORY OCCUPIED BY USER PROGRAM
1= ERASE PROGRAM MEMORY
2= COPY USER PROGRAM
```

Это меню выводится каждый раз, когда система включается в режиме интерактивного программирования, но память занята программой пользователя.

С первым пунктом меню все очевидно – это очистка памяти, а вот второй пункт требует пояснений.

Форт-машины AFM имеют встроенные средства репликации программ, то есть копирования вашей программы в другие модули AFM. Для передачи данных используется канал AFM-Link. Подробную информацию о репликации программ вы можете получить из технической документации на модули AFM.

2.11.2. Автономный режим работы Форт-машины

Чтобы перейти к автономной работе, отключите модуль AFM от порта USB (если используется питание от другого источника, отключите и его).

Снимите перемычку Mode на плате модуля. Теперь снова подключите модуль AFM к терминалу – вы увидите картину "волны" из "звездочек".

Чтобы вернуться к интерактивному режиму программирования, необходимо установить на место перемычку Mode и удалить программу из памяти. Как это сделать мы уже обсуждали – см. п. 2.2.2 Очистка памяти Форт-машины на стр. 18.

3. Представление данных в Форт-системе

Как вы уже убедились, с помощью интерактивной среды программирования вполне возможно составлять программы на языке Форт, даже не понимая, как устроен компьютер, который вы используете. Для решения время от времени несложных задач этого может быть достаточно, но для построения новых систем из Форт-машин вы должны иметь о них лучшее представление.

Как связана память компьютера и данные? Вся информация, поступающая в компьютер и обрабатываемая программой, представляется в виде двоичных чисел, разбитых на байты. Именно в таком виде она хранится в физической памяти. Все другие способы представления данных – лишь условности, принятые для облегчения программирования. Далее мы рассмотрим, какие средства представления разнообразных данных предлагает Форт и какие правила вы должны соблюдать, работая с интерактивной средой программирования.

3.1. Основы вычислительной техники

Обсуждение дальнейших тем предполагает, что вы знакомы с основами организации хранения данных в компьютерах в виде двоичных битов и байтов. Для новичков (и тех, кто что-то забыл) ниже представлены краткие сведения об основах вычислительной техники.

3.1.1. Двоичное кодирование

Представьте переключатель, например, для включения освещения. Такой переключатель может быть либо включен, либо выключен, иными словами - имеет два состояния. Можно закодировать их так, чтобы состояние переключателя представлялось числами. Будем считать, что включенный переключатель соответствует единице, а выключенный – нулю.

Предположим, что имеется 8 таких переключателей. Если состояние каждого из них представлять единицей или нулем, мы сможем составить некоторое количество чисел.

В таблице показаны числа, построенные таким образом. Восемь переключателей могут представлять 256 значений (от 0 до 255). Количество значений получается из количества состояний одного переключателя (2) и количества переключателей: с помощью n переключателей можно запомнить 2^n чисел ($2^8 = 256$).

Таблица 3.1 Двоичное кодирование чисел

Состояние	Число	Состояние	Число	Состояние	Число	Состояние	Число
00000000	0	00001001	9	. . .	. . .	11110111	247
00000001	1	00001010	10	11101111	239	11111000	248
00000010	2	00001011	11	11110000	240	11111001	249
00000011	3	00001100	12	11110001	241	11111010	250
00000100	4	00001101	13	11110010	242	11111011	251
00000101	5	00001110	14	11110011	243	11111100	252
00000110	6	00001111	15	11110100	244	11111101	253
00000111	7	00010000	16	11110101	245	11111110	254
00001000	8	. . .	. . .	11110110	246	11111111	255

Все современные компьютеры используют двоичное кодирование, поскольку это самый удобный способ представления информации с помощью электрических сигналов.

Память компьютера — просто большое количество переключателей, которые могут включаться и выключаться с огромный скоростью.

Каждый переключатель в памяти называется *битом*, от английского binary digit (двоичная цифра). Числа из нулей и единиц, которые мы представили в таблице, называются *двоичными числами*.

Если значение содержимого бита равно 1, то говорят, что он *взведен* или *установлен*, если значение равно 0, то говорят, что он *сброшен* (иногда - *очищен*).

Работать с отдельными битами не слишком удобно, поэтому их объединяют в группы. Исторически сложилось так, что самая маленькая группа содержит 8 бит. Такой 8-разрядный элемент памяти называют *байтом*, в байтах также принято измерять размер памяти.

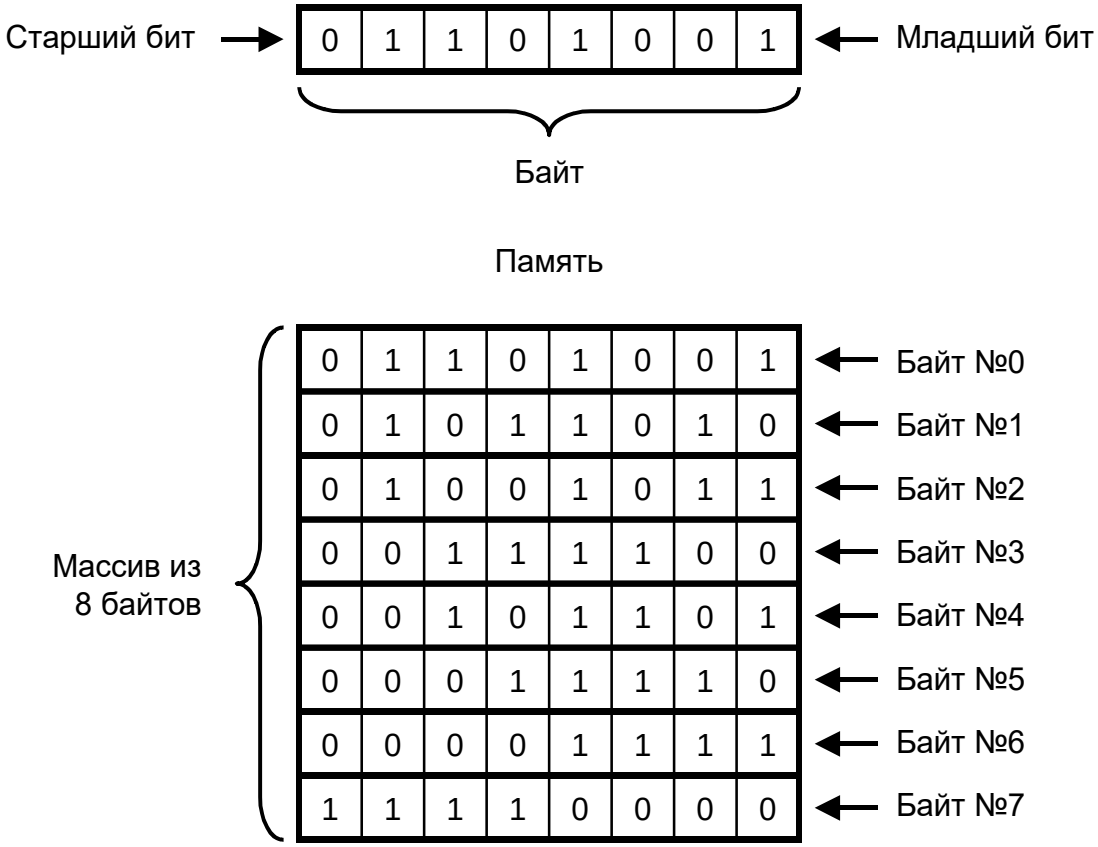


Рисунок 3.1 Биты и байты в памяти

Посмотрите на рисунок. Здесь мы изобразили отдельный байт, состоящий из 8 битов и фрагмент памяти, состоящий из 64 битов. Если биты памяти разложить по восемь, как на рисунке, получится 8 байтов, непрерывно размещенных в памяти. Такая область памяти, состоящая из последовательно идущих байтов, называется *массивом*. Аппаратура компьютера всегда считывает и изменяет биты памяти только группами, не менее чем по 8 бит, то есть не менее, чем один байт за раз. В большинстве случаев считывается или изменяется содержимое двух, четырех или даже восьми байтов и более, но считать или

записать отдельный бит в памяти компьютер не может. Таким образом повышается эффективность работы оборудования.

3.1.2. Адресация

Для того, чтобы аппаратура компьютера могла находить именно тот байт памяти, который потребовался для обработки, всем байтам памяти присвоены порядковые номера (см. Рисунок 3.1).

Число, значение которого представляет порядковый номер байта от начала памяти, называется *адресом байта* или просто *адресом*.

Счет начинается с 0, таким образом, адрес 0 соответствует самому первому байту памяти, адрес 1 – второму байту и так далее. Используя адрес, можно прочитать значение байта или записать новое.

Зная адрес последнего байта, можно сказать чему равен размер памяти – он будет на 1 больше адреса. В вычислительной технике счет адресов всегда ведется с 0, помните об этом, вычисляя размеры массивов.

Для обозначения больших объемов памяти применяются слова с префиксами кило-, мега- и гига-, которые в метрической системе означают соответственно тысячу, миллион и миллиард (например, килограмм = 1000 грамм). В вычислительной технике применяются другие единицы, близкие к указанным, но привязанные к степени числа 2.

Килобайт (сокращенно KB) это 2^{10} = 1024 байта. *Мегабайт* (сокращенно MB) равен 1024 килобайтам, то есть $2^{10}x2^{10} = 2^{20}$ = 1048576 байтам. *Гигабайт* (сокращенно GB) составляет 1024 мегабайт, или $2^{10}x2^{20} = 2^{30}$ = 1073741824 байт.

В некоторых случаях вместо байтов измеряют число битов, обычно это касается цифровой связи или представления в цифровом виде звука и изображения. Соотношения остаются те же, а получаемые величины называются *килобит*, *мегабит* и *гигабит*. Обычно их обозначают Kb, Mb, Gb (то есть с маленькой буквой b вместо большой), но строго этого правила не придерживаются.

3.2. Системы счисления

Когда мы составляем числа из отдельных битов, используется так называемая двоичная запись - соглашение о записи чисел с помощью только двух цифр 0 и 1 (отсюда двоичная цифра, или бит). В повседневной жизни мы имеем дело с записью, которая использует 10 цифр (от 0 до 9) или так называемой десятичной записью, но компьютеры имеют дело с единицами и нулями, то есть с двоичной системой.

В вычислительной технике иногда бывает удобно работать с числами, использующими для записи 8 цифр (от 0 до 7), то есть восьмеричную запись, а очень часто удобнее использовать 16 чисел (от 0 до 9 и от A до F), или так называемую шестнадцатеричную запись. В шестнадцатеричной записи десятичное число 10 обозначается буквой A, 11 - буквой B и так далее, число 15 — буквой F, число 16 обозначается как 10.

Количество различных цифр, используемых для представления чисел, называется основанием системы счисления или просто основанием. При двоичной записи основание равно 2, при восьмеричной - 8, при десятичной - 10, при шестнадцатеричной — 16.

Далее по тексту система счисления, в которой записано число, может быть записана как нижний индекс. Например, запись

$1B_{16} = 27_{10}$

означает "1B в шестнадцатеричной системе равно 27 в десятичной". Если система счисления понятна из контекста, нижний индекс не указывается.

Отличие языка Форт состоит в том, что, хотя все числа он хранит в двоичной форме (как и другие языки программирования, что определяется аппаратурой, а не программами), он может принимать их и отображать с любым основанием вплоть до 64. Далее вы увидите, какие преимущества дает эта возможность Форта.

В системной области памяти AFS постоянно хранится число, которое представляет основание системы счисления, действующей в настоящий момент при вводе и выводе чисел.

3.2.1. Слова RADIX! и RADIX@

Для изменения текущего основания системы счисления используется слово **RADIX!**. Слово извлекает значение с вершины стека и записывает его в системную переменную в качестве нового основания.

Например,

```
2 RADIX!
```

установит двоичную систему счисления, если текущая система — десятичная.

Если уже установлена двоичная система, Форт выдаст ошибку, поскольку двойка — недопустимая цифра в двоичной системе.

Чтобы посмотреть, чему равно основание в текущий момент, используется слово **RADIX@**. Слово помещает на вершину стека значение системной переменной, определяющей текущее основание системы счисления.

Если вы введете

```
RADIX@ .
```

то получите

```
10 Ok
```

независимо от того, какая именно система счисления используется. Происходит это потому, что преобразование числа в текст происходит на основании текущей системы счисления.

Основание счисления во всех случаях представляется как 10, так как это - два в двоичной системе, 8 десятичное - это 10 в восьмеричной системе и 16 десятичное соответствует 10 в шестнадцатеричной системе.

Получается, что мы не можем узнать, в какой системе счисления осуществляется ввод и вывод чисел? Следующие три слова помогут выйти из этого затруднения.

3.2.2. Слова BIN, DECIMAL и HEX

Обычно интерактивная система программирования работает с десятичным представлением чисел, то есть с основанием 10. Однако, в процессе написания программ основание системы может быть изменено. Для возврата в десятичную систему язык Форт предоставляет слово **DECIMAL** (десятичный).

Введите

```
DECIMAL
```

Система счисления может быть изменена на шестнадцатеричную (основание 16), если вы введете

```
16 RADIX!
```

Теперь, когда система использует шестнадцатеричную систему счисления, вы можете ввести

```
1A DECIMAL .
```

и получите

```
26 Ok
```

Число 26 в десятичной записи представляет то же самое значение, что 1A в шестнадцатеричной. Следующий пример произведет обратное действие:

```
26 16 RADIX! .
```

Результат будет 1A, то есть шестнадцатеричный эквивалент числа 26.

Шестнадцатеричная система счисления используется очень часто, поэтому, чтобы не писать 16 RADIX! постоянно, в языке Форт предусмотрено слово **HEX** (сокращение от hexadecimal – шестнадцатеричный). Оно устанавливает шестнадцатеричную систему счисления.

Слово **BIN** (сокращение от binary – двоичный) устанавливает двоичную систему счисления.

Восьмеричная система используется редко, поэтому специальное слово в Форте для нее не предусмотрено. Если вам по какой-то причине оно понадобится, вы легко можете определить его самостоятельно.

Введите

```
DECIMAL

: Octal 8 RADIX! ;
```

Теперь вы сможете использовать восьмеричную систему счисления для ввода и вывода чисел в дальнейших экспериментах, просто выполнив слово **Octal**.

Если вы забыли текущее основание, то слово **DECIMAL** всегда возвратит вас к основанию 10, независимо от того, в какой системе вы работали до этого. Хорошей привычкой может стать возврат к десятичной системе всякий раз, когда нет необходимости использовать какую-либо другую.

В стандарте Форт нет слов RADIX! И RADIX@. Основание системы счисления хранится в переменной BASE и для доступа к ней необходимо использовать пару слов BASE ! или BASE @. В Advanced Forth слово BASE зарезервировано для высокоуровневых систем.

Слова BIN, DECIMAL и HEX являются словами немедленного выполнения в Advanced Forth, но не в стандарте Форт. Применение этих слов в AFS стало удобнее. (О словах немедленного выполнения см. стр. 120).

3.2.3. Использование неординарной системы счисления

Как мы уже говорили, в Форте можно задать любое основание системы счисления. При работе с портами ввода-вывода эта возможность очень облегчает программирование и позволяет избежать ошибок.

Например, для установки режимов работы выводов порта PA модуля AFM необходимо записать в регистр конфигурации число, где каждому выводу соответствует два бита. То есть, биты 1:0 содержат режим вывода PA0, биты 3:2 – режим вывода PA1, биты 5:4 – режим PA2 и так далее.

Эти два бита принимают 4 значения:

00_2 = вход порта PA,

01_2 = выход порта PA,

10_2 = вывод используется периферийным устройством,

11_2 = вывод подключен к входу АЦП.

Обычно при программировании таких битовых полей используют либо двоичные, либо шестнадцатеричные числа.

Если мы хотим установить для PA0 режим АЦП, для PA1 режим входа, PA2 сделать выходом, а PA3 предоставить периферийному устройству, мы должны передать в регистр конфигурации вот такое двоичное число (слева бит 7):

10010011_2

Получилось 8 бит, целый байт. Порт имеет до 16 выводов, соответственно в программе придется обращаться с числами из 32 символов, что не очень удобно.

Можно представить то же значение в шестнадцатеричной системе:

$10010011_2 = 93_{16}$

Запись стала компактнее, но понять, какой режим соответствует каждому выводу, стало сложнее.

Вспомним, что два бита дают всего 4 возможных значения:

$00_2 = 0$, $01_2 = 1$, $10_2 = 2$ и $11_2 = 3$.

Установим систему счисления с основанием 4 и перепишем наше число соответствующим образом:

```
DECIMAL 4 RADIX!
```

```
2103
```

Теперь в стеке лежит число, которое мы можем записать в регистр конфигурации, а его представление позволяет сразу сказать, какой режим устанавливается для каждого вывода порта.

Чтобы убедиться, что это то же самое число, вернем двоичную систему и распечатаем число из стека:

```
BIN .
```

получаем

```
10010011 Ok
```

Мы еще вернемся к программированию портов ввода-вывода в соответствующем разделе.

3.3. Биты, байты и ячейки

С помощью 8 битов или 1 байта могут быть представлены 256 чисел – от 0 до 255. Этого явно недостаточно для каких-либо вычислений, поэтому для представления чисел используется объединение двух или более байтов.

С помощью двух байтов (16 битов или *разрядов двоичного числа*) могут быть представлены уже 65536 чисел - от 0 до 65535. Этого достаточно для большого числа применений, но иногда и этого мало. Тогда количество совместно используемых байтов

удваивают и получают 32 разряда (4 байта), с помощью которых можно представить 4294967296 чисел.

Такое объединение нескольких байтов в языке программирования Форт называется *ячейкой*.

В архитектуре AFS могут использоваться ячейки разных размеров – из 2 байтов (16-битные), из 4 байтов (32-битные) и 8 байтов (64-битные). Возможно создание систем с ячейками из большего количества байтов.

В программе могут использоваться ячейки только одного размера. Связанные с размером ячеек особенности программирования будут рассмотрены в соответствующих разделах.

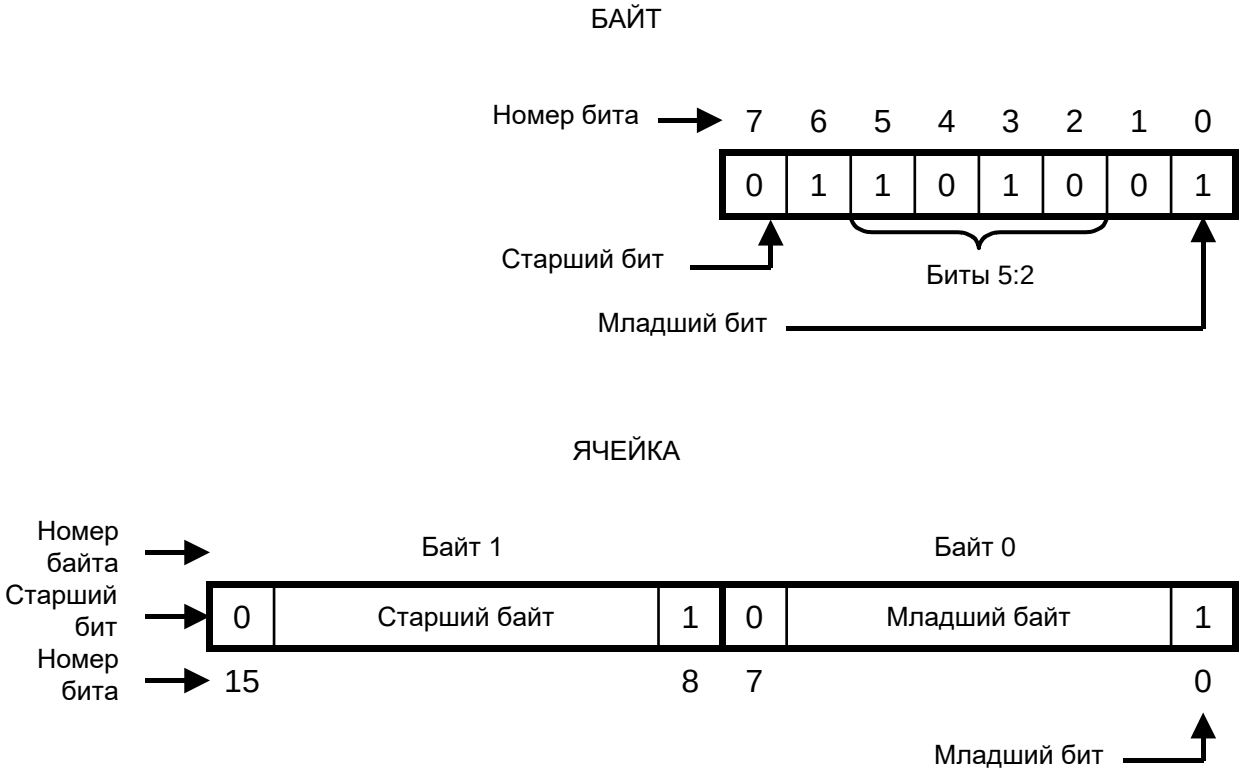


Рисунок 3.2 Биты, байт и ячейка

Ячейка является основным элементом представления информации в системе. В ячейках хранятся числа и адреса.

В AFS адресация данных осуществляется по принципу "младшее значение по меньшему адресу". Применительно к различным типам данных этот принцип реализуется следующим образом.

Биты в двоичных числах нумеруются справа налево, начиная с нуля. Нулевой бит является младшим значащим битом. Старшим значащим битом является крайний левый бит с наибольшим порядковым номером.

Если требуется указать на группу смежных битов внутри байта (такая группа называется *поле битов* или *битовое поле*), указывают номер первого бита группы и номер последнего бита группы через двоеточие. На рисунке 3.2 показано поле битов 5:2 имеющее значение 1010_2.

Если говорить о битах ячейки, то младший бит первого байта ячейки будет иметь номер 0, старший бит первого байта будет иметь номер 7, младший бит второго байта ячейки будет иметь номер 8 и так далее.

При сохранении ячейки в памяти младший байт ячейки записывается по адресу addr, являющемуся адресом данной ячейки. Следующий байт ячейки записывается по адресу addr+1 и т.д. Самый старший байт ячейки из четырех байт сохраняется по адресу addr+3.

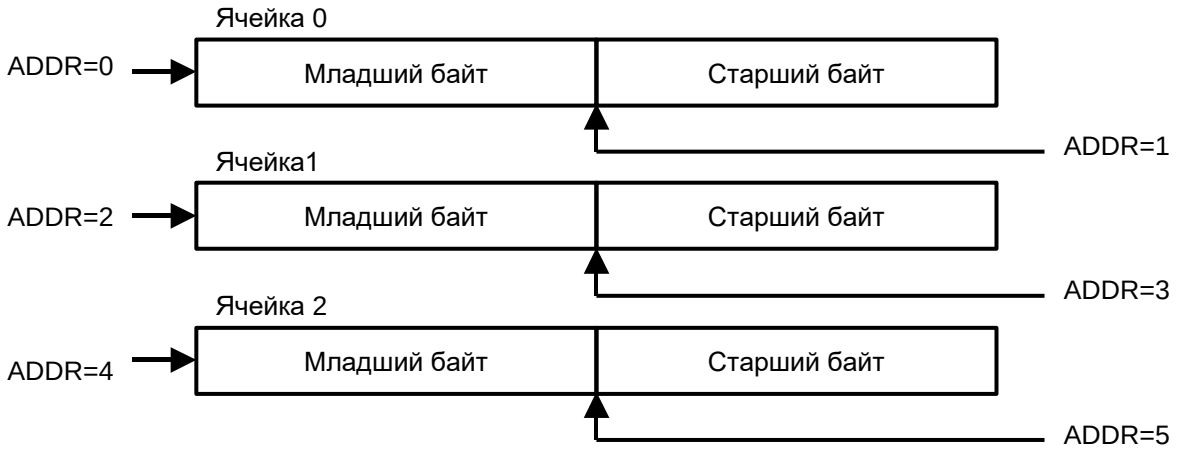


Рисунок 3.3 Адресация ячеек в памяти

Одиночные ячейки и массивы ячеек имеют адреса, кратные размеру ячейки в байтах, т.е. выровненные на границу ячейки (2 или 4 байта). Это требование должно соблюдаться для обеспечения переносимости Форт-программ на разную аппаратуру.

Большинство современных процессоров требуют, чтобы при обращении к памяти адрес был выровнен в соответствии с размером данных. Нарушение выравнивания фиксируется процессором как аппаратная ошибка.

В случае нарушения выравнивания в AFS выдается сообщение

```
SYSTEM ERROR
```

Выполнение слова, вызвавшего ошибку, немедленно прекращается, а стек приводится в исходное состояние.

Адреса памяти, выровненные на границу ячейки, обозначаются a-addr (aligned address – выровненный адрес).

Обозначение addr используется в описании слов Форта для произвольного невыровненного адреса памяти или адреса байта.

При записи значения ячейки в стек порядок сохранения байтов такой же, как при сохранении значения в памяти (поскольку стеки сами являются массивами памяти).

3.4. Представление текста

Текст, выводимый на экран или печатающее устройство, представляет собой последовательность изображений букв, цифр, знаков препинания и т.д. Для удобства обработки сами изображения хранятся в аппаратуре видеосистемы или принтера, а в программах используются только короткие коды, по которым аппаратура находит соответствующую "картинку".

3.4.1. Символы

Дополнительным типом данных в языке Форт является *символ*. Символы сохраняются в байтах.

Одиночные символы и массивы символов в памяти имеют адреса, кратные байту и для них используется обычное обозначение addr или c-addr, если требуется подчеркнуть, что это адрес символа.

При сохранении символа в стеке его значение помещается в ячейку, занимая младший байт. Старшие биты ячейки обнуляются. Таким образом, независимо от размера ячейки, используемого в программе, символы всегда сохраняют свой размер.

Название "символ" используется потому, что буквы, цифры и другие знаки текста, а точнее - коды соответствующих символов, традиционно занимают один байт. Во многих языках программирования имеется тип данных "символ" (char), фактически соответствующий байту. Так сложилось исторически, в языке Форт в том числе.

В документации языка Форт символы условно обозначаются буквой "c" или словом char (от английского character). Обозначение char обычно используется в стековых диаграммах.

В некоторых случаях, когда речь идет о представлении текстовой информации, между символом и байтом может быть разница. Например, при передаче текстов по последовательным каналам связи символ может иметь длину 7 бит, а не 8, как байт.

В официальном стандарте языка Форт сказано, что символ может занимать в памяти более 1 байта. Фактический размер символа в стандартном Форте зависит от реализации системы. В Advanced Forth символ всегда занимает один байт в памяти.

Мы будем использовать официальный термин "символ" в описании слов Форта и в тех случаях, когда речь идет о символах в буквальном значении (т.е. алфавитно-цифровые символы). Когда же надо подчеркнуть, что мы говорим о произвольном числе, занимающем 8 бит, будет использоваться термин "байт".

3.4.2. Кодирование алфавитно-цифровых символов

Как было сказано выше, байты в памяти могут представлять алфавитно-цифровые символы. Символы просто кодируются числами, при этом наиболее распространенным является код ASCII, полное наименование которого American Standard Code for Information Interchange (см. Приложение B).

Первоначально код ASCII предназначался для телеграфии, а с появлением ЭВМ стал использоваться в вычислительной технике, потому что для ввода и вывода информации к первым компьютерам подключали устройства, заимствованные из телеграфного оборудования.

Коды ASCII содержат значения для представления букв латинского шрифта, цифр, знаков препинания и так называемых *управляющих символов*. Управляющие символы перекочевали в компьютеры прямо из устройств автоматической связи, поэтому некоторые названия могут быть непонятными.

Управляющие символы используют коды от 0 до 31. Управляющие коды ASCII довольно часто встречаются при обработке текста и в командах управления терминальными устройствами (см. стр. 61).

В стандартном коде ASCII фактически используется только 7 битов (то есть значения от 0 до 127). В современных компьютерах используются все 8 битов байта. Значения кодов от 128 до 255 называются расширенными кодами ASCII и используются для кодирования

символов национального алфавита, например, кириллицы или греческого, символов графики или математических символов.

Символы, соответствующие расширенным кодам, в разных системах могут выглядеть по-разному. Для правильной передачи таких символов существуют специальные таблицы кодировки. Если две системы используют одинаковые таблицы, текст, написанный в одной системе будет правильно выглядеть и во второй.

Форт-система обрабатывает только символы, коды которых находятся в диапазоне значений от 0 до 127. Например, при вводе новых определений в словарь автоматически выполняется преобразование букв к верхнему регистру, но только для символов латиницы. Символы расширенной кодировки ASCII передаются без изменений.

3.4.3. Счетные строки

Естественным представлением текста являются строки. Существует несколько форматов текстовых строк, отличающихся способом определения длины текста.

Многие языки программирования используют простой формат строки, в котором конец текста обозначается специальным символом. Чаще всего этим символом является NUL, т.е. байт, содержащий значение ноль. Такие строки часто называют ASCIIZ (Z от zero).

В Форт-системе используются так называемые *счетные строки* (см. рисунок).

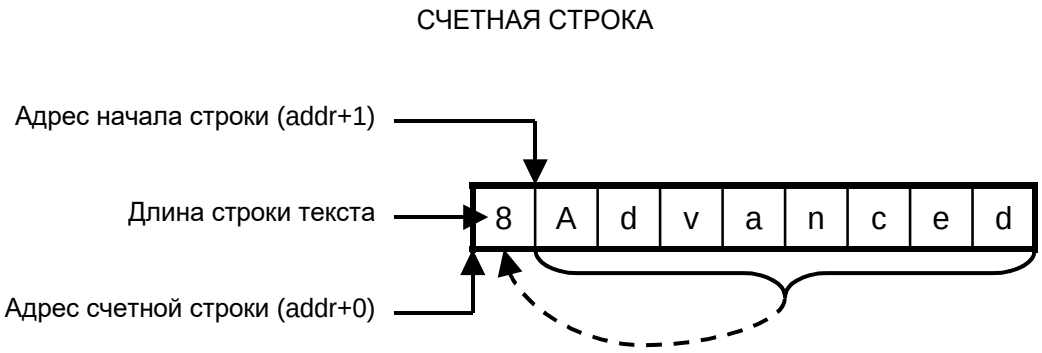


Рисунок 3.4 Счетная строка

Счетная строка представляет собой массив байтов. Первый байт массива содержит длину строки, последующие байты содержат символы строки.

Длина строки является целым числом без знака и может принимать значения от 0 до 255.

Пустая счетная строка должна состоять из одного байта, содержащего значение 0.

Если строка, признаком конца которой является символ с кодом 0 (строка ASCIIZ), должна быть представлена в виде счетной строки, нулевой символ также включается в длину строки.

3.5. Представление чисел

Для представления чисел в Форте используются ячейки. Как вы уже знаете, ячейки могут иметь разный размер. В нашей учебной AFS используются ячейки из двух байтов, то есть занимающие 16 битов памяти.

Диапазон представления чисел без знака для одной ячейки от 0 до 65535 (2^{16}=65536).

Числа со знаком в одной ячейке имеют могут принимать значения от -32768 до 32767.

Условное обозначение числа со знаком в стековой диаграмме - символ n, для числа без знака применяется символ u.

Для чисел со знаком используется дополнительный код.

3.5.1. Кодирование чисел со знаком

Как было сказано выше, с помощью 16 битов можно представить числа от 0 до 65535 (2^{16}=65536). Давайте проверим это утверждение на практике.

Введите

```
65535 .
```

Результат получится такой:

```
-1 ok
```

Что произошло? Очевидно, то, что система воспринимает число 65535 как отрицательное число. Теперь попробуйте

```
65534 .
```

и получите

```
-2 ok
```

А теперь введите

```
32768 .
```

система выдаст

```
-32768 ok
```

В то же время, если ввести

```
32767 .
```

вы получите

```
32767 ok
```

Что же происходит? Все правильно, поскольку система работает с так называемыми числами в дополнительном коде со старшим знаковым битом. Все, что находится в диапазоне от 32768 до 65535, система рассматривает как отрицательные числа, если вы не указываете, что они вовсе не имеют знака. Попробуйте ввести

```
65535 U.
```

Результат

```
65535 Ok
```

получится таким, каким он и должен быть. Если вы используете слово **`U.`** с другими числами, включая те, которые больше 32767, вы будете получать те же числа, что и ввели. Слово **`U.`** предназначено для вывода чисел, не принимая во внимание знак числа, то есть "чисел без знака".

Теперь попробуйте ввести

```
-1 U.
```

в результате получится число 65535. Система ошиблась? Вот что происходит на самом деле. Мы перевели числа из наших примеров в двоичную кодировку, то есть в тот вид, в котором они хранятся в памяти и используются аппаратурой, и поместили в таблицу.

Таблица 3.2 Кодирование чисел со знаком и без знака

Число без знака	Двоичное значение	Число со знаком
65535	**1**111111111111111	-1
65534	**1**111111111111110	-2
32768	**1**000000000000000	-32768
32767	**0**111111111111111	32767
32766	**0**111111111111110	32766
0	**0**000000000000000	0

Для компьютера 65535 или –1 одно и то же число (1111111111111111_2), в нем все 16 разрядов установлены в 1. Мы уже упоминали, что кроме нулей и единиц в памяти ничего нет, а числа, символы адреса и команды – абстракции, помогающие в нужный момент представить эти данные в правильном виде.

Если вы внимательно изучили числа в таблице, то наверняка заметили, что пока крайний слева (то есть старший) бит двоичного значения равен 1, это значение может быть представлено и как положительное, и как отрицательное. Как только старший бит становится равен 0, число становится только положительным.

При операциях с числами в дополнительном коде старший бит (мы его выделили) называют знаковым и считают, что значение 1 в знаковом бите является признаком отрицательного числа.

Как именно будет представлено значение, зависит от программы, а точнее – от того, что написал в тексте программы программист.

Принятое в языке Форт соглашение об использовании чисел со знаком называется арифметикой с дополнением по модулю два. Не вдаваясь в причины, скажем, что дополнительная арифметика используется практически во всех вычислительных машинах.

Здесь мы не будем углубляться в математику, но небольшое пояснение дадим. Как известно, отрицательное число связано с равным ему по абсолютному значению следующим образом:

$$|-a| = a$$

$$0 - a = -a$$

$$-a + a = 0$$

Если вычесть любое двоичное число, состоящее из 16 битов, из нуля, вы получите равное ему по величине, но противоположное по знаку число, представленное в виде дополнения по модулю два. Сложение полученного результата с исходным числом даст ноль, как и должно быть.

Таким образом, главное свойство соглашения об арифметике с дополнением по модулю два – соблюдение общепринятых математических правил без дополнительных преобразований чисел.

3.5.2. Числа двойной длины

До сих пор мы говорили о числах, представленных в системе с размером ячейки 16 бит. Иногда бывает необходимо использовать значения, превышающие 65535 для чисел без знака или 32767 для чисел со знаком. Для таких случаев в языке Форт существует еще один тип данных – числа двойной длины.

Числа двойной длины (или просто двойные числа) требуют для хранения пары ячеек. Такие пары (двойные ячейки) адресуются по правилам, применяемым для обычных ячеек. При сохранении двойного числа в памяти младшая ячейка числа записывается по адресу a-addr, являющемуся адресом данной пары ячеек. Старшая ячейка числа записывается по адресу a-addr+n, где n - размер ячейки в байтах.

В Advanced Forth при записи двойных ячеек в стек параметров сначала записывается старшая ячейка, затем младшая (на вершину), что обеспечивает соблюдение принципа "младшее значение по меньшему адресу", так как в памяти стек параметров обычно "растет" от больших адресов к меньшим.

Числа двойной длины в двоичном представлении выглядят точно также, как и обычные, но имеют в два раза больше разрядов, что позволяет использовать их для представления большего диапазона значений. Двойные числа могут быть со знаком и без знака.

Условное обозначение двойного числа со знаком в диаграмме состояния стека - символ d. Для двойных чисел без знака используется обозначение ud.

В стандартном языке Форт определен обратный порядок сохранения двойных чисел в стеке — старшая ячейка на вершине, младшая под ней. Такой порядок был принят в старых больших ЭВМ, но на современных процессорах такая реализация неудобна и снижает производительность.

4. Ввод и вывод с использованием текстовой консоли

Ввод и вывод текста является основным средством связи интерактивной среды программирования с пользователем.

AFS общается с вами через системную текстовую консоль, служебную программу, отвечающая за связь с пользователем через доступные устройства – последовательный или сетевой терминал, клавиатуру и дисплей, и т.п.. Эта программа присутствует во всех Форт-машинах семейства AFM.

Текстовая консоль является виртуальным устройством ввода-вывода. Какое физическое устройство будет использовано для связи с пользователем, зависит от возможностей аппаратуры Форт-машины и системных настроек. В нашем конкретном случае простейшего модуля AFM текстовая консоль работает с последовательным терминалом через универсальный асинхронный приемопередачик микроконтроллера.

Мы еще вернемся к обработке текстов в следующих разделах, а сейчас познакомимся с некоторыми основными принципами взаимодействия с Форт-системой через текстовую консоль.

4.1. Вывод чисел в виде текста

Чаще всего приходится выводить различные числовые значения. Это могут быть числа со знаком и без знака, двойной длины, символы и их коды.

4.1.1. Слова . (точка) и U. (U-точка)

Слово **.** (точка) вам уже известно. Оно извлекает число из стека, преобразует его в текст как число со знаком, используя текущую систему счисления, выводит на консоль и добавляет пробел в конце.

Для вывода чисел без знака используется слово **U.** (U-точка, от unsigned). Оно производит те же действия, что и **.** (точка), но рассматривает все биты ячейки как неотрицательное значение.

4.1.2. Слова D. (D-точка) и UD. (U-D-точка)

Число двойной длины занимают в стеке две ячейки. Для правильного представления и вывода на консоль таких значений используются специальные слова.

Слово **D.** (D-точка) преобразует в текст и выводит на консоль число двойной длины со знаком.

Слово **UD.** (U-D-точка) преобразует в текст и выводит на консоль число двойной длины без знака.

Как всегда, для преобразования используется текущее основание системы счисления.

4.2. Вывод строк и отдельных символов

Вы можете выводить на консоль не только числа, но и текст, представленный строками и отдельными символами.

4.2.1. Слово ." (точка-кавычка)

Первое слово, выводящее на экран строку текста, вам встретилось в самом начале знакомства с языком Форт.

Слово **."** (точка-кавычка) используется внутри определения. Слово сохраняет в метакоде определения текст, начинающийся сразу за отделяющим его пробелом и заканчивающийся кавычкой.

При выполнении определения текст выводится на экран.

Если перед завершающей текст кавычкой имеется пробел, он также будет выведен в конце строки, так как кавычка не является словом языка, а лишь используется словом **."** для нахождения конца строки.

4.2.2. Слова TYPE и COUNT

Для вывода на консоль текстовой строки предназначено слово TYPE (addr u --). Слово **TYPE** извлекает из стека длину строки и адрес первого символа строки.

Как мы помним, в Форте используются счетные строки. Для удобства обращения с такими строками Форт предоставляет слово COUNT (addr1 -- addr2 u) . Слово **COUNT** получает в стеке адрес счетной строки, а возвращает адрес первого символа строки и ее длину, как того требует слово **TYPE**.

Как создать строку в памяти и использовать слова **COUNT** и **TYPE** мы узнаем позже, в разделе, посвященном созданию переменных в памяти.

4.2.3. Слова EMIT, CHAR и [CHAR]

Для вывода на консоль отдельного символа используется слово EMIT (n --).

Если вы введете

```
65 EMIT
```

вы увидите

```
AOk
```

Слово **EMIT** получает число с вершины стека и выводит на экран соответствующий ему символ ASCII. Если вы заглянете в таблицу кодов ASCII (Приложение B), то обнаружите, что 65 как раз и есть код символа «A».

Для получения кода символа можно использовать слово **CHAR**. Оно помещает в стек значение первого символа слова, следующего за ним в потоке ввода.

Введите

```
CHAR A .
```

и вы увидите

```
65 Ok
```

Теперь вы можете самостоятельно проверить, какие символы соответствуют различным кодам и сравнить их с таблицей. Будьте внимательны при выводе управляющих символов!

Вернемся к нашей программе, рисующей гистограммы. Мы определили слово, рисующее "звездочку" следующим образом:

```
: star ." *" ;
```

Используя EMIT можно переписать его так:

```
: star 42 EMIT ;
```

Число 42 – это десятичный код символа "звездочка".

Правильный стиль программирования, в данном случае, подразумевает использование слова **[CHAR]** :

```
: star [CHAR] * EMIT ;
```

Слово **[CHAR]** предназначено для использования внутри определения нового слова. Оно так же определяет код первого символа следующего слова, но не передает его в стек, а сохраняет в определении нового слова. Когда новое слово будет выполняться, код символа будет помещен в стек и его смогут использовать следующие слова определения.

Последний вариант слова **star** дает точно такой же код программы, как и предыдущий, но отличается наглядностью, не требует знания кода ASCII символов и манипуляций с системами исчисления.

4.2.4. Слова SPACE и NL

Некоторые символы и их комбинации так часто выводятся на консоль, что в Форте для них предусмотрели специальные системные слова.

Слово **SPACE** выводит один пробел, а слово **NL** перемещает курсор в начало следующей строки (NL = New Line, новая строка).

В стандартном языке Форт вместо слова NL используется слово CR. В Advanced Forth слово CR – системная константа, возвращающая управляющий код перемещения курсора в начало строки.

4.3. Управление текстовым терминалом

Вместо одиночного управляющего символа ASCII можно использовать последовательности символов, которые терминалы воспринимают как управляющие команды. Для обеспечения совместимости терминальных устройств был разработан стандартный набор таких команд, утвержденный Американским Национальным Институтом Стандартов (ANSI).

Команды ANSI начинаются с управляющего символа с кодом 27 (Escape), за которым следуют один или несколько символов ASCII. Такие команды часто называют Escape-последовательности.

Попробуем воспроизвести некоторые из них. Введите

```
27 EMIT 91 EMIT 50 EMIT 74 EMIT
```

Экран терминала очистится, как только вы нажмете Enter.

С помощью таблицы кодов ASCII можно расшифровать, что мы вывели на терминал последовательность символов ***Escape*****[2J** . Это команда ANSI для очистки экрана.

Создадим слово для очистки экрана терминала:

```
: CLRSCR 27 EMIT ." [2J" ;
```

Здесь вместо **EMIT** для каждого символа мы использовали вывод строки с помощью **."** .

Выполните слово **CLRSCR** – экран очистится.

Было бы неплохо после очистки перемещать курсор в верхний левый угол. Для этого также найдется команда ANSI. Она представляет собой последовательность "Escape[*строка*;*столбец*H", где "строка" и "столбец" – текстовые значения координат курсора на экране. Верхний левый угол экрана имеет координаты 1;1 .

Вы можете создать слово для перемещения курсора "домой":

```
: CRSHOME 27 EMIT ." [1;1H" ;
```

Использование этой и других команд ANSI значительно расширяет возможности управления выводом на терминал.

4.4. Ввод чисел и текста

Для ввода текста Форт-система задействует множество ресурсов. На данном этапе мы не будем углубляться в эту тему, а рассмотрим лишь необходимые основные средства ввода. В первую очередь это касается особенностей ввода чисел в интерактивной среде программирования.

4.4.1. Ввод чисел двойной длины

Когда вы вводите любой текст, Форт-система разбирает его на отдельные слова и проверяет, есть ли такое слово в словаре. Если в словаре слово не найдено, предпринимается попытка преобразовать текст в число, используя действующую систему счисления.

В случае успеха число помещается на вершину стека или компилируется в определение.

До сих пор мы говорили о числах, занимающих одну ячейку. Что делать, если требуется ввести число двойной длины?

Если вы хотите, чтобы введенный текст был преобразован в число, занимающее две ячейки, просто поставьте десятичную точку в любом месте текста:

```
-1234.567
```

Это двойное число! Для вывода значения из стека используйте слово для чисел двойной длины:

```
D.
```

выводит

```
-1234567 Ok
```

Точка отсутствует, но само число выведено правильно.

4.4.2. Слова KEY и KEY?

Слово KEY (-- n) ожидает поступления символа от текстовой консоли и помещает в стек код символа ASCII, введенного с клавиатуры.

Введите

```
KEY
```

и ничего не случится, даже сообщение "Ok" не появится. Теперь нажмите клавишу "A", и на экране тут же появится

```
Ok
```

Теперь введите точку:

```
.
```

и вы увидите, что в стек было помещено число 65:

```
65 Ok
```

Разберемся с происходящим. Когда вы ввели **KEY**, система приостановила выполнение в ожидании нажатия клавиши на клавиатуре. Как только вы нажали "A", соответствующий символу "A" код был помещен в стек и система вывела свое обычное сообщение о готовности к дальнейшим действиям.

Если соединить слово **KEY** и **EMIT**, можно увидеть символ, соответствующей нажимаемой клавише:

```
KEY EMIT
```

Слово **KEY** помещает в стек код символа, соответствующего клавише, а слово **EMIT** преобразует этот код в соответствующий символ на экране.

Если вы не хотите останавливать выполнение программы и ждать, пока на клавиатуре будет нажата клавиша, используйте слово KEY? (-- flag) для проверки наличия ввода с консоли.

Слово **KEY?** проверяет количество символов в буфере ввода консоли. Если буфер пуст, возвращается флаг "ложь". Если слово вернуло флаг "истина", в буфере есть символ и вы можете получить его с помощью слова **KEY** не задерживая выполнение программы.

4.4.3. Слово ACCEPT

Слово ACCEPT (addr n1 -- n2) используется для приема от консоли строки текста с редактированием. Ввод продолжается до нажатия клавиши Enter. Слово **ACCEPT** извлекает из стека адрес буфера ввода и его размер, а возвращает длину принятой строки.

Как создать буфер ввода и испытать слово **ACCEPT** в действии мы расскажем в разделе, посвященном созданию переменных в памяти.

4.4.4. Слова OMIT и WORD

Слово OMIT (char --) получает на вершине стека код символа-разделителя, просматривает поток ввода и удаляет все символы, пока не встретится первый символ-разделитель (который также удаляется). Поясним действие слова на примере.

Вы можете создать слово **(** , которое позволяет использовать комментарии в старом стиле классического Форта:

```
: ( [CHAR] ) OMIT ;
```

При выполнении слова **(** в потоке ввода будут пропущены все символы до закрывающей круглой скобки (которая также удаляется):

```
( комментарий в старом стиле)
```

Слово WORD (char -- c-addr) получает в стеке код символа разделителя, выделяет в потоке ввода слово, ограниченное этим символом, копирует его в секцию данных программы в виде счетной строки и возвращает ее адрес.

Для сохранения строки, содержащей слово, используется динамически распределяемая память данных – см. "Указатель данных и слово HERE" на стр. 77.

5. Использование стека

В языке Форт операции с данными производятся главным образом в стеке. Поначалу стек может показаться вам странным, однако пользоваться стеком очень просто, если вы поймете, что это такое и как с ним работать.

Все данные, вводимые вами с клавиатуры или поступающие из других источников, фактически попадают в стек. Передача информации между словами языка также происходит через стек и именно стек делает Форт-программы такими компактными, гибкими и мощными.

5.1. Организация стека в памяти

Прежде всего разберемся с тем, как стек параметров организован в памяти компьютера.

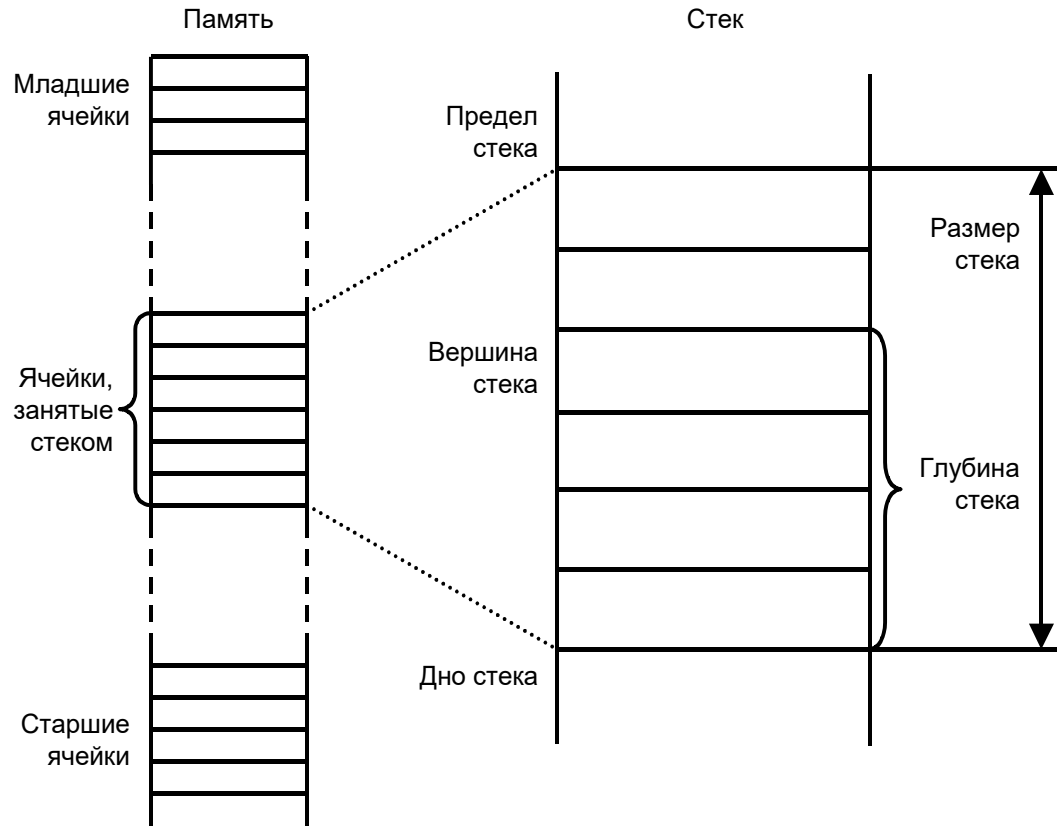


Рисунок 5.1 Стек в памяти

На рисунке мы изобразили память вычислительной машины. В начале памяти расположены младшие ячейки, то есть ячейки, состоящие из байтов с адресами 0, 1, 2 и так далее. Ячейки с большими значениями адресов байтов называются старшими ячейками.

Где-то в произвольном месте памяти операционная система организовала стек параметров, выделив для него массив ячеек.

Размер стека определяется числом ячеек памяти, выделенных системой. Глубина стека определяется числом ячеек стека, занятых данными.

Не путайте глубину стека с его размером! Размер стека – величина постоянная, глубина же постоянно меняется. Когда в стек помещается очередное число, глубина стека увеличивается на одну ячейку, когда число извлекается – глубина уменьшается.

Вершиной стека называется последняя занятая данными ячейка. Дно стека – это адрес памяти, с которого начинаются данные, не принадлежащие стеку.

Предел стека – самая верхняя (на рисунке) ячейка стека. Если вершина стека пересечет эту границу, возникнет переполнение стека и аварийная ситуация.

Первое число, которое было внесено в стек, будет находиться на дне, последнее введенное число - на вершине стека. Числа из стека можно брать только с вершины.

В тексте мы будем часто изображать стек в виде диаграммы состояния, то есть положенным горизонтально, в строчку, причем дно стека находится слева, а его вершина – справа. Такое представление стека весьма удобно.

Если вы введете

```
1 2 3
```

то есть в строчку, как на диаграмме состояния стека, числа в стеке разместятся так:

3 (вершина)

2

1 (дно)

Теперь введите

```
. . .
```

Первым из стека было изъято и выведено на экран число 3, затем то же самое было сделано с числом 2, а потом с числом 1. Обратите внимание, что числа на экране расположены в обратном порядке по сравнению с тем, как они представлены в стеке. Теперь стек пуст.

Попробуйте ввести еще раз

```
.
```

Вы увидите вывод произвольного значения и сообщение системы:

```
P-STACK DESTROYING
```

("Разрушение стека параметров").

Ошибка была вызвана попыткой получить данные из пустого стека.

Обратите внимание, что система не выдает сообщение "Ok", потому что выполнение вашей последней команды привело к ошибке (а значит, не все в порядке).

Еще раз нажмите клавишу ввода, чтобы получить на экране новое приглашение "Ok", хотя в этом нет особой необходимости. Фактически система готова к вводу новой информации, даже если нет сообщения "Ok".

Разрушение стека — весьма опасная ситуация (выше и ниже стека находятся другие данные!), поэтому Форт-система пытается отследить ее и блокировать. К сожалению, это не всегда удается, поэтому будьте внимательны!

5.2. Операции с данными в стеке параметров

В языке Форт имеется множество слов, производящих разнообразные операции с данными в стеке параметров. Приложение С содержит список слов этой группы, доступных в AFS уровня L0 (Таблица С.5).

Для безопасного исследования стека определим новое слово:

```
: .S DEPTH ?DUP IF 0 DO DEPTH 1- ROLL DUP . LOOP
   ELSE ." Стек пуст! " THEN ;
```

Как видите, кроме знакомых вам слов, определение использует много новых слов для операций в стеке, поэтому мы используем его в качестве примера для изучения в конце раздела.

Теперь, если вы введете

```
1 2 3 .S
```

то увидите на экране

```
1 2 3 Ok
```

Введите теперь

```
. . .
```

и вы увидите

```
3 2 1 Ok
```

Ценность слова **.S** в том, что оно ничего не изменяет в стеке. Если вы введете **.S**, когда стек пуст, будет выведено сообщение "Стек пуст!", которое является частью определения этого слова.

Выполняя следующие примеры имейте ввиду, что на дне стека могут оставаться значения от предыдущих опытов, ведь слово **.S** не очищает стек!

Если вы хотите очистить стек параметров, выполните слово **ABORT**. Стек будет приведен в исходное состояние (сообщение **Ok** не выводится, поскольку **ABORT** – слово аварийного завершения программы).

Если вы наберете какую-нибудь абракадабру, например, **QWERTY** , Форт-система, не найдя такого слова в словаре, сообщит

```
WORD DOES NOT EXIST: QWERTY
```

и очистит стек параметров.

Запомните – *стек параметров всегда очищается, если система фиксирует ошибку*.

5.2.1. Слова DROP и 2DROP

Слово **DROP** удаляет ячейку с вершины стека. Например, после ввода

```
1 2 3 DROP .S
```

вы увидите

```
1 2 Ok
```

Помимо того, что слово **DROP** полезно для уничтожения неправильно введенных данных при вычислениях, оно чаще всего используется для того, чтобы убрать какие-либо числа из стека во время выполнения программы. В нашей первой программе мы использовали слово **DROP** для удаления из стека слишком большого числа.

Очевидно, при пустом стеке мы получим сообщение об ошибке:

```
P-STACK DESTROYING
```

Вас может удивлять, почему система не знает о том, что стек пуст, и не игнорирует слово **DROP** в этом случае.

Дело в том, что проверка на наличие ошибок и принятие решения, реагировать на них или игнорировать, занимает немало времени. Слова Форта оптимизированы с точки зрения быстродействия. Если бы в каждое из них было включено принятие решений об ошибках, исполнение слов происходило бы с более низкой скоростью.

В Форте принято, что программист, а не система несет ответственность за предотвращение ошибок. С другой стороны, разрабатывая программу, вы будете отлаживать каждое ее слово, легко обнаруживая ошибки. Поэтому Форт не нуждается в сложных и требующих больших затрат времени методах контроля ошибок, необходимых другим языкам.

Время от времени приходится иметь дело с параметрами, занимающими пару ячеек. Для удаления сразу двух ячеек с вершины стека используйте слово **2DROP**.

Ведите

```
1 2 3 2DROP .S
```

и вы получите

```
1 Ok
```

Чтобы получать в последующих опытах те же результаты, что и в тексте примеров, очищайте стек от данных предыдущих экспериментов. Вы можете использовать слова **DROP** и **2DROP** , если уверены, что в стеке лежит одна или две ячейки. Если ячеек больше или их количество неизвестно, используйте слово **ABORT** .

Приводимые далее примеры подразумевают, что стек изначально пуст.

5.2.2. Слова DUP и 2DUP

Следующее слово вам уже известно, это слово **DUP** (от duplicate), которое создает копию верхней ячейки стека.

Выполнение

```
7 DUP .S
```

приводит к тому, что в стеке окажется

```
7 7 Ok
```

Слово **DUP**, пожалуй, одно из наиболее часто употребляемых слов для операций в стеке.

Аналогичное слово **2DUP** применяется для копирования пары ячеек.

5.2.3. Слово ?DUP

Слово **?DUP** представляет собой специальный вариант слова **DUP**. Оно делает копию числа, находящегося на вершине стека, если оно не равно нулю, и не копирует число, если оно равно нулю.

Например:

```
1 4 5 ?DUP .S
```

дает в стеке

```
1 4 5 5 Ok
```

В то время как

```
1 4 0 ?DUP .S
```

приводит в результате к

```
1 4 0 Ok
```

Таким образом, в последнем случае слово **?DUP** не производит никаких действий.

Слово **?DUP** часто используется вместе с конструкцией IF-THEN, примером чего является наше слово **.S** .

5.2.4. Слова SWAP и 2SWAP

Слово **SWAP** меняет местами две верхние ячейки стека (см. рисунок).

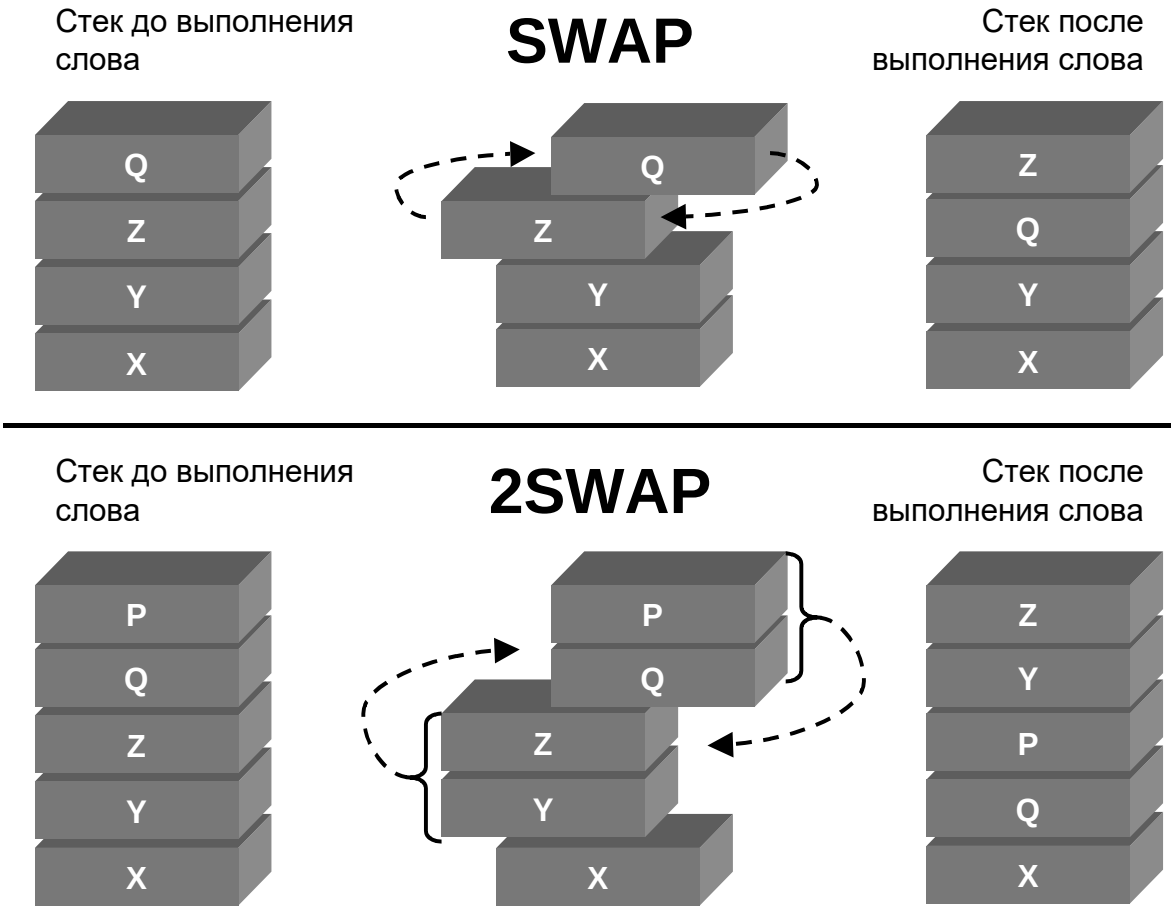


Рисунок 5.2 Действие слов SWAP и 2SWAP

Если ввести

```
1 2 3 4 SWAP .S
```

вы получите

```
1 2 4 3 Ok
```

Это удобно, если нужно сделать операцию вычитания или деления, но операнды стоят в неправильном порядке.

Для перестановки пары ячеек используется слово **2SWAP** :

```
3 4 5 6 7 8 2SWAP .S
```

дает в результате

```
3 4 7 8 5 6 Ok
```

Рассмотренные примеры оставили много значений в стеке. Выполните слово **ABORT** и приведите стек в исходное состояние.

5.2.5. Слова OVER и 2OVER

Попробуйте ввести

```
1 2 3 4 OVER .S
```

и вы увидите

```
1 2 3 4 3 Ok
```

Слово **OVER** копирует вторую ячейку стека на вершину стека (см. рисунок).

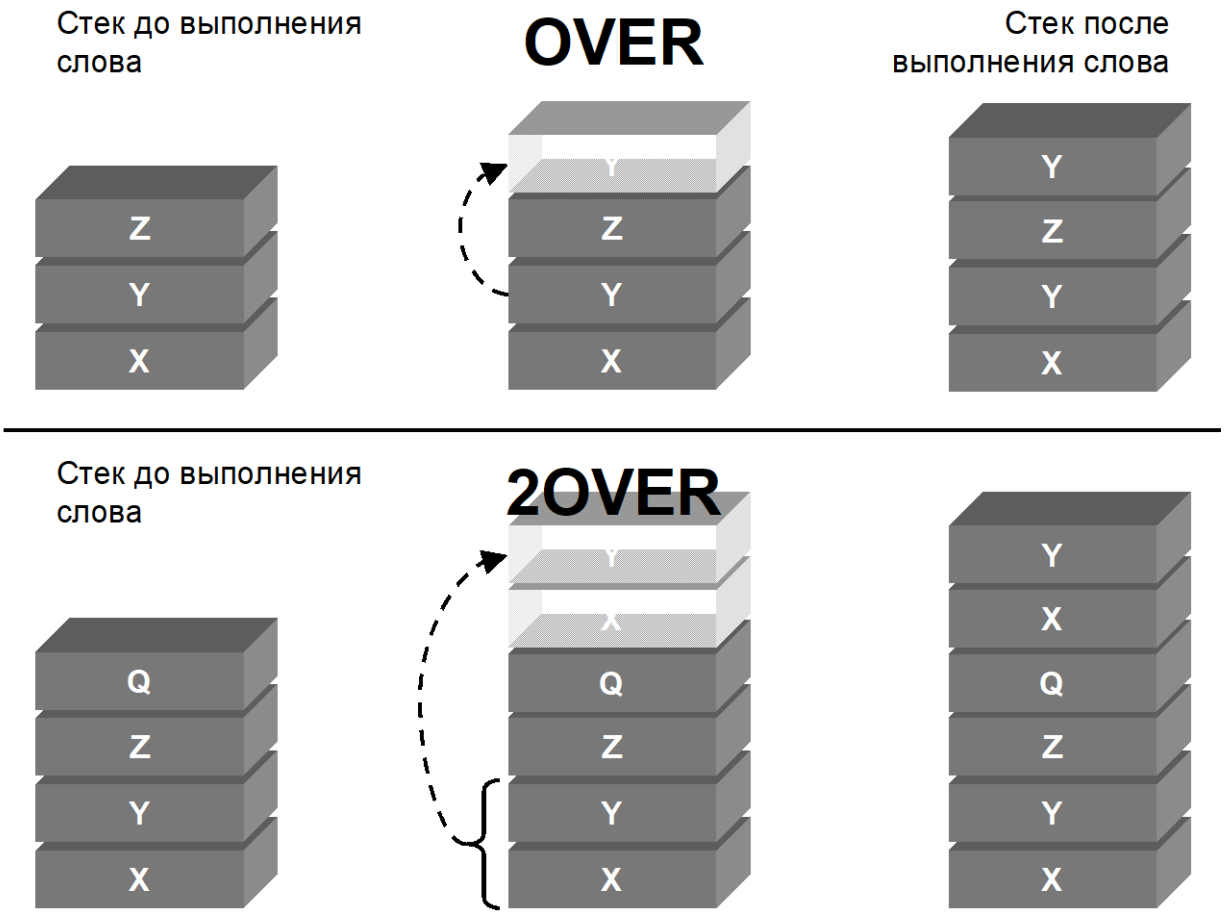


Рисунок 5.3 Действие слов OVER и 2OVER

Слово может быть полезно во многих случаях, когда какое-либо число должно быть использовано несколько раз.

Для операций с парами ячеек предусмотрено аналогичное слово **2OVER** :

```
1 2 3 4 5 6 2OVER .S
1 2 3 4 5 6 3 4 Ok
```

5.2.6. Слово ROT

Слово **ROT** (от rotate) производит ротацию трех верхних ячеек стека, то есть перекладывает третье сверху число в стеке на его вершину, так что

```
3 4 5 6 7 ROT .S
```

приводит к

```
3 4 6 7 5 Ok
```

Третья сверху ячейка стека перемещается на вершину стека, а следующие две ячейки продвигаются на одну позицию в глубь стека (см. рисунок).

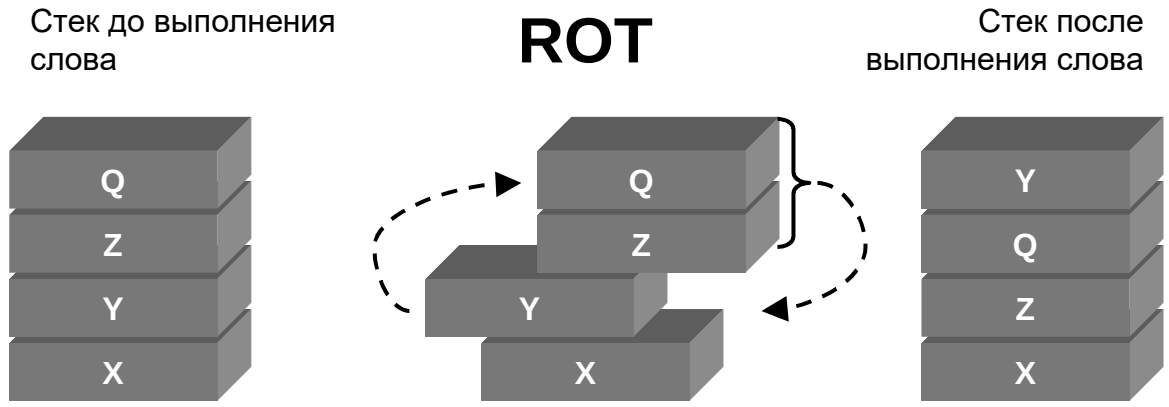


Рисунок 5.4 Действие слова ROT

Применение слова **ROT** поначалу не очевидно, но это слово весьма востребовано. Предположим, что вы хотите определить значение выражения (5+2)*(7+3), если в стеке находится

```
5 2 7 3
```

Для решения введите

```
+ ROT ROT + * .
```

и вы получите правильный результат

```
70 Ok
```

Возможно, вам не совсем понятен этот пример. Давайте проследим его выполнение по шагам, используя диаграммы состояния стека.

После выполнения первого сложения состояние стека изменится следующим образом:

(5 2 7 3 -- 5 2 10)

Результат сложения на вершине стека, но нам надо сложить еще одну пару чисел. Первое слово **ROT** приводит стек к такому состоянию:

(5 2 10 -- 2 10 5)

Как видите, первое слагаемое уже переместилось на вершину, осталось вытащить второе, что и делает второе слово **ROT** :

(2 10 5 -- 10 5 2)

Теперь числа расположены в нужном порядке, можно их сложить:

(10 5 2 -- 10 7)

После выполнения второго сложения в стеке остались только две суммы, которые перемножаются словом ***** .

Если требуется произвести сложные операции в стеке вроде приведенного примера, составление диаграмм состояния стека может оказаться очень полезным. Наиболее часто ошибки в определении слова происходят при операциях в стеке.

5.2.7. Слово PICK

В отличие от предыдущих, слово **PICK** требует наличия на вершине стека аргумента - числа, которое указывает, какую по счету ячейку сверху (не считая аргумента) вы хотите скопировать на вершину стека.

Ячейки рассматриваются как массив. За начало принимается вершина стека, а счет ведется по направлению в глубь стека (см. рисунок).

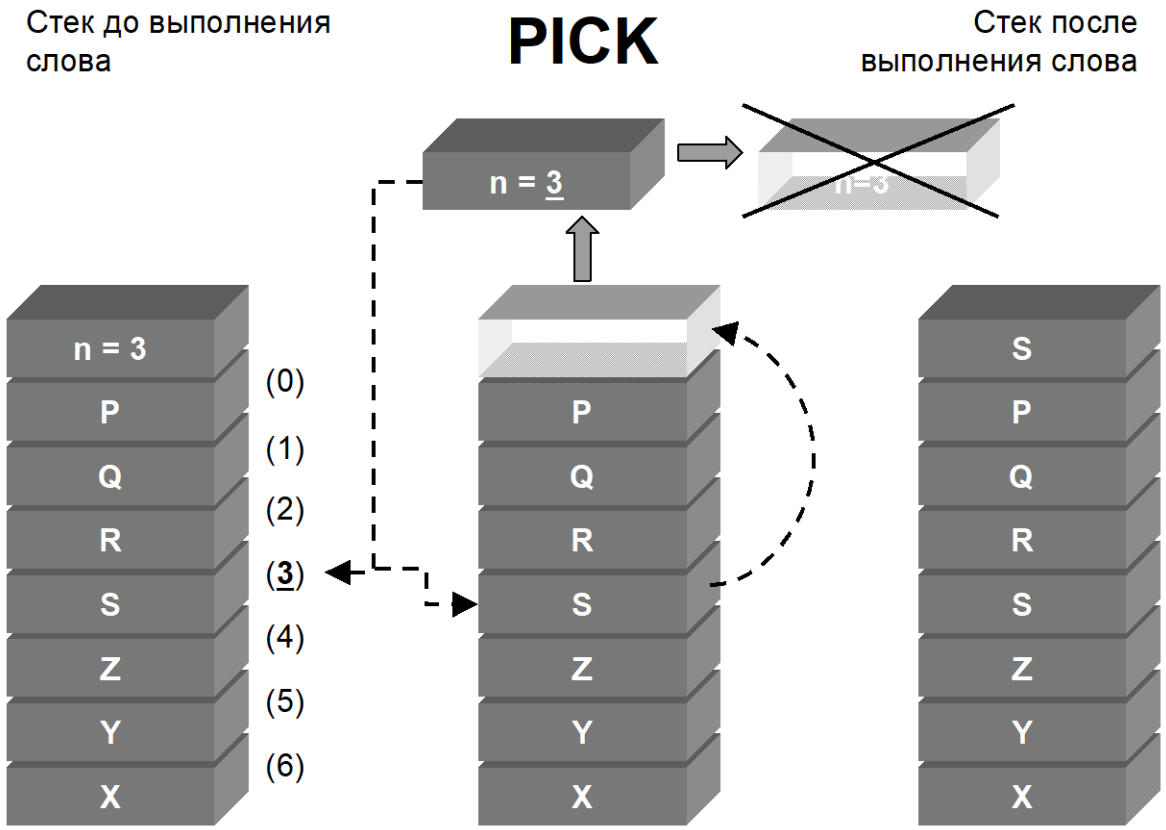


Рисунок 5.5 Действие слова PICK

Как обычно, счет ведется с нуля, таким образом, после извлечения из стека аргумента на вершине оказывается ячейка массива с номером 0, за ней – с номером 1 и так далее.

Разберем действие слова на примере. Очистите стек от имеющихся в нем данных. Затем введите

```
14 13 12 11 10 3 PICK .S
```

Как видите, мы разместили в стеке пять чисел, а в качестве аргумента указали 3. В результате вы должны получить

```
14 13 12 11 10 13 Ok
```

Таким образом, слово **PICK** копирует ячейку из глубины стека, не производя в нем других изменений.

5.2.8. Слово ROLL

Слово **ROLL** похоже на **PICK**, но в отличие от него число, при перемещении на вершину, на старом месте удаляется и ячейки, лежащие в стеке выше, как бы проваливаются вниз (см. рисунок).

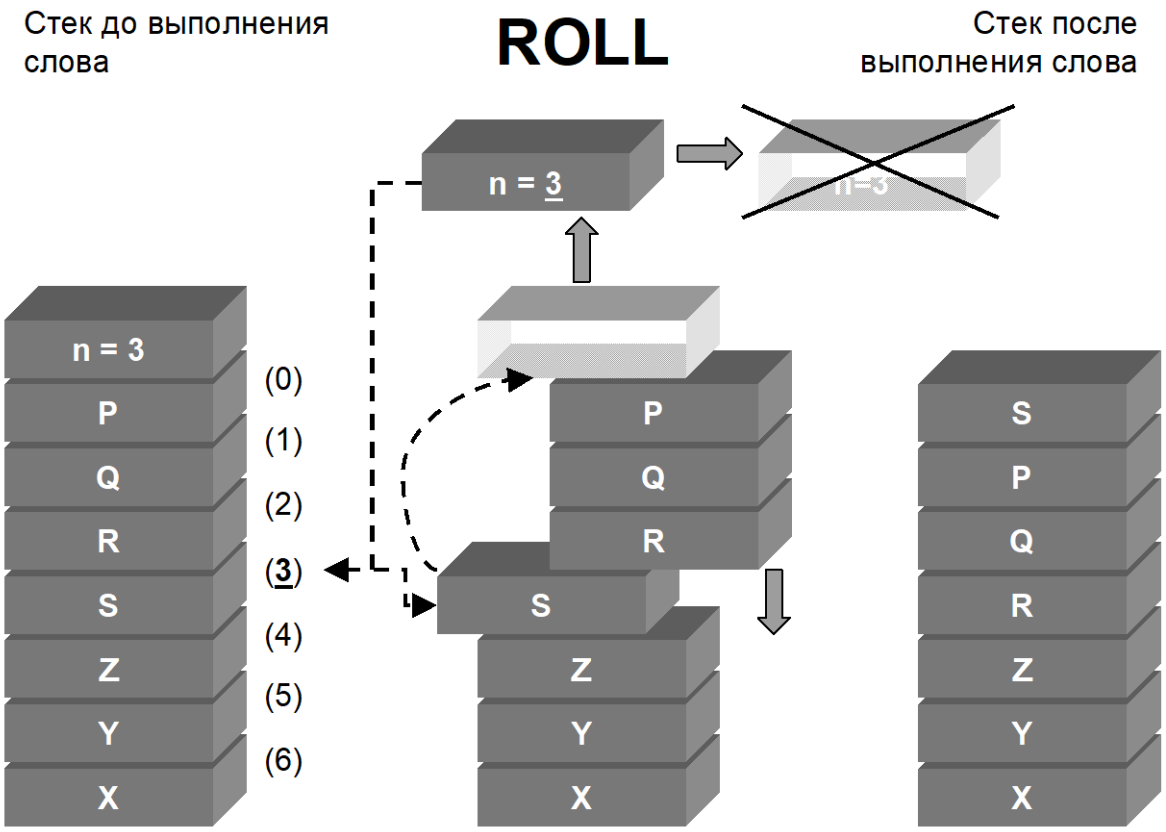


Рисунок 5.6 Действие слова ROLL

Вы можете увидеть, как это происходит, заменив в предыдущем примере слово **PICK** на слово **ROLL** :

```
14 13 12 11 10 3 ROLL .S
```

В результате вы должны получить

```
14 12 11 10 13 Ok
```

Слово **ROLL** следует применять, если невозможно использовать ничего другого, так как оно работает медленнее, чем, например, слова **DUP**, **SWAP** или **ROT**. Чтобы избежать путаницы, нужно просто держать в стеке не более четырех - пяти значений, которые используются в определении слова, и, если вы будете придерживаться этого правила, вам редко потребуется слово **ROLL**.

5.2.9. Слово DEPTH

Последнее слово, **DEPTH**, не производит никаких перестановок в стеке. Вы уже встречали его. Оно подсчитывает количество занятых ячеек в стеке и помещает это значение на вершину стека.

Если ввести

```
5 6 7 8 9 DEPTH .
```

получим

```
5 Ok
```

где 5 — это число ячеек, находившихся в стеке перед выполнением слова **DEPTH**.

5.2.10. Анализ слова .S

Теперь мы можем разобраться, как работает созданное нами слово:

```
: .S
   DEPTH ?DUP
   IF
    0 DO DEPTH 1- ROLL DUP . LOOP
   ELSE
    ." Стек пуст! "
   THEN
;
```

Слово **DEPTH** возвращает число занятых ячеек стека, и если оно не равно 0, слово **?DUP** делает его копию. Слово **IF** воспринимает 0 в стеке как флаг "ложь", а не-нулевое значение — как флаг "истина".

Здесь мы использовали еще один оператор ветвления — слово **ELSE**. Оно работает между **IF** и **THEN**, создавая альтернативную ветвь для флага "ложь".

Итак, если слово **IF** обнаружило в стеке 0, то есть стек пуст, выполняются слова между **ELSE** и **THEN** — выводится сообщение и выполнение заканчивается.

Если стек не был пуст и слово **?DUP** сделало копию числа ячеек в стеке, эта копия извлекается словом **IF**, а оригинал остается в стеке. Поскольку **IF** считает, что в стеке был флаг "истина", выполняются слова между **IF** и **ELSE**.

В стеке у нас было число использованных ячеек, сверху добавляется 0 и эти два числа извлекаются словом **DO** для инициализации цикла. Пределом цикла и будет число ячеек в стеке.

Внутри цикла еще раз выполняется слово **DEPTH**, следующее за ним слово **1-** уменьшает значение на вершине стека на 1, что дает номер самой нижней ячейки, считая от 0, как это требуется слову **ROLL**. Слово **ROLL** прокручивает стек таким образом, что последняя ячейка оказывается на вершине, а вышележащие ячейки проваливаются вниз.

Слово **DUP** делает копию только что перемещенной на верх ячейки, чтобы слово "точка" не уменьшило число ячеек в стеке. Наконец, "точка" выводит значение ячейки, извлеченной снизу.

Слово **LOOP** выполняет цикл столько раз, сколько ячеек находится в стеке. На последней итерации слово **ROLL** приводит стек в исходное состояние, а мы видим на экране распечатку стека от дна к вершине.

6. Память и данные программы

До сих пор мы говорили только об обработке данных, находящихся в стеке. Если вы уже имеете некоторый опыт программирования с использованием традиционных языков, например Бейсик или Си, такой подход покажется вам достаточно необычным. Большинство языков программирования не используют стек непосредственно, а данные хранят в переменных. Переменные представляют собой именованные ячейки памяти с фиксированными адресами.

Форт-система также предоставляет программе область памяти, в которой можно создавать переменные для хранения данных.

Как используется память при создании программы? Часть памяти должна быть выделена для хранения собственно программы, то есть последовательностей команд, реализующих алгоритмы. Эта часть называется секцией метакода Форт.

В используемой нами версии AFS код вашей программы является расширением самой AFS, поэтому вы не сможете непосредственно увидеть, где именно хранится результат вашей работы и сколько места занимает. Существуют сервисные слова системы, которые позволяют ориентироваться в использовании ресурсов, но сейчас они нам не интересны.

Вы можете свободно обращаться только с той памятью, которая выделена для хранения данных вашей программы. Эта часть памяти называется секцией данных программы.

Данные занимают непрерывную последовательность байтов, начиная с самого младшего байта адресного пространства секции данных (см. рисунок).

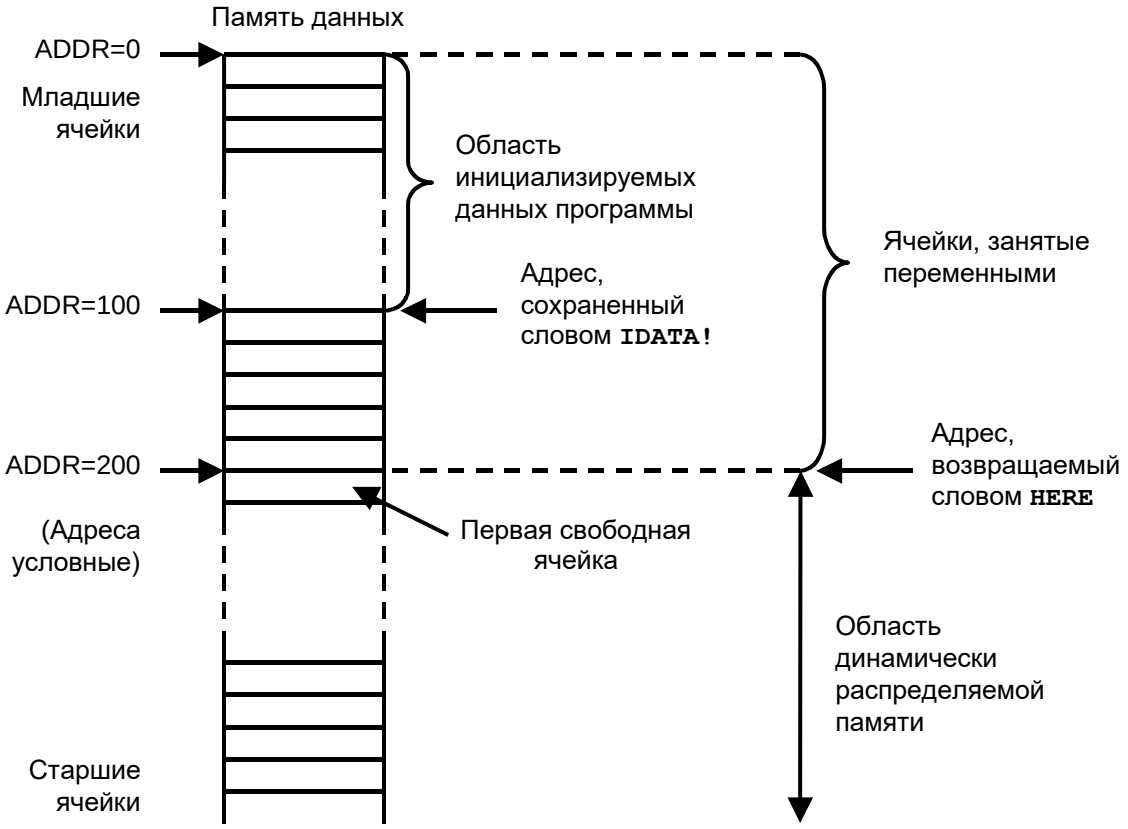


Рисунок 6.1 Секция данных программы

Какое-то количество памяти остается свободным и образует так называемую *область динамически распределяемой памяти*. Она называется так потому, что хранение данных в этой области носит временный характер и даже размер ее может меняться.

Приложение C содержит список слов для доступа к ячейкам секции данных программы, доступных в AFS уровня L0 (Таблица C.7).

6.1. Указатель данных и слово HERE

Адрес первого свободного байта области динамически распределяемой памяти постоянно хранится в системной переменной, которая называется указателем данных программы (см. Рисунок 6.1). Этот адрес, или содержимое указателя, возвращает слово HERE (-- u).

По мере создания переменных программы, они занимают байты с увеличивающимися адресами, образуя статическую область данных и отодвигая границу свободной памяти. Это движение наглядно отражает изменение значения, возвращаемого словом **HERE** .

Если в самом начале сеанса программирования вы введете

```
HERE U.
```

вы получите ноль, потому что еще не создали ни одной переменной.

В дальнейшем вы будете видеть число, которое фактически соответствует количеству байтов, уже занятых вашей программой в секции данных.

Область памяти, адрес которой возвращает **HERE**, иногда используется для временного хранения данных. Например, слово **WORD** возвращает здесь выделенное из потока ввода слово.

6.2. Создание переменных

Самый простой способ создания переменной в Форте - определение слова, возвращающего адрес ячейки памяти. Когда такое слово исполняется, то единственное, что оно должно делать, это помещать в стек адрес определенной ячейки, что обеспечивает доступ к ее содержимому с помощью других слов **.**

6.2.1. Слово @ и слово !

Для осуществления доступа к содержимому памяти по указанному адресу в Форте предусмотрено несколько слов. Наиболее важными из них являются **@** (fetch, извлечь) и **!** (store, сохранить).

Если поместить на вершину стека адрес, то слово **@** заменит адрес числом, которое хранится по этому адресу.

Вы можете посмотреть, какое значение находится в первой свободной ячейке динамической области данных:

```
HERE @ U.
```

Если на вершине стека находится адрес, а вторым элементом стека является число, то слово **!** производит запоминание этого числа по указанному адресу.

Слово **!** следует использовать осмотрительно, так как можно изменить содержимое важной части памяти, если вы запишете какое-либо число по случайно выбранному

адресу. Каждый байт памяти имеет определенное значение в работе системы. Никогда не производите запись по адресам, назначение которых вам не известно!

6.2.2. Слово CREATE

Слово **CREATE** создает в словаре Форта имя переменной и связывает это имя с некоторым адресом в секции данных программы. Когда исполняется слово, определенное словом **CREATE**, то в стек помещается адрес, связанный с именем при создании слова.

Откуда берется этот адрес? Когда **CREATE** создает новое слово, оно использует текущее значение указателя данных, то есть адрес, возвращаемый словом **HERE**.

Введите

```
CREATE var1
```

для создания слова **var1**.

Теперь введите

```
var1 .
```

и вы увидите адрес, возвращаемый словом **var1**.

Но если вы введете

```
HERE .
```

адрес будет тем же самым.

Получается, что адрес, возвращаемый словом, созданным **CREATE**, перекрывается с адресом начала динамической области данных. Хранить данные по этому адресу опасно, так как они могут быть испорчены другими словами, использующими динамическую область. Как же быть?

6.2.3. Слово ALLOT

Слово ALLOT (n --) резервирует определенное число байтов в динамической области данных. Делается это путем смещения указателя данных. Если вы введете

```
HERE .
```

и запомните значение адреса, а потом введете

```
4 ALLOT
```

и проверите новое значение указателя данных, введя

```
HERE .
```

вы обнаружите, что новое значение ровно на 4 больше предыдущего.

Мало того, ввод

```
-4 ALLOT
```

приведет к тому, что

```
HERE .
```

снова покажет исходный адрес. Получается, что с помощью слова **ALLOT** можно не только занимать байты из динамической области, но и освобождать их.

Вернемся к созданию переменных. Слово **ALLOT** можно использовать, чтобы зарезервировать нужное нам количество байтов, начиная с адреса, возвращаемого словом, созданным **CREATE**. Например,

```
CREATE var2 2 ALLOT
```

создает в словаре слово с именем **var2** и резервирует в динамической области данных программы 2 байта, в которых может быть помещено число. Если вы введете

```
123 var2 !
```

слово **var2** поместит в стек зарезервированный адрес ячейки, а **!** запишет в нее число 123.

Чтобы прочитать значение переменной, введите

```
var2 @ .
```

слово **var2** даст нам адрес, затем **@** прочитает по этому адресу число и поместит его в стек.

Простые переменные в языке Форт редко создаются таким образом. Мы использовали этот способ, чтобы продемонстрировать основные принципы работы с динамической памятью данных программы.

Обычно **CREATE** в сочетании с **ALLOT** используется для создания больших массивов и структур данных. К этому вопросу мы вернемся чуть позже.

Вы можете использовать **ALLOT** для сохранения данных, полученных словом **WORD** . Используйте **COUNT** для получения длины строки и определения количества байтов, резервируемых **ALLOT** .

Слово **ALLOT** следует использовать осмотрительно, так как можно переместить указатель данных таким образом, что будут уничтожены ранее созданные переменные.

6.2.4. Создание буфера для ввода и вывода текста

Когда мы рассматривали слова для ввода и вывода текста с использованием системной консоли, мы обещали вернуться к этой теме и поэкспериментировать. Сейчас самое время это сделать.

Для начала создадим достаточно большой буфер для приема текстовой строки:

```
CREATE TextBuf 80 ALLOT
```

Мы получили массив из 80 байтов, адрес которого возвращает слово **TextBuf** .

Как мы помним, слово **ACCEPT** должно получить в стеке адрес буфера и его длину, а вернет оно нам длину введенной строки. Попробуем ввести текст:

```
TextBuf 80 ACCEPT
```

Теперь наберите что-нибудь и завершите ввод нажатием Enter. Например, введите

```
Advanced Forth System
```

В стеке должна находиться длина введенной строки. Проверяем:

```
.
22
```

(Если вы ввели тот же текст, то все верно. Число на 1 больше количества введенных букв, так как символ перевода строки, введенный клавишей Enter, тоже сохранен и посчитан).

Теперь в буфере находится текстовая строка. Попробуем вывести её на консоль. Для этого мы используем слово **TYPE** , которому требуется адрес первого символа строки и её длина на вершине стека.

Введите

```
TextBuf 22 TYPE NL
```

На экране должно появиться

```
Advanced Forth System

Ok
```

Если результат отличается от указанного или появились сообщения об ошибках, проверьте, все ли вы делаете правильно.

Мы добавили слово **NL** после **TYPE**, чтобы Ok выводилось на новой строке. Без него последний символ строки, управляющий код возврата курсора, вернет курсор в начало строки и Ok наложится на первые символы.

6.2.5. Слова VARIABLE и 2VARIABLE

Слово **VARIABLE** применяется для определения имени переменной и резервирования под ее значение ячейки в секции данных программы. Таким образом

```
VARIABLE var5
```

производит те же действия, что и

```
CREATE var5 2 ALLOT
```

но намного удобнее.

У слова **VARIABLE** есть существенное преимущество. Во-первых, оно выравнивает указатель данных на границу ячейки перед созданием переменной, во-вторых, оно записывает ноль в новую переменную.

Применение слова **VARIABLE** в программе вместо **CREATE** делает ее более простой и понятной. С переменной, которая определена словом **VARIABLE**, обращаются так же, как с переменной, созданной словом **CREATE**.

Не менее удобно слово **2VARIABLE** , которое создает переменную из двух ячеек. Пары ячеек используются не так уж редко – это могут быть и числа двойной длины, и связанные значения (такие как, например, указатель на начало буфера в памяти и его размер).

Даже в самой простой версии AFS предоставляется минимальный набор слов для обращения с двойными ячейками.

6.2.6. Слово 2@ и слово 2!

Если есть возможность создать переменную двойной длины, должна быть и возможность сохранять ее значение и считывать при необходимости.

Слово **2@** получает на вершине стека адрес пары ячеек, а возвращает в стек число двойной длины, которое хранится по этому адресу.

Для записи двойного числа поместите его в стек, затем укажите адрес переменной. Слово **2!** производит запоминание двойного числа по указанному адресу.

Работает это так. Создаем переменную двойной длины:

```
2VARIABLE dTest
```

Сохраняем значение двойной длины:

```
123.456 dTest 2!
```

Читаем двойное число из переменной и выводим на экран:

```
dTest 2@ D.
```

Получаем

```
123456 Ok
```

При создании переменной мы использовали строчную букву d в начале имени, а само имя начали с заглавной буквы (без пробела, разумеется). Этот простой метод позволит вам не путать размеры переменных. Используйте в начале имени условные обозначения, принятые для записи стековой нотации: n, u, d, ud.

6.2.7. Слово CONSTANT

Похожим на слово **VARIABLE** является слово **CONSTANT** (константа). Оно используется для хранения таких чисел, которые внутри программы не будут изменяться.

Для него требуется число в стеке, а после слова **CONSTANT** - имя слова, в котором будет храниться это число. При исполнении созданного слова в стек помещается его содержимое (не адрес!).

Например, если определить

```
1024 CONSTANT kilobyte
```

то при исполнении слова **kilobyte** в стек будет помещено число 1024.

Слова, определенные с использованием слова **CONSTANT**, лучше всего применять для таких чисел, как коэффициенты пересчета единиц измерения или константы уравнений.

Конечно, с таким же успехом можно ввести число непосредственно в программу, но если программа подвергается изменениям, тогда каждое вхождение этого числа также должно быть изменено, в то время как при использовании константы число потребуется изменить только однократно. Кроме того, использование слова вместо числа способствует облегчению понимания программы.

Например, вы могли бы сделать нагляднее слова, выводящие управляющие последовательности ANSI на терминал, с помощью такой константы:

```
27 CONSTANT ESC
```

Слово **ESC** возвращает в стек управляющий код *Escape*.

Замечательным свойством констант является то, что они не расходуют память данных. Значение константы хранится в коде программы, благодаря чему она возвращает свое значение быстрее, чем переменная, но не может быть изменена после определения.

6.2.8. Системные константы BL, CR, LF, NUL, TRUE

Продуманное использование констант в программе позволяет сделать код компактным, а значит – разместить более сложную программу в ограниченном объеме памяти.

Например, вы хотите создать слово, издающее звонок (или другой характерный звук) на терминале. Для этого надо передать на терминал специальный символ с кодом 7 (этот символ имеет условное обозначение BEL, от bell – колокол, и ведет свое происхождение от первых телеграфных машин).

Для передачи в стек кода символа 7 вы используете число внутри определения (это называется литерал):

```
: Ring0 7 EMIT ;
```

Форт-система создает метакод, в котором литерал занимает две ячейки. Одна ячейка содержит команду, помещающую число в стек, а вторая ячейка хранит само число.

Определим константу **BEL** , хранящую число 7, а уже эту константу используем в определении:

```
7 CONSTANT BEL

: Ring BEL EMIT ;
```

В метакоде определения **Ring** для константы будет использована только одна ячейка. Если код звонка используется часто (например, при выводе сообщений об ошибках, завершении операции и т.п.), каждая замена литерала на константу экономит одну ячейку в коде программы.

Мы не учли только один момент – определение константы тоже занимает место, это две ячейки метакода.

Итого: определение константы и два использования константы занимают 4 ячейки, два литерала тоже занимают 4 ячейки. Если число используется только один раз, нет смысла заводить константу. *Если число используется более двух раз – константа экономит память.*

Некоторые числа, например, ноль, используются огромное количество раз. Для таких значений в словаре Форт уже определены системные константы. Рассмотрим их.

Слово **BL** (от blanc) помещает в стек код пробела, то есть 32 десятичное.

Слово **CR** (от cursor return) помещает в стек управляющий код возврата курсора в начало строки, то есть 13 десятичное.

Слово **LF** (от line feed) помещает в стек управляющий код перевода строки, то есть 10 десятичное.

Системная константа **NUL** помещает ноль на вершину стека. Это слово также используется в качестве флага "ложь". Далее мы всегда будем использовать **NUL** в определениях, например, для начального значения цикла.

Системная константа **TRUE** помещает на вершину стека ячейку, все биты которой установлены в 1. Это слово используется в качестве флага "истина".

В шестнадцатеричной системе счисления это число выглядит как FFFF, десятичное значение, как числа со знаком, равно –1, а как числа без знака равно 65535 (наибольшее целое значение для 16-битной ячейки).

Мы рекомендуем использовать эти константы вместо чисел.

6.2.9. Слово ALIAS

Константы с одинаковым значением в разных частях программы могут иметь совершенно разный смысл. Например, при обращении к порту ввода-вывода PB его номер

представлен как 1. В другом случае число 1 – это маска бита номер 0 в каком-либо регистре.

С помощью слова **ALIAS** вы можете создать любое количество псевдонимов одной константы, не увеличивая при этом размеров кода.

Создадим константу:

```
1 CONSTANT #1
```

и определим ее псевдонимы:

```
ALIAS MASK0 PORT-B
```

Слова **#1**, **MASK0** и **PORT-B** являются одной и той же константой, а в метакоде использовано всего 2 ячейки. Если вы удалите любое из этих слов с помощью **SMUDGE**, на оставшихся псевдонимах это никак не отразится.

Слово **ALIAS** должно выполняться сразу после определения константы. Вы можете перечислить список псевдонимов в одной строке или разместить их на нескольких строках. В последнем случае каждая строка должна начинаться словом **ALIAS**.

Количество псевдонимов для одной константы ограничено только размером памяти, отводимой системой для словаря.

В стандартном языке Форт слово ALIAS отсутствует.

Как быть, если вы хотите добавить псевдонимы к системной константе? Например, число 0 используется очень широко. Применение системной константы **NUL** не всегда наглядно, а заводить еще одну константу с тем же значением было бы не логично.

В Advanced Forth вы можете использовать слово **'** (апостроф) для нахождения любых слов в словаре. При успешном нахождении слова внутренние системные переменные устанавливаются в такое же состояние, как после создания слова, что позволяет использовать **ALIAS**.

Используя **'** (апостроф) можно задать псевдонимы для любой системной переменной:

```
' NUL ALIAS FALSE PORT-A
DROP
```

Слово **'** возвращает в стеке некое число, назначение которого нам пока не интересно, поэтому во второй строке мы используем **DROP** для очистки стека.

Другое назначение слова **'** и возвращаемого им числа мы рассмотрим позже.

6.2.10. Слово VALUE и слово TO

Если в вашей программе значение переменной используется часто, а изменяется редко, вам поможет слово **VALUE**, которое создает переменную в памяти данных, используя начальное значение, как для константы:

```
1024 VALUE kilo
```

Слово **kilo** возвращает значение 1024, но хранится оно в зарезервированной ячейке памяти данных и его можно изменить.

Для изменения значения переменной, созданной **VALUE**, используется слово **TO** :

```
1000 TO kilo
```

Теперь **kilo** будет возвращать значение 1000.

6.2.11. Автоматически инициализируемые данные. Слово IDATA!

При включении питания или перезагрузке Форт-машины по сигналу сброса все ячейки секции данных программы принимают значение 0. Это не очень удобно, если вам требуются другие начальные значения переменных.

Чтобы не инициализировать переменные нужными значениями при каждом запуске программы с помощью слов **!** и **TO**, в Advanced Forth System предусмотрена автоматическая инициализация части секции данных.

При сохранении сеанса программирования или окончательного варианта программы, фрагмент секции данных записывается в постоянную память Форт-машины. При следующем старте системы этот фрагмент восстанавливается в оперативной памяти.

Эта часть секции данных называется *областью инициализируемых данных* (см. Рисунок 6.1). Начальные значения всех переменных, попавших в эту область, автоматически загружаются системой при старте программы.

Область инициализированных данных начинается с адреса 0 секции данных. Размер области устанавливается системным словом **IDATA!** .

Слово **IDATA!** получает на вершине стека размер области инициализированных данных в байтах и сохраняет это значение в системной переменной.

При разработке программы размещайте переменные, определенные словом **VALUE**, а также другие переменные, значение которых при старте вы хотите установить автоматически, в начале секции данных. Когда все необходимые переменные определены, выполните

```
HERE IDATA!
```

Текущее значение указателя данных будет сохранено как граница области инициализированных данных.

Вы можете использовать слово **IDATA!** неограниченное число раз в процессе разработки программы. Система учитывает только последнее сохраненное значение.

Помните также о том, что при сохранении сеанса программирования или программы пользователя, в постоянную память записываются текущие значения переменных в области инициализированных данных. Если они отличаются от начальных, перед сохранением установите их в желаемое состояние.

В младших моделях Форт-машин, построенных на основе микроконтроллеров с минимальными размерами памяти, возможны ограничения при сохранении сеанса программирования, связанные с размером области инициализированных данных.

Системный параметр ROM_IDATA (стр. 128) позволяет оценить, какой размер постоянной памяти доступен в данный момент для сохранения сеанса программирования, включая область инициализированных данных.

В стандартном языке Форт слово IDATA! отсутствует, поскольку стандарт не предусматривает возможность автоматической инициализации переменных.

6.3. Операции с символами (байтами)

Время от времени требуется извлечь из памяти или записать в нее не целую ячейку, а только один байт. Чаще всего такие действия производятся при обработке информации, представленной в виде текста, так как символы как раз и размещаются в отдельных байтах.

6.3.1. Слова C@ и C!

Для операций с отдельными символами (байтами), хранящимися в памяти, в языке Форт используются два слова: C@ (addr -- char) и C! (char addr --).

Слово **C@** аналогично слову **@**, но оно извлекает значение только одного байта (буква C обозначает тип данных character, символ). Слово **C@** возвращает прочитанное значение в младшем байте ячейки на вершине стека, старшие байты ячейки обнуляются.

Слово **C!** подобно **!**, но оно производит запись только младшего байта ячейки, полученной в стеке.

Для создания переменных, состоящих из байтов, специальных слов нет. Если требуется создать переменную из одного байта или массив байтов, следует воспользоваться словами **CREATE** и **ALLOT**.

Слова **!** и **C!** следует использовать осмотрительно, так как можно изменить содержимое важной части памяти, если вы запишете какое-либо число по случайно выбранному адресу. Каждый байт памяти имеет определенное значение в работе системы. Никогда не производите запись по адресам, назначение которых вам не известно!

6.4. Операции с ячейками памяти

Как уже упоминалось, одним из обязательных требований является выравнивание адреса для чтения или записи ячейки памяти. Большинство современных процессоров требуют, чтобы адреса данных при обращении к ним были кратны размеру этих данных. Проще говоря, двухбайтные ячейки должны находиться по четным адресам, четырехбайтные — по адресам, кратным 4-м и т. д.

При этом допускается создание переменной из 1 или 3 или 5 байтов, что неизбежно приведет к тому, что указатель данных будет иметь некратное значение.

Обратите внимание на то, что слово **HERE** возвращает произвольный адрес байта (который может быть нечетным), а слова **@**, **2@**, **!** и **2!** требуют выровненного адреса.

6.4.1. Слова ALIGN и ALIGNED

Для выравнивания адреса, возвращаемого **HERE**, используется слово **ALIGN**. Оно не требует никаких параметров в стеке, но после его выполнения слово **HERE** возвращает адрес, выровненный на границу ячейки.

Слово **ALIGN** просто передвигает указатель данных вперед, если это требуется для выравнивания, резервируя нужное число байтов.

Теперь вы знаете, что при создании переменной или массива ячеек с помощью **CREATE** сначала необходимо выполнить слово **ALIGN**.

Слова **VARIABLE** и **VALUE** выравнивают адрес автоматически.

Слово ALIGNED (addr -- a-addr) похоже на **ALIGN**, но работает не с указателем данных, а с произвольным адресом. Полученный в стеке адрес заменяется на выровненное значение:

```
3 ALIGNED .
```

возвращает

```
4 Ok
```

6.4.2. Слова CELLS и CELL+

Как мы уже говорили, Форт-системы могут использовать разные размеры ячеек. Существует слово CELLS (n1 -- n2), которое получает число ячеек и возвращает число байтов в этих ячейках в соответствии с размером ячейки в системе.

Введите

```
1 CELLS .
```

и вы узнаете, сколько байтов занимает ячейка в используемой вами Форт-системе.

Это довольно удобно для создания переносимых текстов программ — не важно, в какой системе вы работаете, размеры данных будут пересчитаны корректно. К тому же, с использованием слова **CELLS** многие определения становятся нагляднее.

Например, встроенный АЦП модуля AFM требует выделения буфера для размещения результатов измерения. Каждому каналу АЦП в буфере соответствует одна ячейка.

Предположим, что мы используем для измерений 5 каналов. Если выполнить

```
ALIGN CREATE BufADC 5 CELLS ALLOT
```

будет зарезервировано место, достаточное для хранения пяти чисел, независимо от размера ячейки.

Слово **BufADC** будет возвращать адрес первого числа (точнее – ячейки, содержащей первое число).

Давайте определим слово для чтения содержимого из ячеек буфера:

```
: BufADC@ CELLS BufADC + @ ;
```

Теперь, если ввести

```
2 BufADC@ .
```

мы увидим результат измерения в канале 2 встроенного АЦП.

Постарайтесь самостоятельно разобраться, как это работает.

Иногда требуется перемещать какой-либо указатель в памяти последовательно, от ячейки к ячейке. Для этого пригодится слово CELL+ (a-addr1 -- a-addr2). Слово получает в стеке выровненный адрес и добавляет к нему размер ячейки в байтах. Полученный адрес возвращается в стек, заменяя исходный.

6.4.3. Слова , (запятая) и C, (си-запятая)

Когда вы создаете массив с помощью **CREATE**, значения ячеек не инициализируются, в них оказываются случайные значения. Можно потом записать нужные значения в каждую ячейку, но есть более удобный способ.

Создадим массив из 5 ячеек, содержащих последовательность чисел от 1 до 5:

```
ALIGN CREATE array 1 , 2 , 3 , 4 , 5 ,
```

В результате указатель данных сдвинулся на 5 ячеек, слово **array** возвращает адрес первой ячейки, а в ячейках массива лежат числа от 1 до 5.

Это легко проверить, вычисляя смещение в массиве с помощью слова CELLS (элементы массива считаем от 0):

```
2 CELLS array + @ .
```

покажет нам содержимое ячейки с индексом 2:

```
3 Ok
```

Слово **,** (запятая) берет число из стека, записывает его по адресу HERE, после чего перемещает указатель данных, резервируя ячейку.

Если вы хотите создать массив символов или небольших чисел (в диапазоне 0-255), вам потребуется слово **C,** (читается "си-запятая"). Оно выполняет то же, что и **,** (запятая), но резервирует только один байт и сохраняет в нем значение младшего байта ячейки, взятой с вершины стека. Таким образом, приведенный выше пример можно представить в таком виде:

```
CREATE c-array 1 C, 3 C, 5 C, 7 C, 9 C,
```

Если ячейки двухбайтовые, то это определение займет в памяти вдвое меньше места (и в 4 раза, если ячейки по 32 бита), но следует помнить, что числа не должны превышать 255.

6.5. Таблицы констант

Не всегда требуются массивы, значения которых изменяются в процессе выполнения работы. Весьма полезными могут быть массивы констант. Примером таких массивов служат разнообразные таблицы кодировки символов, таблицы выбора и т.п.

Если создавать такие массивы в памяти данных, их придется включать в область инициализируемых данных программы, а значит сокращать доступный для других переменных объем секции данных, увеличивать размер сохраняемой части и т.д.

В Advanced Forth предусмотрены средства конвертирования массивов, размещенных в секции данных программы, в массивы констант, помещенные в секцию метакода Форт. Такие массивы в AF называются *таблицами констант*.

6.5.1. Слова TABLE и CTABLE

Вы можете создавать таблицы с двумя вариантами констант: размером в ячейку и размером в байт.

Для создания таблицы констант, имеющих обычный размер (1 ячейка), используется слово TABLE (a-addr n --).

Слову **TABLE** необходимо передать в стеке параметров число констант в таблице (на вершине стека) и адрес массива в секции данных, который будет преобразован в таблицу.

После слова **TABLE** в строке ввода необходимо указать имя таблицы. Будет создано слово с этим именем, которое получает номер константы на вершине стека, а возвращает значение этой константы из таблицы.

Номера констант (индексы в таблице) считаются от нуля. Если указан номер, превышающий размер таблицы, возвращается значение последнего элемента. Если указать отрицательный индекс, вернется размер таблицы (число элементов).

Поясним на примере.

Создаем массив в секции данных:

```
ALIGN HERE 100 , 101 , 102 , 103 ,
```

Мы выровняли адрес и сохранили в стеке адрес начала массива. Затем ввели четыре значения, резервируя под каждое ячейку с помощью слова "запятая".

Преобразуем массив в таблицу констант:

```
4 TABLE TestTable
```

Теперь в словаре появилось слово TestTable (n – x). Слово **TestTable** получает в стеке индекс константы, а возвращает ее значение:

```
1 TestTable .
101 Ok
```

Если указать индекс больше 3 (напоминаем, что счет ведется от нуля), будет выдано значение с индексом 3:

```
5 TestTable .
103 Ok
```

Поскольку нам больше не нужен массив в секции данных, зарезервированные ячейки лучше сразу освободить:

```
4 CELLS NEGATE ALLOT
```

Мы получили количество байтов в 4 занятых ячейках, изменили знак этого числа на отрицательный и передали это значение слову **ALLOT**, которое освободило 4 ячейки, переместив указатель данных назад.

Слово CTABLE (a-addr n --) работает аналогично, но создает таблицу констант, размер которых равен байту. Обратите внимание: выравнивание адреса исходного массива здесь также обязательно!

Если при создании таблицы констант в стеке будет находится невыровненный адрес данных, будет зафиксирована системная ошибка.

Одно из самых распространенных применений **CTABLE** – таблицы кодировки символов. Один из таких примеров мы рассмотрим в разделе об использовании устройств ввода-вывода (см. стр. 151).

7. Вычисления

Еще раз напомним, что в нашей Форт-системе все числа представляют целые значения со знаком или без знака. Большинство операций предназначены для обработки данных, размер которых ограничен одной ячейкой (16 бит в нашем случае). В некоторых случаях используются пары ячеек, увеличивающих разрядность данных до 32 бит (двойные числа).

7.1. Арифметические операции

Список слов для арифметических операций с целыми числами, доступных в AFS уровня L0, содержит Приложение C (Таблица C.1).

7.1.1. Слово + и слово –

Эти два слова вы узнали еще при первом знакомстве с языком Форт. Напомним, что:

сложение выполняется словом + (n1 n2 -- n3), где n3 = n1 + n2 ;

вычитание выполняется словом - (n1 n2 -- n3), где n3 = n1 - n2.

7.1.2. Слово * и слово /

Эти слова также хорошо знакомы и не требуют дополнительных пояснений.

Умножение выполняется словом * (n1 n2 -- n3), где n3 = n1 * n2 ;

деление выполняется словом / (n1 n2 -- n3), где n3 = n1 / n2 .

7.1.3. Слова 1+ и 1-

Слова **1+** и **1–** соответственно увеличивают и уменьшают на единицу число, находящиеся на вершине стека.

Операция увеличения на 1 называется *инкрементом*. Операция уменьшения на 1 называется *декрементом*.

7.1.4. Слова 2* и 2/

Довольно часто в алгоритмах требуется деление и умножение на 2. Форт предоставляет для этих целей специальные слова, которые выполняются очень быстро.

Слово 2* (n1 -- n2) умножает число на вершине стека на 2, слово 2/ (n1 -- n2) делит число на вершине стека на 2. Для ускорения операции используется сдвиг значений на 1 бит, при этом учитывается знак числа для деления:

```
-14 2/ .
```

дает в результате

```
-7 Ok
```

7.1.5. Слово +!

Слово +! (n a-addr --) накапливает сумму в ячейке памяти данных. Слово получает адрес ячейки данных на вершине стека, значение в этой ячейке складывается с числом из второй ячейки стека и результат сохраняется в памяти данных.

Определим переменную **var0** (при создании она инициализируется значением 0):

```
VARIABLE var0
```

теперь выполним

```
111 var0 +!
```

и посмотрим, что теперь содержит переменная

```
var0 @ .
```

должно получиться

```
111 Ok
```

Добавим

```
22 var0 +!
```

и проверим содержимое переменной:

```
var0 @ .
```

Результат

```
133 Ok
```

соответствует ожидаемому.

7.1.6. Слова ABS и NEGATE

Слово **ABS** возвращает в стек абсолютную величину числа. Так, например,

```
-5 ABS
```

возвращает в стек 5, а

```
5 ABS
```

оставляет в стек то же самое число 5.

В свою очередь, слово **NEGATE** изменяет знак числа на противоположный. То есть

```
-5 NEGATE
```

поместит в стек 5, в то время как

```
5 NEGATE
```

вернет в стек -5.

7.1.7. Слова DABS и DNEGATE

Слово **DABS** возвращает в стек абсолютную величину числа двойной длины:

```
-12345.6 DABS D.
```

возвращает:

```
123456 Ok
```

Слово **DNEGATE** изменяет знак числа двойной длины на противоположный.

7.1.8. Слова S>D и U>D

Для преобразования чисел, занимающих одну ячейку, в числа двойной длины в языке Форт используются слова **S>D** и **U>D**. Первое слово выполняет преобразование с учетом знака числа, второе – без знака (см. рисунок).

Слово **U>D** просто добавляет нулевую старшую ячейку, располагая двойное число в стеке в соответствии с правилами Advanced Forth (младшая ячейка на вершине, старшая под ней).

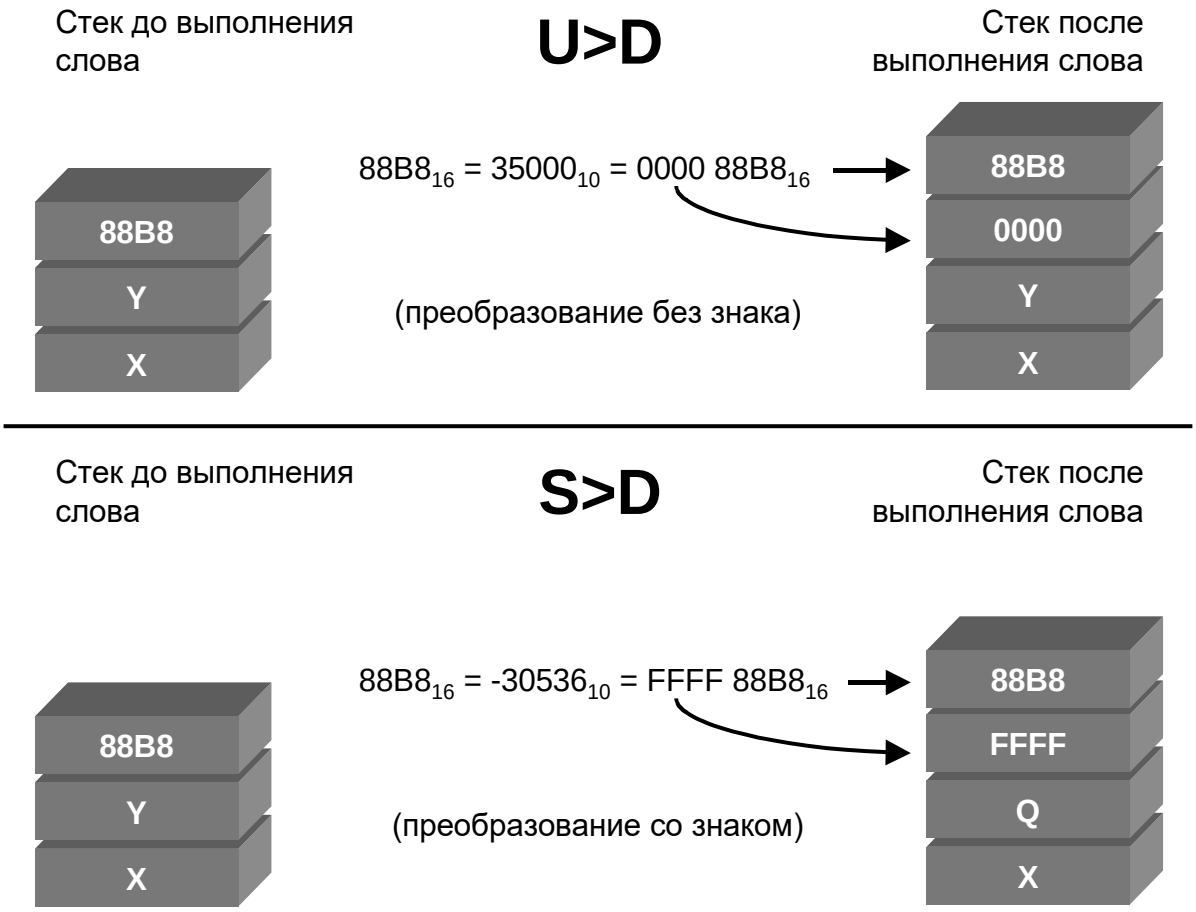


Рисунок 7.1 Преобразование в число двойной длины

Слово **S>D** учитывает знак числа. Знаковый бит расширяется на старшую ячейку двойного числа, что дает число удвоенной разрядности с тем же значением и знаком.

Основное назначение этих слов – преобразование обычных целых чисел для использования со словами, требующими наличия в стеке числа двойной длины.

Кроме того, использование этих слов позволяет скрыть разницу между Advanced Forth и стандартным Форт в порядке хранения чисел двойной длины в стеке параметров.

7.2. Умножение и деление с удвоенной разрядностью

Отдельно стоит рассмотреть группу специальных слов Форта, позволяющих избежать проблем при умножении и делении.

Как вы помните, в 16-битных ячейках может хранится число без знака в диапазоне от 0 до 65535 или число со знаком в диапазоне от –32768 до 32767.

Предположим, что вам необходимо умножить число 27000 на 11 и разделить произведение на 9, при этом должно получиться 33000. Очевидно, что все перечисленные числа могут быть размещены в одной ячейке (без знака).

Попробуйте выполнить эти операции, введя

```
27000 11 * 9 / U.
```

Результат, который вы увидите, точно не будет равен 33000 (что именно выдаст система зависит от используемой вами версии программного обеспечения и аппаратуры).

При умножении 27000 на 11 результат равен 297000, что больше 65535 и в одну ячейку не помещается. Происходит так называемое переполнение. Заведомо неверный результат умножения делится на 9. Но эту задачу можно решить иначе.

Введите

```
27000 9 / 11 * U.
```

и вы получите

```
33000 Ok
```

Переставив числа и операторы, вы получили правильный результат. Но ведь не всегда же можно узнать, будут ли расположены большие и малые числа в правильной последовательности. К тому же, при выполнении деления первым, могут быть потеряны младшие биты результата.

7.2.1. Слово */

Слово ***/** разрешает проблему переполнения, запоминая промежуточный результат в виде двойного числа, а не в одной ячейке, как обычно. Следовательно, при 16-разрядных ячейках допускается значение произведения до 4271406735. Таким образом устраняется возможность переполнения при умножении.

Вернемся к нашему примеру, но используем слово ***/**:

```
27000 11 9 */ U.
```

получаем

```
33000 Ok
```

Итак, слово */ (n1 n2 n3 -- n4) умножает третье число в стеке (n1) на второе (n2), сохраняя произведение в виде числа двойной длины, которое делит на число с вершины стека (n3). Результат (n4) возвращается на вершине стека.

7.2.2. Слово M* и слово UM*

Если вам надо просто перемножить два числа без риска переполнения, воспользуйтесь словом M* (n1 n2 -- d). Буква M здесь от mixed, имеется в виду умножение со "смешенной" разрядностью.

Слово **M*** получает в стеке два числа со знаком, каждое размером в одну ячейку, перемножает их и возвращает результат в виде числа двойной длины со знаком. Как мы помним, числа двойной длины занимают две ячейки в стеке.

Введите

```
-3000 200 M* D.
```

и получите

```
-600000 Ok
```

Если вы используете числа без знака, то вам потребуется слово UM* (u1 u2 – ud). Слово **UM*** перемножает два числа без знака, каждое размером в одну ячейку, и возвращает результат в двойном числе без знака:

```
50000 20 UM* UD.
```

Результат:

```
1000000 Ok
```

7.2.3. Слова MOD, /MOD, */MOD и UM/MOD

При целочисленных операциях сложения, вычитания и умножения результат представляет собой целое число. Теперь посмотрим, что происходит при делении. Например, если 3 поделить на 2, должно получиться 1.5, то есть 1 и 5/10.

Очевидно, что нередко будут возникать ситуации, когда нужно каким-то образом определить остаток от деления (который также называется модулем деления). Остаток находится с помощью слова **MOD** (от module).

Если вы введете

```
23 7 MOD .
```

то получите остаток от деления 23 на 7:

```
2 Ok
```

Слово MOD (n1 n2 -- n3) определяет остаток, или модуль, от деления второго числа в стеке (n1) на число с вершины стека (n2). Результат (n3), как обычно, помещается на вершину стека.

Для большей гибкости в языке Форт есть и другие слова, которые дают в результате остаток.

Слово /MOD (n1 n2 -- n3 n4) помещает в стек остаток (n3) от деления двух чисел и частное (n4 = n1/n2), которое помещается поверх остатка.

Таким образом,

```
5 2 /MOD . .
```

дает в результате пару чисел

```
2 1 Ok
```

Слово */MOD (n1 n2 n3 -- n4 n5) представляет полезный гибрид ***/** и **MOD**.

Введите

```
20000 5 7 */MOD . .
```

и вы получите результат

```
14285 5 Ok
```

Второе (n2) и третье (n1) число в стеке (в данном случае 5 и 20000) перемножаются, образуя результат двойного размера, чтобы избежать переполнения. Затем это произведение делится на число, находящееся на вершине стека (n1). Остаток от деления (n4) сохраняется в стеке, поверх него на вершину стека помещается частное (n5).

Перечисленные слова выполняют операцию деления с учетом знака чисел.

Если вы работаете с числами без знака, используйте слово UM/MOD (ud u1 -- u2 u3) для "смешенного" деления чисел без знака.

Слово **UM/MOD** получает на вершине стека делитель размером в одну ячейку (u1), ниже в стеке находится делимое в виде двойного числа (ud). Слово возвращает частное без знака на вершине стека (u3) и остаток во второй ячейке (u2).

7.3. Логические операции

В этом разделе мы будем использовать преимущественно двоичные числа, поэтому необходимо установить двоичную систему счисления, выполнив слово **BIN** .

Теперь вы сможете вводить только числа, составленные из нулей и единиц, то есть двоичные числа, а также видеть содержимое отдельных битов в том виде, как они хранятся в памяти компьютера.

Кроме того, в приводимых примерах будет использовано слово **U.** (U с точкой), здесь оно необходимо для вывода двоичных чисел в "чистом" виде, без учета особенностей представления знака числа.

Список слов для выполнения логических операций, доступных в AFS уровня L0, представлен в Приложении C (Таблица C.2).

7.3.1. Слова AND, OR, XOR и NOT

Представим себе, что в стеке находится двоичное число 10011101 и вам необходимо изменить третий справа разряд, то есть вы хотите превратить это число в 10011001. Нагляднее всего проследить за изменениями отдельных битов, если при вводе поместить числа одно под другим.

Введите:

```
10011101
```

```
11111011 AND U.
```

и вы получите

```
10011001 Ok
```

Слово **AND** (И) сравнивает поразрядно два числа, и если в обоих (одном и другом) эти разряды установлены в 1, то результат будет 1, иначе результатом будет 0. Слово **AND** относится к так называемым *логическим операторам*, список которых вы видите в таблице. Для каждого оператора описывается, какой результат получается при сравнении двух битов. (Такие описания называются *таблицами истинности*.)

Логические операторы часто обрабатывают значения, представляющие понятия "истина" и "ложь", пришедшие в вычислительную технику из формальной логики. С точки зрения логики, бит, равный нулю представляет значение "ложь", а единица – "истина". Мы еще вернемся к этой теме, обсуждая операторы сравнения и ветвления.

Таблица 7.1 Логические операторы

Первое значение	Второе значение	Результат
AND (логическое И)		
0	0	0
0	1	0
1	0	0
1	1	1
OR (логическое ИЛИ)		
0	0	0
0	1	1
1	0	1
1	1	1
XOR (исключающее ИЛИ)		
0	0	0
0	1	1
1	0	1
1	1	0
NOT (логическое НЕ, инверсия)		
0		1
1		0

Второе число (11111011 в нашем примере) иногда называют *битовой маской*. Битовая маска – это двоичное число, которое логический оператор применяет к другому числу с целью изменить значение отдельных битов.

Как следует из таблицы истинности для оператора **AND** (И) единица в результате получается только в том случае, если первое значение равно 1 *И* второе значение также равно 1. Из той же таблицы следует, что если требуется изменить значение 0 на значение 1, оператор **AND** (И) не сможет этого сделать.

Если вы внимательно изучите таблицы истинности, вы поймете, что для этого необходимо использовать оператор **OR** (ИЛИ), который выдает в результате 1, если первое значение равно 1 *ИЛИ* второе значение равно 1.

Другими словами, слово **OR** также производит поразрядное сравнение двух чисел, но, если хотя бы один из разрядов установлен в 1, то в результате тоже будет 1.

Попробуем установить в 1 третий разряд в последнем примере. Для этого случая мы используем битовую маску 00000100:

```
10011001

00000100 OR U.
```

получим

```
10011101 Ok
```

Третий логический оператор **XOR** (исключающее ИЛИ) также производит поразрядное сравнение. Если два разряда различны, то в бите результата устанавливается 1, если одинаковы, то результат будет 0.

Предположим, требуется изменить значения трех младших разрядов числа на противоположные, то есть преобразовать число 10011101 в 10011010. Используем для этого слово **XOR** .

Введите

```
10011101

00000111 XOR U.
```

и получите

```
10011010 Ok
```

Так как в маске три младших разряда были установлены в 1, то там, где в исходном числе была 1, в результате стал 0, а если был 0, стала 1.

Последний оператор, **NOT** (НЕ), не требует битовой маски. Слово **NOT** изменяет значение всех разрядов на противоположное. Таким образом, если вы введете

```
01100110 NOT U.
```

результат будет

```
1111111110011001 Ok
```

(В ячейке 16 битов, поэтому после инверсии мы видим их все).

Оператор **NOT** (НЕ) также называют *оператором инверсии*. Когда значение двоичного разряда меняется на противоположное, говорят, что разряд был *проинвертирован*. Противоположные логические значения называют *инверсиями*, например значение "ложь" есть инверсия значения "истина".

7.3.2. Слово CLR

Следующее слово не является логическим оператором, но выполняет похожие действия с битами. Слово CLR (n1 n2 -- n3) обнуляет биты первого числа, если во втором числе соответствующие биты равны 1.

Введите

```
11011011
```

```
00011100 CLR U.
```

и получите результат

```
11000011
```

7.3.3. Слова <<< и >>>

Сдвиг - одна из важнейших операций с битами. Сдвиг битов в двоичных числах позволяет быстро вычислять адреса по индексам, производить умножение на степень двойки, манипулировать малыми числами и делать массу других полезных вещей.

Во всех вычислительных машинах битовый сдвиг производится на аппаратном уровне, поэтому скорость его выполнения очень велика. Фактически, слова Форта **2*** и **2/** используют машинные команды сдвига числа на один бит.

Если все биты числа сдвинуть влево на один бит, получится значение вдвое большее исходного. Сдвиг вправо на один бит дает вдвое меньшее число. Сдвиг на два бита уменьшает или увеличивает число в четыре раза, сдвиг на три бита – в восемь раз и так далее.

Слово <<< (x1 n1 -- x2) выполняет сдвиг влево, а слово >>> (x1 n1 -- x2) – вправо. Оба слова используют число на вершине стека как величину сдвига второго числа.

Посмотрим, как это работает.

Введите

```
11100111 11 <<< U.
```

и вы увидите

```
11100111000 Ok
```

Мы все еще работаем в двоичной системе, поэтому ввели величину сдвига как двоичное число 11, то есть десятичное 3.

То же самое, но сдвигаем вправо:

```
11100111 11 >>> U.
```

получаем

```
11100 Ok
```

Теперь попробуем в десятичной системе:

```
DECIMAL
```

```
7 3 <<< U.
```

результат

```
56 Ok
```

(мы умножили 7 на 8, то есть $7 * 2^3$).

Аналогично делим на 2^3:

```
1024 3 >>> U.
```

дает в результате

```
128
```

В стандартном языке Форт словам <<< и >>> соответствуют слова LSHIFT и RSHIFT. Операции сдвига используются довольно часто, поэтому в Advanced Forth было решено использовать более короткий и наглядный вариант записи.

8. Управление выполнением программы

Одно из наиболее важных свойств любого языка программирования - возможность выполнения операций на основании истинности или ложности некоторых условий. Например, если два верхних элемента в стеке равны, должно быть выполнено одно действие, но если они не равны, то должно быть сделано что-то другое. Такое условное выполнение вам встречалось в ознакомительном примере – мы использовали слова **IF** и **THEN** для ограничения длины строки:

```
: limit79  DUP 79 > IF DROP 79 THEN ;
```

Подобные конструкции, которые управляют порядком выполнения программы, называются *управляющими структурами*. К управляющим структурам относятся конструкция IF-THEN, счетные циклы и другие средства для осуществления переходов в программе.

Как вы помните, в ознакомительном разделе мы обсуждали операторы сравнения и ветвления как независимые абстракции языка. В этом заключается еще одно отличие языка Форт от других языков программирования: в большинстве языков высокого уровня операторы сравнения являются частью управляющих структур и не могут использоваться в программе самостоятельно.

8.1. Операторы сравнения

В языке Форт операторы сравнения являются самостоятельными словами, которые получают в стеке значения для сравнения и возвращают результат в виде числа на вершине стека. Это число называется *логическим флагом* или просто *флагом*.

Список операторов сравнения, доступных в AFS уровня L0, содержит Приложение C (Таблица C.3).

8.1.1. Логические флаги

Логический флаг может принимать только два значения, "истина" и "ложь". Значение "истина" устанавливается, если условие, задаваемое оператором сравнения, выполняется, в противном случае флаг устанавливается в значение "ложь".

Мы знаем, что в стеке хранятся только целые числа, значит, какие-то из них должны соответствовать понятиям "истина" и "ложь". Действительно, язык Форт использует ноль для значения флага "ложь" и число –1 (минус один) для флага "истина".

То, что значение флага "истина" равно -1, а не 1 или другому числу, имеет определенное основание. Если вы помните, -1 в двоичном виде содержит 1 во всех разрядах ячейки, независимо от ее размера. При обнаружении истинности условия число, содержащее во всех разрядах единицы, оказывается более удобным для использования его с логическими операторами.

Например, требуется заменить число нулем, если флаг имеет значение "ложь", и оставить без изменения, если флаг имеет значение "истина". Если число и флаг находятся в стеке, то оператор AND ("И") легко выполнит эту задачу.

Хотя подобные случаи не так уж часты, программа может дать выигрыш по времени, пользуясь этой особенностью флага "истина". Вы можете использовать результаты проверки различных условий и комбинировать их с помощью логических операторов, получая в результате стандартный флаг.

Для наглядности и оптимизации кода программы Форт предоставляет константу **TRUE** , значение которой соответствует флагу "истина". Флагу "ложь" соответствует системная константа **NUL** .

8.1.2. Слова MAX и MIN

Слова **MAX** и **MIN** не являются полноценными операторами сравнения, поскольку не возвращают логический флаг. Эти слова сравнивают два числа в стеке и оставляют в стеке соответственно большее или меньшее из них.

Так, например,

```
7 3 MAX
```

оставляет в стеке 7, в то время как

```
7 3 MIN
```

возвращает в стек 3.

Одно из наиболее частых применений этих двух слов - ограничение диапазона чисел, поступающих на входе. Вспомните определение слова, ограничивающего число выводимых на экран "звездочек" из вашей первой программы:

```
: limit79 DUP 79 > IF DROP 79 THEN ;
```

Используя слово **MIN** можно сделать определение коротким и более быстрым:

```
: limit79 79 MIN ;
```

Как вы помните, перед выполнением слова **limit79** на вершине стека находится длина строки "звездочек". В новом определении в стек помещается число 79 и эти два числа сравниваются словом **MIN**. Оба числа извлекаются из стека, а обратно возвращается только меньшее. Таким образом, если заданное число "звездочек" меньше 79, слово **MIN** вернет в стек его, а если больше – заменит на 79.

8.1.3. Основные операторы сравнения. Слова <>, =, < и >

Операторы сравнения извлекают два числа из стека, сравнивают их и возвращают в стек логический флаг, соответствующий результату сравнения.

Слово **<>** (не равно) возвращает флаг "истина", если два числа в стеке не равны.

Слово **=** (равно) возвращает флаг "истина", если два числа в стеке равны.

Слово **<** (меньше) возвращает флаг "истина" если второе число в стеке меньше числа на вершине, а слово **>** (больше) если второе число в стеке больше числа на вершине. Обратите внимание: числа сравниваются с учетом их знака.

Вы можете самостоятельно разобраться с этими словами, вводя пары чисел и проверяя результат. Например, если вы введете

```
1234 1234 = .
```

то увидите

```
-1 Ok
```

то есть флаг "истина", поскольку в стек были введены равные значения.

Напротив, ввод

```
123 321 = .
```

дает

```
0 Ok
```

что тоже правильно, так как ноль является флагом "ложь" а числа в стеке не равны.

8.1.4. Сравнение чисел без знака. Слова U< и U>

Сравнение чисел без знака имеет некоторые особенности, поэтому для него предусмотрены специальные операторы **U<** и **U>**.

Если ввести

```
60000 20000 < .
```

то вы получите

```
-1 Ok
```

Другими словами, в стек будет возвращено значение истина, потому что слово **<** воспринимает 60000 как отрицательное число.

Операторы **U<** и **U>** применяются для сравнения чисел без знака, чтобы они не рассматривались как отрицательные.

Если вы используете слово **U<** в предыдущем примере

```
60000 20000 U< .
```

вы получите правильный результат

```
0 Ok
```

то есть флаг "ложь".

8.1.5. Сравнение чисел с 0. Слова 0< , 0<> , 0= и 0>

Если требуется установить, равно ли число в стеке нулю, это можно сделать, поместив в стек 0 и выполнив оператор сравнения:

```
123 0 = .
```

Сравнение чисел с нулем выполняется в программах очень часто, поэтому в языке Форт для таких случаев предусмотрены специальные слова, позволяющие увеличить производительность.

Слова **0<** , **0<>** , **0=** и **0>** требуют наличия в стеке только одного числа.

Слово **0=** возвращает флаг "истина", если число равно 0, а слово **0<>** - если не равно 0.

Слово **0<** возвращает флаг "истина" если число меньше 0, то есть отрицательное. Слово **0>** возвращает флаг "истина" если число больше 0, то есть положительное. Обратите внимание: оба слова вернут флаг "ложь", если число равно 0.

8.2. Операторы ветвления

Операторы сравнения были бы бесполезны без других слов, реагирующих на значение флага, возвращаемого в стеке. К таким словам относятся уже известные вам **IF** и **THEN**, их называют *операторами ветвления* или *операторами условного выполнения*, поскольку выбор дальнейшего хода выполнения программы определяется некоторым условием.

8.2.1. Слова IF, ELSE и THEN

Управляющая структура ветвления программы состоит из трех слов: IF...ELSE...THEN.

Слово **ELSE** не является обязательным и часто опускается. Если оно отсутствует, то остается только одна возможность: исполнить слова, находящиеся между **IF** и **THEN**, если оператор **IF** обнаруживает в стеке флаг "истина".

У слова **IF** есть одно замечательное свойство: оно считает флагом "истина" любое значение, отличное от нуля (ноль всегда остается флагом "ложь"). Это очень удобно в ряде случаев, например, при работе с устройствами ввода-вывода.

Предположим, в стеке передано состояние дискретных входов порта ввода-вывода. К входам могут быть подключены датчики или кнопки. Каждому биту ячейки соответствует состояние одного входа порта.

Нам нужно выполнять определенные действия, только если один из сигналов принимает состояние логической 1 . Пусть этому входу соответствует бит номер 0 (самый младший), тогда мы легко получаем логический флаг, соответствующий его состоянию, с помощью слова **AND** :

```
1 AND IF … THEN
```

Если нулевой бит числа в стеке равен 1, слово IF получит в стеке значение 1, что приведет к выполнению слов между **IF** и **THEN** .

Слово **ELSE** открывает альтернативную ветвь выполнения программы. Посмотрите на рисунок.

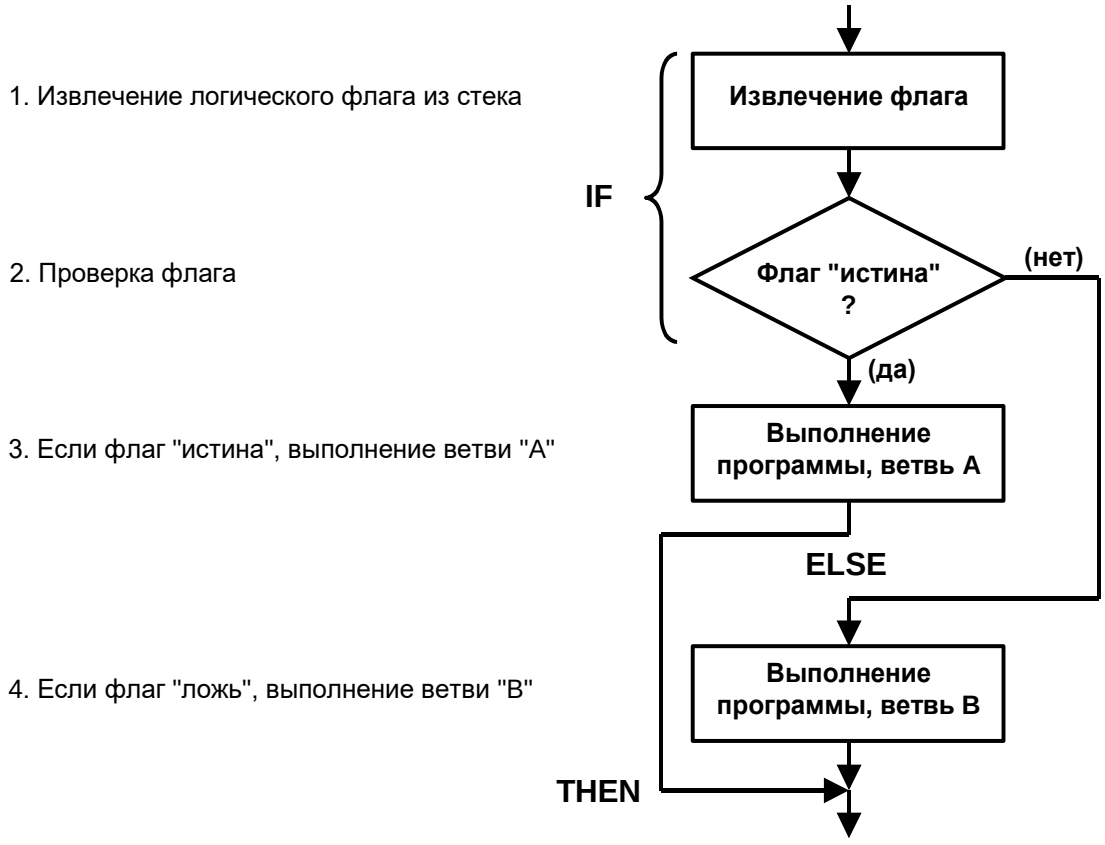


Рисунок 8.1 Действие структуры IF…ELSE…THEN

Если флаг, принятый **IF** , имеет значение "истина", выполняются слова, находящиеся между **IF** и **ELSE**. Если флаг имеет значение "ложь", выполняются слова, заключенные между **ELSE** и **THEN**. В обоих случаях исполнение продолжается после слова **THEN**.

8.3. Циклы с неопределенным числом повторений

В языке Форт, как и в других языках программирования, существует возможность повторения выполнения некоторого фрагмента кода. Мы уже сталкивались с циклическим повторением операций при создании нашей первой программы.

8.3.1. Слова BEGIN и UNTIL

С помощью структуры, очень похожей на IF…THEN можно создать "логическую петлю" внутри программы, или цикл с неопределенным числом повторений.

Слово **BEGIN** отмечает начало цикла (или "петли"). Его назначение сходно со словом **THEN** .

Слово **UNTIL** напоминает слово **IF** – оно так же извлекает из стека логический флаг, проверяет его и принимает решение о переходе к точке программы, отмеченной словом **BEGIN** .

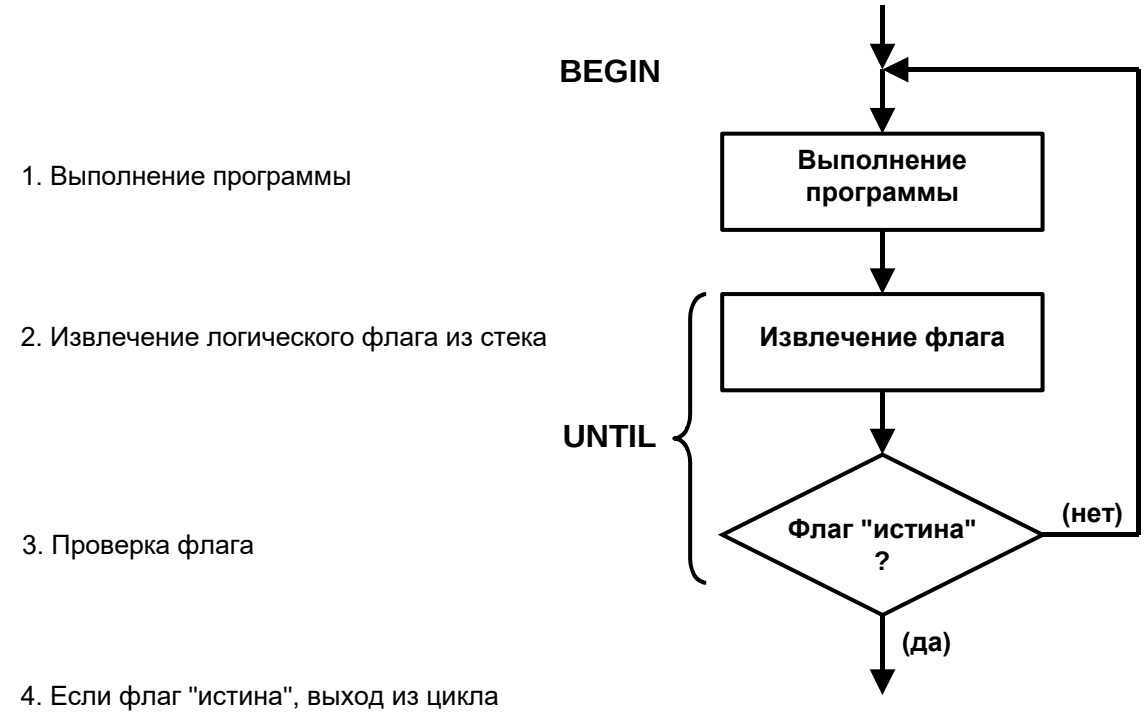


Рисунок 8.2 Цикл BEGIN…UNTIL

Слова, расположенные между **BEGIN** и **UNTIL** выполняются как минимум один раз. Перед выполнением слова **UNTIL** в стеке должно оказаться значение, определяющее дальнейший ход программы. Если **UNTIL** извлекает из стека ноль (то есть флаг "ложь"), выполнение программы повторяется с точки, отмеченной словом **BEGIN** .

Как и слово **IF**, слово **UNTIL** считает флагом "истина" любое число, не равное нулю.

Если **UNTIL** обнаруживает в стеке такое значение, выполнение программы продолжается словами, следующими после слова **UNTIL** .

Давайте проведем небольшой эксперимент.

Создадим новое слово:

```
: Test1 BEGIN KEY DUP EMIT CR = UNTIL ;
```

Слово **BEGIN** отмечает начало цикла. Следующее слово **KEY** ожидает нажатия клавиши на терминале. Полученный код символа дублируется словом **DUP** – теперь в стеке два одинаковых значения. Верхнее значение извлекается словом **EMIT** и передается на терминал – мы видим символ на экране.

Константа **CR** помещает в стек управляющий код возврата курсора, он же – код клавиши Enter. В стеке уже лежит копия введенного символа, эти два числа сравниваются словом **=** . Если коды равны (была нажата клавиша Enter), вместо них в стеке остается флаг "истина", если не равны (любая другая клавиша) – возвращается флаг "ложь".

Слово **UNTIL** извлекает флаг с вершины стека. Если это "ложь", выполнение повторяется со слова, следующего за **BEGIN**, если "истина" – выполнение слова **Test1** завершается.

Таким образом, слово **Test1** показывает на экране вводимые символы, пока не будет нажата клавиша Enter.

8.3.2. Слова WHILE и REPEAT

Более сложный и гибкий тип цикла можно получить с помощью слов **WHILE** и **REPEAT** .

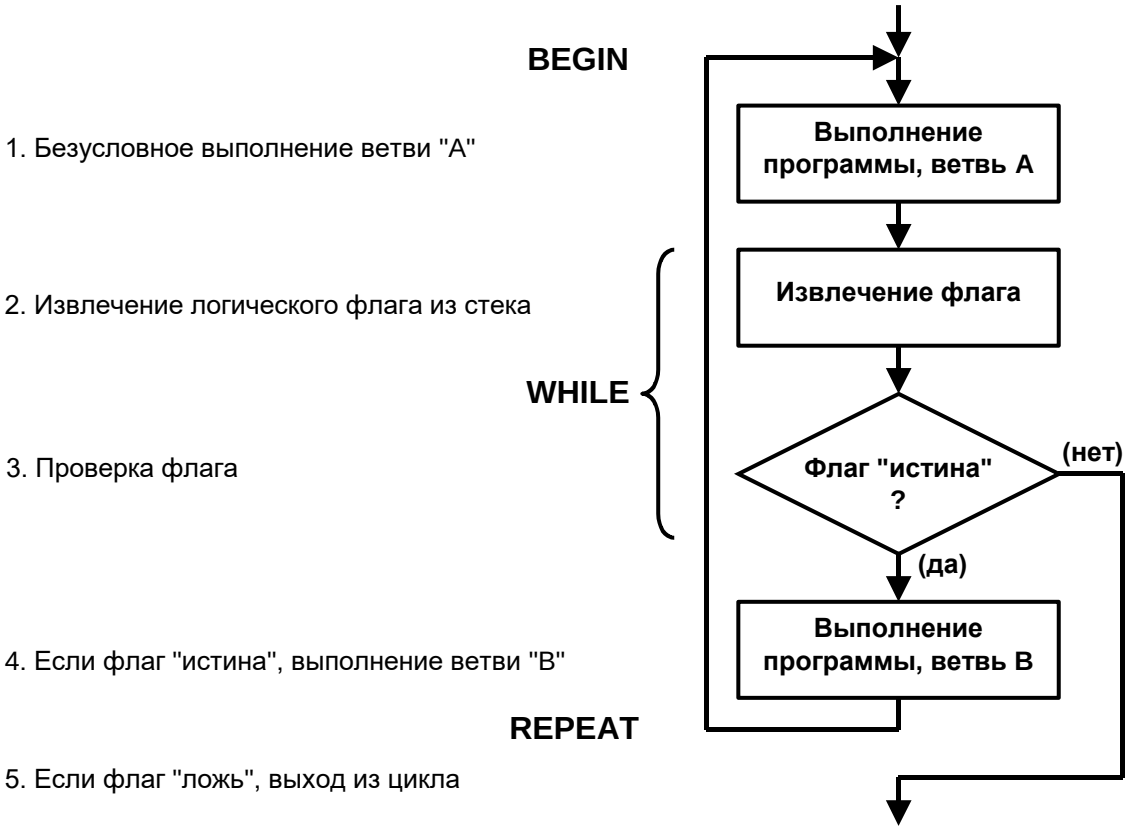


Рисунок 8.3 Цикл WHILE…REPEAT

Посмотрите на Рисунок 8.3. Как и в предыдущем варианте, слово **BEGIN** отмечает начало цикла.

Слова, расположенные между **BEGIN** и **WHILE** выполняются как минимум один раз. Перед выполнением слова **WHILE** в стеке должно оказаться значение, определяющее дальнейший ход программы.

Если **WHILE** обнаруживает в стеке значение, не равное нулю (флаг "истина"), выполняются слова между **WHILE** и **REPEAT** , после чего слово **REPEAT** возвращает выполнение программы в точку, отмеченную словом **BEGIN** и цикл начинается снова.

Если **WHILE** извлекает из стека ноль (то есть флаг "ложь"), выполнение программы повторяется с точки, следующей за словом **REPEAT** , происходит выход из цикла.

Внесем изменения в наше экспериментальное слово, принимающее символы с клавиатуры и показывающее их на экране.

Пусть оно работает до тех пор, пока мы не нажмем клавишу Escape. Код клавиши Escape (десятичное 27) является началом управляющей последовательности для терминала, поэтому выводиться он не должен.

Создадим новое слово:

```
: Test2 BEGIN KEY DUP ESC = NOT WHILE EMIT REPEAT DROP ;
```

Слово **BEGIN** отмечает начало цикла. Следующее слово **KEY** ожидает нажатия клавиши на клавиатуре. Полученный код символа дублируется словом **DUP** – теперь в стеке два одинаковых значения.

Верхнее значение мы сравниваем с кодом клавиши Escape, который возвращает константа **ESC** (стр. 81), и получаем флаг "истина", если была нажата клавиша Escape.

Слово **NOT** инвертирует флаг на вершине стека – теперь он будет "ложь", если нажата клавиша ESC и "истина" для всех остальных символов.

Слово **WHILE** извлекает флаг с вершины стека. Если это "истина", выполняется слово **EMIT**, которое берет копию кода символа из стека и выводит его на экран. Следующее за ним слово **REPEAT** возвращает выполнение к началу цикла, отмеченному **BEGIN** .

Если **WHILE** получает флаг "ложь", выполнение продолжается со слова, следующего за **REPEAT** – это слово **DROP**, которое удаляет копию кода ESC из стека, после чего выполнение слова **Test2** завершается.

Выполните слово **Test2** и убедитесь, что все работает правильно.

8.3.3. Бесконечный цикл. Слово AGAIN

Самый простой вид цикла – бесконечный цикл, организуемый с помощью слова **AGAIN**.

Как всегда, слово **BEGIN** используется для обозначения начала цикла. За ним следуют слова, которые мы будем выполнять снова и снова. Слово **AGAIN** направляет выполнение в точку, отмеченную словом **BEGIN**, не проверяя каких-либо условий.

Продемонстрируем выполнение бесконечного цикла на примере уже известного нам слова, выводящего на экран предложение "Hello, World!".

Для начала определим слово Hello:

```
: Hello ." Hello, World!" NL ;
```

А теперь создадим главное слово нашей программы:

```
: Main BEGIN Hello 1000 MS AGAIN ;
```

В слове **Main** мы впервые использовали системное слово MS (u --). Слово **MS** приостанавливает выполнение программы на время, выраженное в миллисекундах, которое получает на вершине стека как число без знака. Мы указали 1000 мс, что равно 1 секунде.

Если теперь выполнить слово **Main**, вы увидите на экране бесконечно повторяющееся сообщение

Hello, World!

обновляющееся каждую секунду.

Чтобы вернуться в новый сеанс программирования вам придется выполнить сброс или отключить питание модуля AFM.

8.4. Стек возвратов

Мы уже упоминали, что в Форт-системе фактически имеются несколько стеков. Один из них вы уже знаете - стек параметров.

Второй стек называется *стек возвратов*. Его основное назначение состоит в слежении за порядком выполнения слов.

Стек возвратов также можно использовать для временного хранения чисел. В частности, в стеке возвратов сохраняются предел и текущее значения индекса цикла.

Диаграмма вида (R: -- x) отражает состояние стека возвратов до и после выполнения слова, аналогично тому, как мы показывали состояние стека параметров.

Если слово использует и стек возвратов, и стек параметров, указывается состояние обоих стеков. При этом диаграмма состояния стека параметров отмечается знаком S: .

Например, диаграмма (S: n --) (R: -- n) показывает, что при выполнении слово извлекает число из стека параметров и помещает его в стек возвратов.

Как стек возвратов используется для выполнения слов?

Давайте вспомним наше первое знакомство с Форт-системой.

Мы определили слово, возводящее в квадрат число с вершины стека:

```
: square DUP * ;
```

Затем, используя это слово, мы определили слово для возведения числа в куб:

```
: cube DUP square * ;
```

Теперь мы можем поместить число в стек и получить его куб:

```
3 cube
```

Давайте представим, что происходит при выполнении этой простой задачи. Мы изобразили на рисунке схему прохождения процесса выполнения последней введенной строки в Форт-системе.

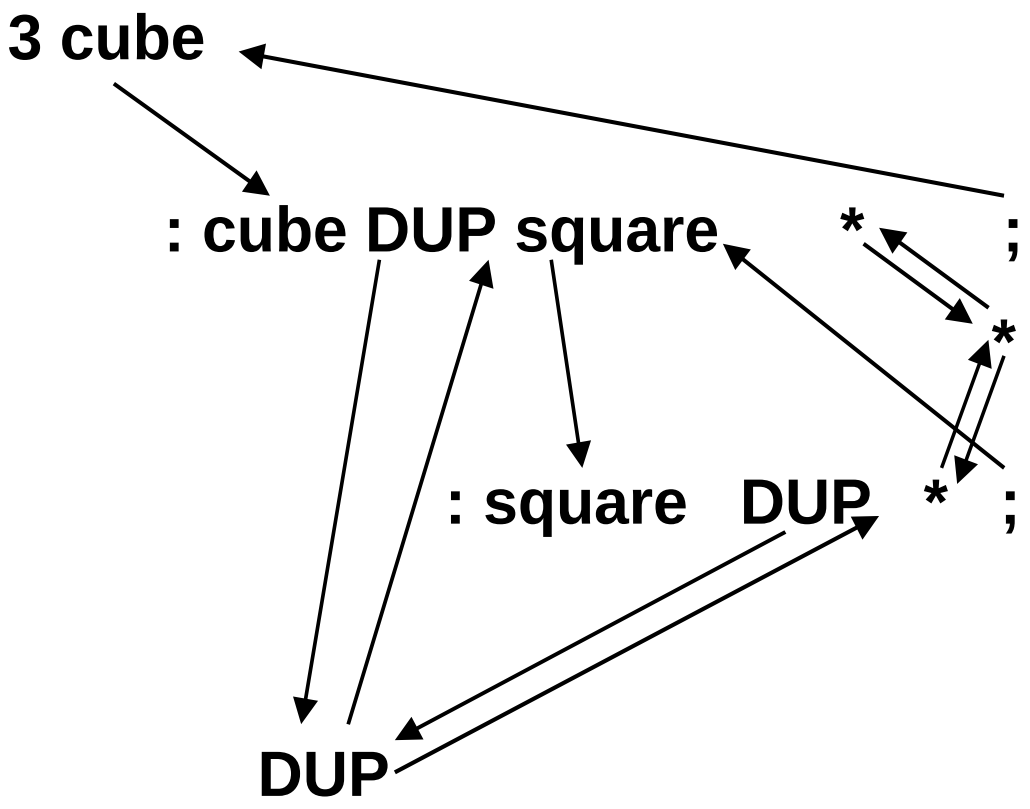


Рисунок 8.4 Процесс выполнения слов Форт-системой

Интерпретатор ввода сначала помещает в стек число 3, а затем обнаруживает, что необходимо выполнить ранее определенное слово **cube** . Для его выполнения находится запись в словаре, из которой выясняется порядок выполнения других слов. Первым выполняется слово **DUP**, затем слово **square**, которое также имеет соответствующую статью в словаре пользователя.

Описание слова **square** также начинается с выполнения слова **DUP**. Как Форт-система выясняет, что в первом случае после выполнения слова **DUP** надо вернуться в определение слова **cube**, а во втором – в определение слова **square**?

Перед вызовом очередного слова из определения, в стеке возвратов сохраняется так называемый адрес возврата. Это число, определяющее место в метакоде Форт, с которого должно возобновиться выполнение после отработки вызываемого слова.

Обычно адрес возврата указывает на границу между отдельными словами в определении. Таким образом обеспечивается связанное выполнения множества слов Форта.

Когда вызывается слово, определенное через двоеточие, вызывающее слово сохраняет в стеке возвратов адрес возврата. Когда выполнение дойдет до конца определения (до **;**), в метакоде будет выполнено специальное слово, которое извлечет адрес возврата из стека возврата и передаст управление в нужное место метакода.

8.5. Счетные циклы

Вы уже использовали цикл с определенным количеством повторений, или счетный цикл, в одном из первых примеров программы. Теперь, когда вы знаете о стеке возвратов, мы можем подробно рассмотреть все средства, которые Форт предлагает для организации таких циклов.

8.5.1. Слова DO и LOOP

С основным видом счетного цикла вы поработали, когда создавали слово, выводящее цепочку из символов "звездочка". Напомним, что для организации такого цикла используются слова **DO** и **LOOP** .

Слово DO (S: x1 x2 --)(R: -- x1 x2) извлекает с вершины стека начальное значение счетчика цикла, а из следующего слова – предельное значение счетчика (или просто *предел*) цикла. Эти значения слово **DO** размещает в стеке возвратов, причем начальное значение счетчика оказывается на вершине стека возвратов (обратите внимание на стековую диаграмму слова).

Счетчик цикла также называют *индексом цикла*. Индекс цикла хранится на вершине стека возвратов, а его начальное значение устанавливается словом **DO** .

Слово **DO** также отмечает начало цикла. Слова, находящиеся в определении между **DO** и **LOOP** будут выполняться указанное число раз.

Слово LOOP (R: u1 u2 -- u1 u3 |) извлекает индекс цикла и его предел из стека возврата, инкрементирует индекс и сравнивает его с пределом цикла. Если индекс меньше предела цикла, оба значения возвращаются в стек возвратов, а выполнение программы продолжается с первого слова после **DO** . Если индекс стал равен пределу цикла, оба числа удаляются из стека возвратов и выполнение программы продолжается со слова, следующего после **LOOP**.

Происходящие изменения в стеке возвратов иллюстрирует стековая диаграмма слова. Вертикальная черта указывает, что возможны два результата выполнения слова, во втором случае из стека возвратов удаляются две верхние ячейки.

Слово **LOOP** рассматривает предел и индекс цикла как числа без знака. Обратите внимание, что слова внутри цикла будут выполнены как минимум один раз, прежде чем слово **LOOP** инкрементирует индекс и сравнит его с пределом.

Из этого следует, что если задать начальное значение индекса цикла равным пределу, вы получите максимально возможное число повторений. При размере ячейки в 16 бит это число равно 65536.

Применять цикла вида DO…LOOP очень просто. Предположим, вам надо выводить некоторое количество пробелов для выравнивания чисел в таблице. В Форте есть слово **SPACE** , которое выводит один пробел на экран.

Определим новое слово:

```
: Spaces NUL DO SPACE LOOP ;
```

Константа **NUL** задает начальное значение индекса цикла, а вот предел должен находиться в стеке до вызова **Spaces** . Как вы следует из определения, слово **Spaces** выводит на экран указанное число пробелов.

8.5.2. Слово +LOOP

С помощью слова **+LOOP** можно организовать более мощный и гибкий вид счетного цикла. Как и в первом случае, для инициализации цикла используется слово **DO** .

Слово +LOOP (S: n1 --)(R: n2 n3 – n2 n4 |) получает из стека параметров величину *приращения* индекса цикла. Это приращение складывается с текущим значением индекса, а полученный результат сравнивается с пределом.

Если новое значение индекса пересекло границу между значением предела и значением предела минус один, выполняется выход из цикла. В противном случае выполняется переход в начало цикла, отмеченное словом **DO** .

Значения приращения, индекса и предела рассматриваются словом **+LOOP** как числа со знаком (то есть могут быть и отрицательные). При выходе из цикла из стека возвратов удаляются значения индекса и предела.

Проиллюстрируем применение слова **+LOOP** на практическом примере.

Предположим, что мы создаем программу для управления автоматическим фасовщиком фруктов.

Фрукты (яблоки, например) по одному поступают из бункера. Каждый фрукт сначала попадает на электронные весы. После взвешивания фрукт падает в фасовочный пакет, когда из бункера подается следующий фрукт и сталкивает его с весов.

Будем считать, что у нас уже имеются слова, управляющие оборудованием:

Next-Fruit (--) – подает следующий фрукт из бункера на весы;

Get-Weight (-- u) – возвращает вес фрукта, находящегося на весах, в граммах;

Open-Pack (--) – открывает пустой фасовочный пакет для заполнения;

Close-Pack (--) – закрывает и удаляет заполненный фруктами фасовочный пакет.

Определим слово, которое выдает из автомата пакет с фруктами, вес которых не превышает, например, 2 кг, но близок к нему (насколько возможно):

```
: Next-Pack
  Open-Pack
  2000 Get-Weight DO Next-Fruit Get-Weight +LOOP
  Close-Pack ;
```

Разберемся, как это работает.

Слово **Open-Pack** подготавливает новый фасовочный пакет для наполнения.

До начала цикла в стек помещается число 2000 – это предельно допустимый вес фруктов в упаковке (2 кг = 2000 г), а слово **Get-Weight** помещает в стек вес фрукта (тоже в граммах), находящегося в данный момент на весах, то есть начальное значение индекса цикла.

Наше слово **Next-Pack** будет вызываться, пока есть фрукты в бункере, поэтому при первом вызове на весах не будет ничего (вес = 0), а при последующих будет лежать фрукт, не попавший в предыдущую фасовку.

Слово **DO** инициирует цикл, индекс которого равен весу фруктов, попавших в пакет, а предел равен максимально допустимому весу фруктов в пакете.

Слово **Next-Fruit** подает на весы следующий фрукт и сбрасывает уже взвешенный фрукт в пакет. Слово **Get-Weight** возвращает в стеке вес фрукта в граммах.

Слово **+LOOP** складывает вес фрукта на весах (приращение цикла) с весом фруктов в пакете (текущее значение индекса цикла) и сравнивает с пределом. Если добавление в пакет фрукта с весов не превысит допустимого веса, индекс цикла обновляется, выполнение передается в начало цикла, где слово **Next-Fruit** сбрасывает взвешенный фрукт в пакет.

Если же вес фрукта слишком велик, цикл завершается, а фрукт так и остается лежать на весах.

Слово **Close-Pack** запечатывает упаковку и выполнение слова **Next-Pack** завершается.

8.5.3. Вложенные циклы. Слова I, J и K

В последнем примере вес фруктов в упаковке совпадал со значением индекса цикла. Было бы хорошо получить это значение, чтобы распечатать вес на товарном ярлыке.

Форт предоставляет возможность получения значения индекса цикла с помощью слова **I** .

Создайте слово

```
: Test-I 5 NUL DO I . LOOP ;
```

и выполните его. Вы увидите на экране

```
0 1 2 3 4 Ok
```

Слово **I** возвращает в стеке параметров значение индекса текущего цикла. Почему именно текущего? Циклы могут быть вложены один в другой, как матрешки.

Создайте еще одно слово:

```
: Test-J
  3 NUL DO
        2 NUL DO
              I . J . NL
        LOOP
  LOOP ;
```

Мы специально выделили отступами вложенные циклы, хотя можно было бы написать все в одну строку. Такое изображение управляющих структур нагляднее.

Выполнив это слово, мы получим на экране:

```
0 0
1 0
0 1
1 1
0 2
1 2
Ok
```

В каждой строке первое число выводится словом **I**, это индекс внутреннего цикла. Второе число возвращается словом **J** – это значение индекса цикла, внешнего по отношению к текущему.

Обратите внимание, что вывод на экран происходит во внутреннем цикле. Если мы хотим выводить значение индекса в каждом цикле, мы должны использовать только **I** .

Слово

```
: Test-I-I
  3 NUL DO
        2 NUL DO
              I .
        LOOP
        I . NL
  LOOP ;
```

выведет на экран уже иной результат:

```
0 1 0
0 1 1
0 1 2
Ok
```

Слово **K** возвращает в стеке значение индекса цикла второго внешнего уровня, по отношению к текущему. Попросту говоря, если вы вложили один в другой три цикла, то для получения индекса самого первого цикла внутри самого последнего вам понадобится **K** .

Введите слово

```
: Test-IJK
  4 NUL DO
        3 NUL DO
              2 NUL DO
                    ." I=" I . ." J=" J . ." K=" K . NL
              LOOP
        LOOP
  LOOP ;
```

Вы должны самостоятельно разобраться, как это работает.

8.5.4. Досрочное завершение цикла. Слово LEAVE

Иногда требуется завершить выполнение цикла раньше, чем индекс достигнет предела.

Для досрочного завершения цикла используется слово LEAVE (R: x1 x2 --). Слово **LEAVE** удаляет из стека возвратов две верхние ячейки (предел и индекс цикла) и продолжает выполнение программы со слова, следующего за ближайшим словом **LOOP** или **+LOOP** .

Слово **LEAVE** обычно используется внутри структуры IF…THEN. Поясним на примере.

Введите

```
: Test-Leave 7 NUL DO I . I 3 = IF LEAVE THEN LOOP ;
```

и выполните это слово. Вы получите

```
0 1 2 3 Ok
```

хотя предел цикла был равен 7.

После вывода на экран индекса цикла он сравнивается с числом 3. Если индекс равен 3, выполняется слово **LEAVE** и цикл завершается.

8.6. Прекращение выполнения слова и программы

Обычно исполнение слова Форт продолжается до тех пор, пока не будет выполнено последнее слово в его определении. Форт-программа выполняется до конца слова, с которого выполнение началось. В некоторых случаях нужно прекратить исполнение слова или программы досрочно. Завершение исполнения слова возвращает управление слову, которое его вызвало, прекращение выполнения программы возвращает управление Форт-системе.

8.6.1. Досрочный выход из слова. Слово EXIT

В качестве примера вспомним слово, которое отображало вводимые с клавиатуры символы до тех пор, пока не нажималась клавиша Enter.

```
: Test1 BEGIN KEY DUP EMIT CR = UNTIL ;
```

В дальнейшем мы модифицировали это слово таким образом, что оно завершало работу при нажатии клавиши Escape, но не выводило при этом управляющий символ с кодом 27 на экран. Попробуем объединить эти примеры в одном слове:

```
: Test0 BEGIN KEY DUP
  ESC = IF DROP EXIT THEN
  DUP EMIT CR = UNTIL ;
```

В первой строке мы отмечаем начало цикла, ждем нажатия клавиши и делаем копию кода введенного символа в стеке.

Во второй строке мы сравниваем верхнюю копию кода символа с кодом клавиши Escape (не забудьте предварительно создать константу **ESC** с десятичным кодом 27). Если была нажата клавиша Escape, выполняются слова между **IF** и **THEN** . Слово **DROP** освобождает стек от оставшейся копии кода клавиши, а слово **EXIT** завершает выполнение слова.

В третьей строке выводится символ и проверяется нажатие клавиши Enter.

Слово **EXIT** извлекает из стека возврата число, который там сохранило слово, вызывавшее **Test0**. Это число используется для возврата управления вызывавшему слову при завершении **Test0** , поэтому оно называется *адрес возврата*.

Строго говоря, одно из действий слова **;** (точка с запятой) при определении нового слова – запись в метакод слова **EXIT** в конце определения.

8.6.2. Выход из слова внутри цикла. Слово UNLOOP

Если вы хотите досрочно выйти из слова внутри счетного цикла вида DO…LOOP (или +LOOP), необходимо помнить, что после адреса возврата в стек возвратов были помещены еще и параметры цикла.

Удалить из стека возвратов параметры текущего цикла можно с помощью слова **UNLOOP**, а уже потом использовать **EXIT**.

Выглядит это так:

```
: Test-Unloop
  10 NUL DO I . KEY CR = IF UNLOOP EXIT THEN LOOP
  ." Цикл завершен! " ;
```

Если вы выполните это слово, после каждого выведенного значения индекса цикла вам придется нажимать клавишу, например пробел:

```
0 1 2 3 4 5 6 7 8 9 Цикл завершен! Ok
```

Но если вы нажмете клавишу Enter до завершения цикла, выполнение слова немедленно прекратится и вы не увидите финальное сообщение.

8.6.3. Досрочный выход из программы. Слова ABORT и QUIT

Слово **ABORT** прекращает выполнение всей программы и передает управление Форт-системе. При этом производится установка в начальное состояние различных структур управления памятью и очищаются стеки. Слово **ABORT** применяется при обнаружении критических ошибок, например, сбоев аппаратуры (поэтому при выполнении слова в интерактивной среде сообщение **Ok** не выводится).

Слово **QUIT** используется для отладки программ при работе в интерактивном режиме. Оно похоже по своему действию на слово **ABORT**, но не очищает стек параметров. Если вы вставите слово **QUIT** внутри определения какого-либо слова, то после выхода в интерактивную систему программирования сможете проанализировать данные в стеке.

Хотя слово **QUIT** допускает использование в автономном режиме, лучше не оставлять его в финальной версии программы, работающей автономно. При выполнении слова **QUIT** в автономном режиме будет произведена попытка запуска интерактивной среды, а это не всегда допустимо.

8.7. Операции в стеке возвратов

В стеке возвратов можно временно хранить используемые словом значения, передавая их из стека параметров (и обратно).

Список слов для операций с данными в стеке возвратов, доступных в AFS уровня L0, содержит Приложение C (Таблица C.6).

Важно помнить, что если в определении используются слова, изменяющие стек возвратов, его нужно восстанавливать в исходное состояние до окончания определения слова или использования слова **EXIT** , а внутри счетного цикла – до выхода из цикла.

8.7.1. Слова >R и R>

Для перемещения числа из стека параметров в стек возвратов в языке Форт используется слово >R (S: x --) (R: -- x). Слово **>R** снимает число с вершины стека параметров и помещает его на вершину стека возвратов.

Для обратного перемещения используется слово R> (S: -- x) (R: x --). Слово **R>** снимает число с вершины стека возвратов и помещает его на вершину стека параметров.

Таким образом можно временно сохранить какое-либо значение, не заводя переменную в памяти данных.

Для примера создадим слово, которое выводит значение числа на вершине стека в шестнадцатеричной системе (без знака), независимо от того, какая система счисления установлена и не изменяя ее.

Установим десятичное счисление и введем определение слова:

```
DECIMAL

: U.HEX RADIX@ >R 16 RADIX! U. R> RADIX! ;
```

Проследим выполнение слова **U.HEX** . Сначала слово **RADIX@** кладет текущее основание системы счисления на вершину стека параметров, откуда его извлекает слово **>R** и помещает на вершину стека возвратов.

Далее с помощью слов **16 RADIX!** мы устанавливаем шестнадцатеричное счисление. Слово **U.** распечатывает число на вершине стека как беззнаковое.

Слово **R>** извлекает из стека возвратов предыдущее значение основания системы счисления, а слово **RADIX!** восстанавливает его.

Разумеется, все изменения системы счисления происходят только в момент выполнения слова **U.HEX** .

8.7.2. Слова 2>R и 2R>

Когда возникает необходимость поместить в стеке возвратов число двойной длины, а затем вернуть его в стек параметров, используйте специально предназначенные для этого слова.

Слово 2>R (S: xx --) (R: -- xx) извлекает с вершины стека параметров двойное число и помещает его на вершину стека возвратов.

Обратное перемещение осуществляет слово 2R> (S: -- xx) (R: xx --).

Использование этих слов гарантирует, что на вершине стека (и параметров, и возврата) всегда будет находиться младшая ячейка числа двойной длины.

8.7.3. Слова R@ и 2R@

Еще одно слово позволяет многократно использовать число, находящееся на вершине стека возвратов. Слово R@ (S: -- x) (R: x -- x) считывает значение из стека возвратов и помещает его копию на вершину стека параметров. Число в стеке возвратов остается на месте и без изменений.

Аналогично, для чисел двойной длины, используется слово 2R@ (S: -- xx) (R: xx -- xx). Порядок ячеек при копировании из одного стека в другой не изменяется.

9. Обработка текста

Отличительной особенностью Форт-системы является ее интерактивность и ориентированность на использование в общении с пользователем языка, близкого к человеческому.

Вы можете использовать в своих программах, написанных на Форте, мощные средства обработки текстовой информации, имеющиеся в системе.

Сводный перечень слов для ввода, обработки и вывода текста содержит Приложение С, Таблица С.12.

9.1. Преобразование в текст и форматирование числовых данных

Наиболее часто требуется выводить в текстовом виде, понятном пользователю, результаты обработки числовых данных.

Форт использует оригинальный способ преобразования числа в текст.

В секции данных программы имеется особая область, именуемая "буфер форматированного вывода", или FOB (Formatted Output Buffer). В этом буфере происходит посимвольная сборка текстовой строки для отображения числового значения. Вы можете управлять этим процессом с помощью нескольких слов системного словаря Форт.

9.1.1. Слова <# , # и #>

Процесс получения текстовой строки, соответствующей заданному числу, начинается словом <# (--). Слово **<#** устанавливает в исходное состояние указатель в буфере FOB (перемещает его в конец буфера) и обнуляет длину строки.

Непосредственное преобразование числа в текст осуществляется словом # (ud1 -- ud2). Слово **#** извлекает двойное число без знака с вершины стека параметров (ud1) и делит его на основание системы счисления. Полученный при делении остаток является числовым значением младшего разряда числа.

Частное от деления возвращается в стек параметров (ud2), а остаток преобразуется в символ ASCII по следующему правилу: числа от 0 до 9 преобразуются в символы 0-9, числа превышающие 9 преобразуются в буквы A, B, C и так далее.

Полученный символ помещается в буфер FOB по текущему положению указателя, после чего указатель смещается на 1 байт в направлении начала буфера, а длина строки увеличивается на 1.

Следующее применение слова **#** к двойному числу в стеке дает очередной разряд числа. Слово **#** может повторяться необходимое количество раз для получения всех разрядов текстового представления заданного числа.

Завершается преобразование словом #> (ud – c-addr u). Слово **#>** удаляет из стека двойное число и помещает вместо него адрес первого символа полученной строки и длину строки (на вершину стека). Если теперь выполнить слово **TYPE**, вы увидите значение заданного числа в установленной системе счисления.

Чтобы было понятно, как это работает, попрактикуемся.

В программировании часто бывает удобно представлять значения в шестнадцатеричной системе, выводя все разряды, в том числе незначащие. Мы работаем с ячейками по 16 бит, значит, нам будет полезно иметь слово, выводящее значение с вершины стека в виде 4-разрядного шестнадцатеричного числа.

Введите:

```
DECIMAL
: .HEX RADIX@ >R 16 RADIX!
U>D <# # # # # #>
TYPE SPACE
R> RADIX! ;
```

В первой строке определения слова **.HEX** мы сохраняем текущее основание системы счисления в стеке возвратов и заменяем его на 16.

Во второй строке мы преобразуем число на вершине стека в двойное число без знака с помощью **U>D** , как этого требуют слова для преобразования в текст.

Слово **<#** инициализирует буфер форматированного вывода. Следующие за ним четыре слова **#** формируют четыре разряда числа от младшего к старшему. Завершает преобразование слово **#>**, которое удаляет из стека двойное число и возвращает указатель на начало строки в FOB и ее длину.

Строка выводится на экран словом **TYPE**, которое извлекает необходимые параметры из стека. Слово **SPACE** выводит в конце строки пробел.

Последняя строка определения восстанавливает из стека возвратов исходную систему счисления.

Проверим, работает ли **.HEX** . Введите (в десятичной системе)

```
35678 .HEX
```

Должно получиться

```
8B5E Ok
```

Если вы введете

```
20 .HEX
```

получите

```
0014 Ok
```

Слово **.HEX** показывает все 4 разряда ячейки, даже если слева будут нули.

9.1.2. Слово #S

Если не требуется преобразование в текст незначащих разрядов числа слева, можно использовать слово #S (ud1 -- ud2).

Слово **#S** вычисляет младший разряд числа и помещает его в буфер FOB, также, как это делает слово **#** . Если частное после деления на основание счисления не равно 0, вычисляется следующий разряд.

Добавление разрядов повторяется до тех пор, пока частное не станет равным 0.

Применение слова **#S** наглядно демонстрирует следующий пример:

```
: UD. <# #S #> TYPE SPACE ;
```

(Набирать пример не надо, так как это слово уже есть в системе).

9.1.3. Числа с фиксированной точкой. Слово HOLD

В языке программирования Форт часто применяются так называемые числа с фиксированной точкой. Это целые числа, при использовании которых подразумевается наличие целой и дробной части, с фиксированной позицией десятичной точки.

Проще говоря, можно получать, например, значения измерений напряжения, выраженные в милливольтах и обрабатывать их как целые значения. При этом выводить на экран результаты вы можете в вольтах, с десятичной точкой и тремя знаками после нее.

Каким образом форматируется вывод такого числа? Комбинация слов **#** и **#S** позволяет преобразовать дробную и целую часть числа, а добавить символ десятичной точки в нужном месте строки поможет слово HOLD (char --).

Слово **HOLD** получает на вершине стека код ASCII символа, который записывается в FOB в текущей позиции указателя, после чего указатель буфера смещается в направлении начала строки.

Поясним на примере, как это работает. Определим слово для вывода напряжения в вольтах с тремя знаками после запятой:

```
DECIMAL

: U.V U>D <# # # # 46 HOLD #S #> TYPE SPACE ;
```

На вершине стека слово **U.V** получает целое значение напряжения, выраженное в милливольтах. Слово **U>D** преобразует его в двойное число без знака. Далее следуют уже знакомые нам слова.

Слово **<#** инициализирует FOB, а три слова **#** формируют три младших разряда дробной части.

Слова **46 HOLD** помещают в буфер символ "точка" в следующей позиции строки (число 46 – это код ASCII символа "точка").

Оставшееся после преобразования младших разрядов двойное число в стеке преобразуется словом **#S** как целая часть, разряды которой добавляются слева от десятичной точки.

Введите

```
12345 U.V
```

и вы получите

```
12.345 Ok
```

Обратите внимание, что разряды числа преобразуются от младшего к старшему и символы добавляются в строку справа налево!

С помощью слова **HOLD** вы можете добавлять в строку форматированного вывода любые символы – разделители групп разрядов, знаки дробей, пробелы и т.п.

9.1.4. Знак числа. Слово SIGN

До сих пор мы преобразовывали в текст только числа без знака. Что делать, если необходимо выполнить преобразование числа со знаком, а число это отрицательное?

Слово SIGN (n --) получает на вершине стека число и проверяет его знак. Если число отрицательное, в буфер FOB помещается символ "минус" в текущей позиции указателя. Если число положительное, ничего не происходит.

Изучим работу слова **SIGN** на примере. Вот так может быть определено хорошо знакомое вам слово **.** ("точка"):

```
: . DUP >R ABS U>D <# #S R> SIGN #> TYPE SPACE ;
```

(Набирать пример не надо, так как это слово уже есть в системе).

Слово **DUP** копирует число на вершине стека, а слово **>R** перемещает копию в стек возвратов для последующего использования.

Слово **ABS** находит абсолютное значение числа, а слово **U>D** преобразует в двойное число без знака.

Далее следует уже знакомое слово **<#** и слово **#S**, которое полностью преобразует абсолютное значение исходного числа в текстовую строку. Указатель буфера FOB теперь находится левее старшего разряда числа.

Слово **R>** перемещает исходное значение из стека возвратов в стек параметров, откуда его извлекает слово **SIGN** для определения знака числа. Если число отрицательное, **SIGN** помещает символ "минус" в буфер FOB и сдвигает указатель к началу строки. Если число было положительное, ничего не происходит.

Действие оставшихся слов вам уже известно.

9.1.5. Преобразование текста в число. Слово TO-NUMBER

Слово TO-NUMBER (ud1 c-addr1 u1 -- ud2 c-addr2 u2 u3) выполняет преобразование текстовой строки в число. Слово имеет множество параметров, как входных так и выходных.

Начнем с входных параметров. На вершине стека слово получает длину текстовой строки u1 и адрес первого символа c-addr1. Ниже находится двойное число без знака ud1, значение которого используется как начальное для преобразования строки. Чаще всего это число равно 0.

Для преобразования используется текущее основание счисления.

Строка преобразуется посимвольно слева направо. Каждый символ анализируется на соответствие системе счисления и преобразуется в число. Находящееся в стеке двойное число без знака умножается на основание системы счисления, что соответствует сдвигу этого числа на один разряд влево. (Здесь мы говорим о разряде числа в действующей системе счисления). Затем к нему прибавляется только что полученное числовое значение разряда.

Эта операция повторяется либо до конца строки, либо до появления символа, который не может представлять число в данной системе счисления.

Если в процессе преобразования встретился нечисловой символ, его код сравнивается с кодом ASCII символа десятичной точки. Если это десятичная точка, ее позиция относительно конца строки запоминается, а преобразование продолжается со следующего символа.

Если встречен нечисловой символ, не являющийся десятичной точкой, либо десятичная точка встретилась повторно, преобразование прекращается.

Слово **TO-NUMBER** возвращает на вершине стека номер позиции справа для десятичной точки u3 (или 0, если точка не встретилась), затем длину оставшейся не преобразованной части строки u2 и адрес первого нечислового символа c-addr2. Если строка преобразована полностью, параметр u2 равен 0. Наконец, двойное число без знака ud2 содержит результат преобразования строки (или первого фрагмента).

Обратите внимание на то, что преобразование производится как для числа без знака. Это сделано намеренно для большей гибкости и не представляет проблем.

Если вы хотите преобразовывать числа со знаком, прочитайте первый символ строки и сравните его с кодом символа "минус". Если строка начинается со знака "минус", надо просто передвинуть вперед указатель на начало строки и уменьшить значение длины. После завершения преобразования потребуется изменить знак полученного двойного числа.

Применяя **TO-NUMBER** последовательно к фрагментам строки, можно преобразовывать самые разные формы записи чисел. Например, вы можете создать слова, распознающие строки вида

-123.45

1'234'567.89

-0.8345E-2

A000:8002 (в шестнадцатеричной системе) и другие.

В стандартном языке Форт аналогичное преобразование выполняется словом >NUMBER, которое отличается выходными параметрами и обладает меньшей гибкостью. В частности, слово >NUMBER не выделяет позицию десятичной точки.

10. Интерпретация, компиляция и выполнение программы

Что происходит в системе, когда вы выполняете существующие слова и создаете новые? Как формируется и выполняется программа?

Существует три главных режима работы Форт-системы: интерпретация, компиляция и выполнение программы.

10.1. Интерпретация

Режим интерпретации – первое состояние Форт-системы при запуске сеанса интерактивного программирования.

В этом режиме вводимые числа помещаются в стек параметров, а вводимые слова немедленно выполняются.

Режим интерпретации позволяет производить предварительные расчеты констант, отладку слов, настройку конфигурации системы и другие вспомогательные действия.

Не все слова могут выполняться в этом режиме. Например, невозможно интерпретировать такие слова, как **THEN** или **LOOP**. Попытка интерпретации слова ***;*** не имеет никакого смысла.

Чтобы не возникало ошибок при интерпретации слов, предназначенных для других режимов, Форт-система распознает их назначение при помощи специальных атрибутов, присвоенных каждой записи словаря.

Когда с помощью слова **WORDS** вы просматриваете содержимое словаря, справа от слова выводятся его атрибуты:

```
*                                   |S---
```

Атрибут S означает, что слово принадлежит системному словарю и его нельзя удалить.

Посмотрим на атрибуты других слов, например:

```
HEX                                 |S--I
```

Атрибут I показывает что слово должно быть выполнено немедленно независимо от режима. Такие слова называются "словами немедленного выполнения". Возможность такого выполнения нужна для режима компиляции, о котором речь пойдет далее.

```
UNTIL                               |S-CI
```

Атрибут C дополняет атрибут I, сообщая, что слово должно быть выполнено немедленно, но только во время компиляции определения нового слова.

```
UNLOOP                              |SR--
```

Атрибут R встречается не часто. Слова с таким атрибутом относятся к времени выполнения слова, в определении которого они скомпилированы. Такие слова не исполняются в режиме интерпретации или компиляции, но компилируются в определении как большинство других слов.

10.2. Компиляция

Переход из состояния интерпретации в состояние компиляции происходит автоматически, когда выполняется слово **:** .

В режиме компиляции выполняются только слова, имеющие атрибут `I`.

В словаре для каждого слова сохраняются не только атрибуты. Если слово найдено, процедура поиска возвращает так называемый токен (token - знак, признак). Мы будем использовать этот термин, чтобы отличать его от указателя – переменной, содержащей адрес в памяти.

Токены AFS являются частью метакода Advanced Forth. С помощью токена можно выполнить слово, скомпилировать его в определение другого слова, создать ссылку на слово для табличного перехода. Далее мы рассмотрим эти возможности подробнее.

Получив токен, Форт-система добавляет его к метакоду создаваемого определения.

Если при разборе входной текстовой строки в режиме компиляции обнаруживается число, в метакод должны быть добавлены команды, помещающие это число в стек параметров при выполнении создаваемого слова.

Существуют скрытые слова Форта, которые нельзя найти в словаре, но для которых также существуют токены. Одно из таких слов компилируется в метакод определения вместе с числом (литералом). При выполнении определения эта пара работает как слово-константа – возвращает в стек параметров число, введенное на стадии компиляции.

Если все вводимые слова не выполняются а компилируются, может возникнуть проблема с управлением этим процессом. Например, если мы создаем цикл внутри определения, мы не можем просто записать в метакод токены для слов **DO** и **LOOP**. Оба этих слова должны быть выполнены дважды.

Первый раз они выполняются во время компиляции определения. Слово **DO** отмечает место в метакоде, на которое должен быть создан переход в конце цикла. Слово **LOOP** создает метакод, осуществляющий переход в точку, отмеченную словом **DO**.

Второй раз эти же слова должны быть выполнены во время выполнения созданного слова. Слово **DO** помещает параметры цикла в стек возвратов, а слово **LOOP** выполняет действия по модификации индекса цикла и принятию решения о повторении или выходе.

Разумеется, столь разные действия не могут производиться одним и тем же словом. На самом деле используется два различных варианта для **DO** и **LOOP**.

Собственно **DO** и **LOOP** являются словами немедленного выполнения в режиме компиляции. Они участвуют в формировании структуры метакода, а в нужные места этой структуры компилируются токены скрытых слов времени выполнения. Эти скрытые слова и выполняют инициализацию цикла и необходимые проверки и переходы.

Слов, которые ведут себя по-разному в разных режимах, довольно много. Практически все слова режима компиляции имеют скрытых "двойников".

Особенно интересно слово **TO**. Оно может выполняться в режимах интерпретации и компиляции, но в каждом режиме выполняется свой вариант, а при компиляции в определение нового слова записывается токен скрытого слова времени выполнения. Итого – три разных слова под одним именем.

10.2.1. Отложенное выполнение. Слово POSTPONE

Иногда возникает необходимость скомпилировать в определение слово немедленного выполнения.

Например, когда мы создавали слово для вывода на экран чисел в шестнадцатеричном формате без изменения системы счисления

```
: U.HEX RADIX@ >R 16 RADIX! U. R> RADIX! ;
```

мы не использовали в нем слово **HEX**, так как оно было бы выполнено немедленно и не попало в определение слова.

Для компиляции слов немедленного выполнения используется слово **POSTPONE** (отложить). Оно временно отменяет немедленное выполнение для следующего за ним слова, и таким образом компилирует его в определение.

Используя слово **POSTPONE** можно записать наш пример так:

```
: U.HEX RADIX@ >R POSTPONE HEX U. R> RADIX! ;
```

Это определение имеет длину метакода на 2 ячейки меньше первоначального варианта.

И раз уж речь зашла об оптимизации, то доведем дело до конца:

```
: U.HEX \ ( u -- ) \ RADIX@ SWAP POSTPONE HEX U. RADIX! ;
```

Здесь мы сэкономили еще одну ячейку, исключив сохранение исходного основания системы счисления в стеке возвратов.

Понятно, что первоначальный вариант как раз и предназначался для демонстрации использования стека возвратов, но, с другой стороны, мы можем видеть здесь, как продуманная оптимизация позволяет сократить размер программы и увеличить ее производительность.

10.2.2. Компиляция литералов. Слова LITERAL и 2LITERAL

Мы уже упоминали, что при компиляции чисел в метакод записывается не только число, но и токен скрытого слова, помещающего это число в стек во время выполнения. Такие "встроенные" числа называются литералами.

Существуют два слова режима компиляции, LITERAL (x --) и 2LITERAL (xx --), которые отвечают за компиляцию литералов. Эти слова (немедленного выполнения) принимают числовые значения не из строки ввода, а из стека параметров (как видно из их стековых диаграмм). Далее все происходит как обычно – каждому из этих слов соответствует свое скрытое слово времени выполнения, токен которого помещается в метакод вместе с числом.

Слово **LITERAL** работает с числами обычного размера, а **2LITERAL** – с числами двойной длины.

Как число попадает в стек во время компиляции определения? Об этом далее.

10.3. Переключение между интерпретацией и компиляцией

Как мы уже говорили, из режима интерпретации в режим компиляции Форт-система переключается автоматически словом **:** . Обратное переключение режимов выполняет слово **;** , когда новое определение закончено и все результаты успешно сохранены.

Также автоматически происходит переключение из компиляции в интерпретацию при возникновении ошибок. Например, в процессе компиляции в словаре не нашлось очередное введенное слово. В этом случае выполняется слово **ABORT**, которое переводит систему в исходное состояние, а это – режим интерпретации.

Возможно принудительное переключение режимов с помощью специальных слов.

10.3.1. Слово [и слово]

Что делать, если вы начали описывать новое слово, и тут вам понадобилось посчитать значение литерала (то есть числа, которое вы хотите скомпилировать в определение)?

Завершать начатое определение по ошибке не очень хорошо. Можно воспользоваться словами переключения состояния системы.

Поясним на примере:

```
: JustTest ." T= " [ 123 12 * 4 - ] LITERAL . NL ;
```

Что мы видим? Внутри определения появились квадратные скобки, а в них – вычисления на стеке параметров. Причем результат остался в стеке!

Слово **[** переключает Форт-систему в режим интерпретации. Это слово немедленного выполнения, поэтому компиляция нового определения приостанавливается, а следующий за **[** текст разбирается уже в режиме интерпретации.

Вся арифметика честно выполняется, а результат сохраняется в стеке. Далее следует слово **]** .

Слово **]** переключает Форт-систему в режим компиляции. После его выполнения возобновляется прерванная компиляция определения, выполняется слово **LITERAL**, которое достает из стека результат вычислений и компилирует его вместе с токеном своего двойника времени выполнения.

Слова **[** и **]** являются мощным средством программирования. Весьма интересным выглядит использование квадратных скобок "наоборот".

Вы можете сделать так:

```
ALIGN CREATE ProcTable ] Proc0 Proc1 Proc2 Proc3 [
```

Предполагается, что слова **Proc0**, **Proc1**, **Proc2** и **Proc3** созданы ранее. Мы получили таблицу переходов к этим словам по индексу.

После переключения в режим компиляции словом **]** , Форт-система продолжила разбор строки ввода. Встреченные слова разыскивались в словаре, а их токены компилировались. Куда?

Поскольку не было открыто новое определение слова, токены компилировались в ячейки секции данных по адресу HERE. После компиляции токена указатель данных сдвигался, чтобы зарезервировать ячейку.

Когда в строке встретилось слово **[** (немедленного выполнения), система вернулась в режим интерпретации, а в памяти данных остался массив, заполненный токенами.

Такую таблицу можно использовать, например, для выполнения слов, соответствующих пунктам меню. Как это сделать, мы узнаем при обсуждении режима выполнения программы.

10.4. Выполнение программы

Выполнение созданной пользователем Форт-программы есть ни что иное, как выполнение последнего определенного слова. Вы можете создать вот такую самую короткую программу и выполнить ее:

```
: TEST \ Самая короткая программа \ ;
```

Очевидно, что в такой программе нет никакого смысла, поэтому последнее определение должно ссылаться на ранее определенные слова, каждое из которых, в свою очередь, использует еще более ранние слова и т.д.

Мы уже рассматривали процесс выполнения слов Форта, когда изучали стек возвратов. Здесь мы коснемся некоторых деталей этого процесса.

10.4.1. Выполнение по токену. Слово EXECUTE

Ранее мы упомянули о возможности создания списка слов, точнее – массива токенов этих слов:

```
ALIGN CREATE ProcTable ] Proc0 Proc1 Proc2 Proc3 [
```

Разумеется, все перечисляемые в списке слова должны существовать. Полученный массив **ProcTable** содержит токен слова **Proc0** в первой ячейке, токен слова **Proc1** во второй ячейке и т.д.

Мы можем прочитать любой из токенов, например, для слова **Proc2**:

```
2 CELLS ProcTable + @
```

(Напоминаем, что первый элемент массива имеет индекс 0).

Теперь на вершине стека лежит токен слова **Proc2** . Мы можем использовать слово EXECUTE (xt --), чтобы выполнить слово по его токену. Слово **EXECUTE** получает токен на вершине стека и выполняет соответствующее слово точно так же, как если бы вы ввели его в интерактивном режиме. (Символ xt в стековой диаграмме обозначает токен AFS).

Посмотрим, как это можно использовать в программе.

Предположим, что при запуске программа выводит меню из 5 пунктов, каждому из которых соответствует свое слово в программе. Для выбора пункта меню надо нажать цифровую клавишу. После несложной проверки и преобразования символа в номер пункта меню, мы имеем число от 0 до 4. Число это соответствует одному из слов, которое надо выполнить.

С теми средствами, которые вы имели до сих пор, программа получилась бы вот такой:

```
: SelectMenu \ ( n -- ) получаем номер слова и выполняем его \
DUP 4 = \ делаем копию номера и сравниваем с 4 \
IF \ если равно, выполняем слово для пункта меню 4 \
 Menu4
ELSE \ если не равно, проверяем следующий вариант \
 DUP 3 =
 IF
  Menu3
 ELSE
  DUP 2 =
  IF
   Menu2
  ELSE
   DUP 1 =
   IF
    Menu1
   ELSE
    Menu0
   THEN
  THEN
 THEN
THEN
DROP ;
```

Скорее всего, вам уже не нравится. А если надо выбирать не из 5, а из 50 вариантов?

Можно предварительно создать список используемых слов:

```
ALIGN CREATE SelectTable ] Menu0 Menu1 Menu2 Menu3 Menu4 [
```

Программа выбора теперь будет выглядеть так:

```
: SelectMenu \ ( n -- ) получаем номер слова и выполняем его \
CELLS \\ вычисляем смещение элемента в таблице
SelectTable + \\ получаем адрес токена в памяти
@ EXECUTE \\ читаем токен и выполняем слово
;
```

Этот вариант понятен, занимает существенно меньше места в памяти и выполняется с максимально возможной скоростью.

Можно существенно улучшить предлагаемое решение.

Скорее всего, список слов не будет изменяться в процессе выполнения программы, поэтому преобразуем его в таблицу констант.

Выполним необходимые подготовительные операции:

```
\\ выравниваем указатель данных и сохраняем его адрес в стеке
ALIGN HERE
\\ создаем массив токенов
] Menu0 Menu1 Menu2 Menu3 Menu4 [
\\ преобразуем массив в таблицу констант из 5 ячеек
5 TABLE MenuSelector
\\ освобождаем ячейки секции данных
5 CELLS NEGATE ALLOT
```

Теперь заново опишем процедуру выбора пунктов меню:

```
: SelectMenu ( n -- ) MenuSelector EXECUTE ;
```

Кроме того, что этот способ работает еще быстрее, он не требует выделения инициализируемых данных в памяти программы.

10.4.2. Получение токена слова. Слово ' (апостроф)

Когда мы создавали псевдонимы системных констант, мы использовали слово `'` (-- xt) для нахождения слов в словаре. Возвращаемое `'` в стеке значение и есть токен слова. Таким образом, если выполнить

```
' WORDS EXECUTE
```

начнется вывод списка существующих слов на экран, как если бы мы выполнили слово **WORDS** .

11. Управление конфигурацией системы

Форт-машина представляет собой тесно связанное сочетание программных и аппаратных средств, использующих ресурсы микроконтроллера на конкурентной основе. Приведем пример.

Первое устройство ввода-вывода Форт-машины, с которым вы познакомились, это текстовая консоль AFS. С помощью консоли вы вводите текст во время сеанса интерактивного программирования и получаете сообщения от системы.

На самом деле это программный интерфейс, который работает поверх аппаратного интерфейса AFM-Link, который, в свою очередь, использует один из универсальных асинхронных передатчиков микроконтроллера (UART).

В младших моделях AFMnano имеется только один UART и если вам необходимо использовать его независимо от текстовой консоли, последнюю придется отключить.

В более сложных моделях может быть два и более UART, что дает возможность использовать текстовую консоль с любым из них.

Для устранения возможных конфликтов и настройки оптимальных режимов работы аппаратуры в AFS предусмотрен специальный программный интерфейс.

11.1. Настройка параметров системы. Слово SYS-CFG!

Слово SYS-CFG! (param n --) предназначено для изменения параметров конфигурации и режимов работы системы. Эти изменения могут относится и к аппаратуре, и к программному обеспечению.

Ячейка на вершине стека содержит номер запроса конфигурации. Вторая ячейка содержит новое значение изменяемого параметра.

Некоторые номера запросов конфигурации и их назначение приведены в таблице.

Таблица 11.1 Номера запросов конфигурации системы и их назначение

Запрос	Название	Описание выполняемого действия
0	CPUCLK	Изменение тактовой частоты процессора
1	HW_CONFIG	Установка типовой конфигурации оборудования

Не все номера запросов могут быть задействованы в используемом вами модуле AFM. Мы привели основные, с которыми вам предстоит встречаться в дальнейшем. Полный список параметров конфигурации приводится в техническом описании модуля.

Если вы укажете неиспользуемый запрос, он будет проигнорирован. Однако, не следует изменять настройки, о которых вы ничего не знаете – это может привести к повреждению аппаратуры.

11.1.1. Запрос конфигурации CPUCLK

Этот запрос предназначен для изменения частоты основного сигнала синхронизации системы – тактового сигнала ядра процессора.

С какой именно частотой работает процессор при включении и в каких пределах эта частота может изменяться, указано в технической документации модуля AFM.

Если вы используете модуль AFMnano-M0.10, то при включении питания или аппаратном сбросе устанавливается частота 24 МГц. Изменить это значение вы можете в диапазоне от 8 МГц до 48 МГц.

В качестве параметра запроса CPUCLK передается новое значение частоты в мегагерцах. Вы можете указать любое значение, но установлено будет ближайшее с шагом в 4 МГц. Если вы укажете значение меньше или больше предельно допустимого, установленное значение будет округлено до минимального или максимального соответственно.

Если вы хотите установить максимальную частоту сигнала синхронизации процессора, введите:

```
48 0 SYS-CFG!
```

Изменение тактовой частоты процессора может потребоваться для уменьшения потребляемого тока питания или для повышения производительности.

Многие устройства ввода-вывода, например, таймеры или UART, используют сигнал синхронизации процессора в качестве опорной частоты. Если вы хотите изменить частоту синхронизации, делайте это в самом начале программы, до инициализации устройств ввода-вывода.

По умолчанию устанавливается частота, оптимальная для данного модуля AFM, поэтому изменять ее без веских причин не рекомендуется.

11.1.2. Запрос конфигурации HW_CONFIG

Запрос конфигурации HW_CONFIG позволяет быстро изменить настройки портов ввода-вывода и подключить встроенные устройства Форт-машины к соответствующим выводам модуля.

В следующем разделе мы подробно рассмотрим устройства ввода-вывода, пока же просто отметим, что существует некоторое число типовых конфигураций аппаратуры, которые используются чаще всего. Эти конфигурации описаны в технической документации модулей AFMnano.

Типовым конфигурациям аппаратуры присвоены номера. Для установки желаемой конфигурации ее номер передается в качестве параметра запроса HW_CONFIG:

```
2 1 SYS-CFG!
```

После выполнения этого примера выводы модуля AFMnano будут установлены в типовую конфигурацию 2.

При включении питания или аппаратном сбросе устанавливается конфигурация с номером 0. Вы всегда можете вернуться к ней, выполнив запрос

```
0 1 SYS-CFG!
```

Прежде чем изменять аппаратную конфигурацию ввода-вывода модуля, изучите документацию и убедитесь, что настройки не создадут конфликт с внешними подключениями.

Не используйте неизвестные значения параметра запроса, это может привести к повреждению аппаратуры. Если вы укажете номер конфигурации, превышающий максимальный поддерживаемый данной системой, будет произведен сброс в исходное состояние – конфигурацию ноль.

11.2. Получение данных настроек и состояния системы. Слово SYS-CFG@

Слово SYS-CFG@ (n -- x) позволяет получить сведения о состоянии и установленных параметрах аппаратуры и программного обеспечения.

Ячейка на вершине стека содержит номер параметра. Слово **SYS-CFG@** возвращает текущее значение параметра. Используемые номера параметров и их описание приведены в таблице.

Таблица 11.2 Номера параметров системы и их описание

Запрос	Название	Описание параметра
0	CPUCLK	Значение тактовой частоты процессора
1	RUN_STATE	Режим автономной программы/интерактивного программирования
2	ROM_PROG	Размер постоянной памяти для сохранения программы пользователя
3	ROM_IDATA	Размер постоянной памяти для инициализируемых данных

Полный список параметров конфигурации приводится в техническом описании модуля.

Если вы укажете неиспользуемый параметр, будет возвращено значение ноль.

11.2.1. Параметр CPUCLK

Выполнение слова **SYS-CFG@** с параметром 0 (CPUCLK) возвращает действующее значение частоты синхронизации процессора в мегагерцах (см. Запрос конфигурации CPUCLK на стр. 126).

11.2.2. Параметр RUN_STATE

В некоторых случаях может оказаться важным, в каком режиме выполняется программа – в автономном или из сеанса интерактивного программирования.

Выполнение слова **SYS-CFG@** с параметром 1 (RUN_STATE) возвращает логический флаг, соответствующий режиму запуска AFS.

Если слово **SYS-CFG@** вернуло флаг "истина", Форт-система запущена в режиме интерактивного программирования. Соответственно, флаг "ложь" возвращается в состоянии автономного выполнения программы.

11.2.3. Параметр ROM_PROG

Параметр ROM_PROG показывает текущий остаток постоянной памяти (Flash-ROM или другого типа), доступной для сохранения словаря и метакода создаваемой программы пользователя.

Размер доступной постоянной памяти уменьшается в процессе создания программы. В этой памяти сохраняется метакод Форт, константы и словарь пользователя. Параметр ROM_PROG показывает, сколько свободной памяти осталось в данный момент.

11.2.4. Параметр ROM_IDATA

Параметр ROM_IDATA представляет размер постоянной памяти (Flash-ROM или другого типа), выделенной для сохранения инициализируемых данных.

По сути, этот параметр является максимальным возможным значением размера инициализируемых данных, который вы можете передать слову **IDATA!**. Если вы попытаетесь создать область инициализируемых данных большего размера, при сохранении сеанса или автономной программы будет выдана ошибка.

11.3. Системное время. Слова SYS-TICK и SYS-TIME

Мы уже использовали слово **MS** , которое задерживает выполнение программы на указанное в миллисекундах время. Для отсчета временных интервалов используется системный таймер Форт-машины. Это аппаратное устройство, которое начинает работу в момент включения или сброса системы.

Системный таймер считает импульсы синхронизации ядра процессора. Зная частоту синхронизации, можно определить количество импульсов ("тиков") в одной миллисекунде. Количество миллисекунд, прошедших с момента старта, считает внутренняя служба времени AFS.

Слово SYS-TICK (-- ud) возвращает текущее значение счетчика тиков системного времени в виде двойного числа без знака.

Слово SYS-TIME (-- ud) возвращает текущее время системы в миллисекундах, также в двойном числе без знака.

11.4. Настройка параметров текстовой консоли AFS. Слово CON-CTRL

Для управления режимами работы текстовой консоли системы предназначено слово CON-CTRL (param n --). Ячейка на вершине стека содержит номер запроса управления. Вторая ячейка содержит новое значение изменяемого параметра.

Используемые номера запросов управления и их назначение приведены в таблице.

Таблица 11.3 Номера запросов управления системной консоли и их назначение

Запрос	Название	Описание выполняемого действия
0	CON_ONOFF	Включение и выключение текстовой консоли AFS
1	CON_ECHO	Управление режимом эха текстовой консоли AFS

Здесь указаны только два основных запроса управления, в сложных системах их может быть больше. Если вы укажете неиспользуемый запрос, он будет проигнорирован.

11.4.1. Запрос управления CON_ONOFF

Запрос CON_ONOFF предназначен для выключения текстовой консоли, если возникает такая необходимость, и включения для возобновления работы.

Если в качестве параметра указать ноль, консоль будет отключена. Если вы выполните

```
0 0 CON-CTRL
```

то больше не сможете работать с AFS и вам придется перезагружать Форт-машину.

Включить консоль можно, задав не равный нулю параметр запроса:

```
1 0 CON-CTRL
```

Для чего же нужна такая возможность? В первую очередь для разделения совместно используемых ресурсов и для отладки. В некоторых младших моделях AFMnano имеется только один UART и если вам необходимо использовать его независимо от текстовой консоли, последнюю придется отключить.

11.4.2. Запрос управления CON_ECHO

Запрос CON_ECHO предназначен для управления режимом эха текстовой консоли AFS. Параметр может принимать два значения – 0 (режим эха отключен) или любое значение, не равное 0 (режим эха включен).

В режиме эха текстовая консоль отображает на экране каждый полученный от клавиатуры символ для облегчения редактирования и контроля передачи информации. Этот режим особенно полезен при удаленном доступе к Форт-машине через последовательный терминал, когда символы могут быть искажены в линии связи.

Если вы выполните

```
1 1 CON-CTRL
```

то увидите, что символы на экране терминала "двоятся".

Это происходит, когда в программе терминала включен режим "локальное эхо" (см. Рисунок 2.2). Первый символ выводится самим терминалом, а второй принимается от консоли AFS. Отключите режим "локальное эхо" в программе терминала и вы снова будете видеть привычный текст, но при этом выводимые на экран символы будут поступать от AFS.

Если режим эха консоли не требуется, отключите его:

```
0 1 CON-CTRL
```

Разумеется, режим "локального эха" терминала в этом случае надо снова включить.

11.5. Получение данных настроек и состояния консоли. Слово CON-STAT

Слово CON-STAT (n -- x) позволяет прочитать состояния режимов, установленных словом **CON-CTRL**.

Ячейка на вершине стека содержит номер параметра. Слово **CON-STAT** возвращает текущее значение параметра. Используемые номера параметров и их описание приведены в таблице.

Таблица 11.4 Номера параметров состояния текстовой консоли AFS и их описание

Запрос	Название	Описание параметра
0	CON_ONOFF	Текущее состояние системной текстовой консоли
1	CON_ECHO	Текущий режим эха текстовой консоли AFS

Как видно из таблицы, слово **CON-STAT** использует те же номера запросов, что и слово **CON-CTRL**. Если режим включен, возвращается значение, не равное 0.

12. Управление устройствами ввода-вывода

Большинство Форт-машин семейства AFM построены на однокристальных микроконтроллерах. Современный микроконтроллер по многим характеристикам может превосходить настольные компьютеры первых поколений. Наиболее впечатляющим выглядит набор встроенных устройств ввода-вывода.

Микросхемы проектируются, используя готовые библиотечные блоки (макроблоки), описывающие ядро процессора, память и различные периферийные устройства. Число и сложность макроблоков устройств ввода-вывода ограничены только площадью кристалла и стоимостью технологии изготовления конкретного микроконтроллера.

Независимое управление питанием и сигналами синхронизации блоков ввода-вывода обеспечивают энергоэффективность микросхемы и снижают электромагнитные помехи.

Даже у самой простой модели семейства, AFMnano-M0, кроме центрального процессора, ОЗУ и ПЗУ, на кристалле микроконтроллера находятся:

- дискретные порты ввода-вывода общего назначения (GPIO);
- аналогово-цифровой преобразователь с мультиплексором входов (ADC);
- последовательный периферийный интерфейс (SPI);
- интерфейс Inter-integrated Circuit (I2C);
- универсальный асинхронный последовательный интерфейс (UART);
- несколько многоканальных таймеров-счетчиков (TIM).

Старшие модели семейства имеют более одного устройства ввода-вывода каждого вида, и другие, не перечисленные здесь.

Дальнейшие примеры подразумевают использование модуля AFMnano-M2, желательно в комплекте с исследовательской платой AFM Evo-I. Модуль AFMnano-M2 имеет достаточное количество выводов для одновременного использования всех имеющихся типов встроенных устройств при установки типовой конфигурации №1.

Если вы используете другой модуль, перед выполнением примеров проверьте по прилагаемой документации, на какие выводы модуля могут быть сконфигурированы те или иные встроенные устройства микроконтроллера.

Каждое устройство ввода-вывода – это набор регистров управления, регистров состояния, регистров данных, буферов и т.п. Для некоторых устройств этот набор выглядит весьма внушительно. Например, сложные многоканальные счетчики-таймеры имеют порядка 20 регистров управления и данных.

Программирование различных устройств ввода-вывода заметно отличается. Символьные устройства, такие как последовательный интерфейс UART или I2C, используются, как правило, для работы с хорошо известными протоколами обмена информацией. Для них все тонкости программирования могут быть скрыты драйвером, достаточно использования нескольких стандартных слов интерфейса прикладной программы (API), чтобы реализовать большинство задач.

С другой стороны, существуют такие устройства, для которых создание унифицированного драйвера и API затруднено или попросту не имеет смысла, например, таймеры-счетчики. Вариантов их аппаратных реализаций и способов применения очень много, и программисты предпочитают работать с таймерами напрямую, программируя отдельные регистры.

Advanced Forth System предлагает оба способа обращения с устройствами ввода-вывода – и через специализированные слова драйверов устройств, и путем прямого доступа к регистрам ввода-вывода.

Интерфейс прямого доступа к регистрам позволяет работать с устройствами ввода-вывода с эффективностью программы, написанной в машинных кодах. При этом AFS скрывает реальные адреса регистров ввода-вывода, что существенно облегчает разработку программ и перенос исходных текстов внутри модельного ряда.

При использовании прямого доступа к регистрам AFS обеспечивает контроль за корректностью адресации, что позволяет избежать случайных ошибок.

В стандартном языке Форт не предусмотрено специальных средств для программирования устройств ввода-вывода. Все описываемые далее слова и возможности имеются только в Advanced Forth.

Мы не можем подробно рассказать здесь о всех устройствах ввода-вывода, имеющихся в Форт-машинах разных моделей из-за огромного объема информации. Далее мы рассмотрим основные приемы программирования распространенных устройств и базовые слова, имеющиеся во всех системах.

Для детального изучения устройств ввода-вывода и методов их программирования вам потребуется получить специальные выпуски "Заметок по применению", которые можно найти на нашем сайте в разделе поддержки пользователей.

12.1. Порты ввода-вывода общего назначения

Дискретные (они же цифровые) порты ввода-вывода – самое простое средство связи вычислительной машины с внешним миром. Дискретный порт используется для приема или передачи значений логического нуля или единицы.

Обычно дискретные порты объединяют в группы по 8 или 16, чтобы облегчить программирование. Такие группы называют параллельный порт ввода-вывода (PIO) или порт ввода-вывода общего назначения (GPIO).

В большинстве современных микроконтроллеров и специализированных периферийных микросхем поддерживается как независимое управление каждой линией параллельного порта, так и групповое управление всеми разрядами.

Параллельные порты обычно обозначаются буквами (PA, PB, PC и т.д.), а линии ввода-вывода этого порта - цифрами. Обозначение PA0 соответствует отдельной линии порта PA с номером 0. Номер линии соответствует номеру бита в регистрах данных этого порта.

В современных однокристальных системах множество встроенных устройств разделяют между собой одни и те же выводы (контакты или "ножки") корпуса микросхемы. Какому устройству будет принадлежать тот или иной вывод, какие функции он будет выполнять – все это решается настройками соответствующих регистров конфигурации.

По умолчанию, контакты микросхемы принадлежат дискретным портам ввода-вывода (PIO). Наименование выводов микроконтроллера по умолчанию совпадает с наименованиями линий PIO. Чтобы получить возможность передачи сигналов других устройств, необходимо предварительно изменить настройки конфигурации PIO.

12.1.1. Настройка конфигурации PIO. Слова PIO-CFG! и PIO-CFG@

Настройка параметров отдельных выводов микроконтроллера требует большого количества управляющих битов. Эти биты находятся в соответствующих регистрах аппаратуры, каждый из которых отвечает за свой параметр.

Слово PIO-CFG! (xx param port --) предназначено для записи данных конфигурации в определенный регистр настроек конкретного порта. Мы ввели нестандартные обозначения в стековую диаграмму, чтобы было нагляднее.

Ячейка на вершине стека содержит номер порта, к которому мы обращаемся для изменения конфигурации. Порту PA соответствует номер 0, PB – номер 1 и так далее.

Прежде чем приступить к программированию, рекомендуем изучить характеристики используемой Форт-машины на предмет наличия или отсутствия тех или иных портов. В системе не обязательно должны присутствовать все порты, некоторые могут быть пропущены.

Обращаться к пропущенным портам не рекомендуется. Если вы укажете номер порта 2 (то есть PC), а его в системе нет, обращение будет переадресовано к последнему известному порту в списке, обычно к PF.

Следующая ячейка стека (мы ее обозначили param) содержит номер параметра конфигурации. Номера параметров конфигурации портов ввода-вывода представлены в таблице. При появлении новых вариантов аппаратуры могут добавиться дополнительные параметры и их номера.

Таблица 12.1 Номера параметров конфигурации портов ввода-вывода

Номер	Название	Описание
0	PIO-MODE	Режим работы каждого дискретного вывода порта
1	OUT-TYPE	Тип для выводов порта, настроенных как выход
2	OUT-SPEED	Скорость изменения состояния выходных сигналов
3	PULL-UP-DOWN	Настройки "подтягивающих" резисторов для каждого вывода
4	FUNCTION-LOW	Код подключенного встроенного устройства, выводы 7..0 порта
5	FUNCTION-HIGH	Код подключенного встроенного устройства, выводы 15..8 порта

Последним из стека извлекается двойное число (xx), содержащее значения устанавливаемого параметра для выводов порта. Большинство параметров требуют более 1 бита для каждого вывода, поэтому одной ячейки (16 бит) не хватает.

Слово PIO-CFG@ (param port -- xx) позволяет прочитать текущее значение определенного параметра порта. Ему также требуется номер порта и номер параметра, а возвращает оно двойное число, содержащее значение настроек. Далее мы подробно рассмотрим назначение и формат этих значений.

12.1.1.1 Параметр PIO-MODE

Параметр PIO-MODE содержит битовые значения режимов работы для каждого дискретного вывода порта. Эти режимы приведены в таблице.

Таблица 12.2 Битовые значения режимов работы выводов порта PIO-MODE

Значение	Название	Описание режима работы
0	INPUT	Вывод порта используется как дискретный цифровой вход
1	OUTPUT	Вывод порта используется как дискретный цифровой выход
2	FUNCTION	Вывод выполняет функцию ввода-вывода встроенного устройства
3	ANALOG	Вывод подключен к мультиплексору аналоговых сигналов АЦП

Четыре возможных значения режимов занимают два бита для каждого вывода порта. В передаваемой через стек паре ячеек эти биты занимают места по принципу "младшие выводы – в младшие биты".

В двойном числе параметра младшая ячейка содержит биты режима для выводов от 0 до 7, старшая ячейка – для выводов от 8 до 15. Младшие выводы при вводе числа находятся справа.

Мы уже упоминали настройку режимов порта, когда изучали системы счисления в Форте (см. 3.2.3. Использование неординарной системы счисления).

При таком размещении битов в ячейке исключительно удобно использовать систему счисления с основанием 4. Например, нам надо установить следующие режимы работы: для PA0..PA3 – аналоговые измерения, PA4..PA7 – дискретные входы, PA8..PA11 – функции встроенных устройства ввода-вывода, PA12..PA15 – дискретные выходы.

Используя таблицу значений режимов и основание счисления 4, мы получаем простой и красивый код (пока не вводите его!):

```
4 RADIX!
11112222.00003333 NUL NUL PIO-CFG!
DECIMAL
```

Обратите внимание, мы указали Форт-системе на ввод двойного значения с помощью точки в числе. Каждый разряд этого числа соответствует режиму работы одного из выводов порта PA (слева PA15, справа PA0).

Почему не надо вводить этот код? Как только будет выполнена вторая строка, вы потеряете связь с Форт-машиной. Дело в том, что интерфейс AFM-Link, который обеспечивает связь с терминалом пользователя, тоже использует выводы порта PA. Если их перепрограммировать, связь будет прервана.

Прежде чем программировать конфигурацию портов ввода-вывода, уточните, какие выводы задействованы для AFM-Link. В нашем примере с моделью AFMnano-M0 это будут выводы PA14 и PA13. При изменении каких-либо параметров порта PA вы должны сохранять настройки для этих выводов. Сделать это можно разными способами.

Самый простой – посмотреть в документации или с помощью слова PIO-CFG@, какие значения параметров конфигурации уже присвоены выводам порта.

Например, выполнив

```
4 RADIX!
NUL NUL PIO-CFG@ UD.
```

мы получим

```
210000000000000 Ok
```

Подсчитав разряды справа налево (именно так, потому что незначащие нули слева не выводятся) мы обнаружим, что для PA13 установлен режим 1 (выход), а для PA14 – режим 2 (функция устройства ввода-вывода).

Используйте эти значения при установке параметров других выводов. С учетом использования PA14 и PA13 для AFM-Link, приведенный выше пример должен выглядеть так:

```
4 RADIX!
12112222.00003333 NUL NUL PIO-CFG!
DECIMAL
```

Второй способ более сложный и заключается в использовании битовых масок и логических операторов. Мы используем его в одном из примеров при программировании устройств ввода-вывода.

На самом деле, ничего затруднительного в настройке портов ввода-вывода нет, так как в подавляющем большинстве случаев все параметры конфигурации известны заранее и программируются один раз при старте программы.

Чаще всего используется одна из типовых конфигураций (см. стр. 127), а дополнительно изменяются настройки лишь двух-трех выводов микроконтроллера, если это необходимо.

12.1.1.2 Параметр OUT-TYPE

Параметр OUT-TYPE содержит битовые значения типа выхода для каждого вывода порта. Типов выходов всего два: push-pull (двухтактный) и open-drain (с открытым стоком). Соответственно, требуется всего один бит, чтобы описать все варианты.

Для 16 выводв порта достаточно одной ячейки для передачи параметра OUT-TYPE через стек, поэтому старшая ячейка передается со значением 0.

Каждый бит младшей ячейки соответствует типу выхода для одного из выводов порта. Если бит равен 0, выход типа push-pull, если бит равен 1 – выход open-drain.

Параметр влияет на работу тех выводов, для которых установлен режим дискретного выхода или функции встроенного устройства ввода-вывода, а уже само устройство использует его как выход. Для других режимов работы установка этого параметра не имеет значения.

Добавим к предыдущему примеру установку для выхода PA15 типа "открытый сток", учитывая при этом уже заданные параметры для AFM-Link.

Чтобы не связываться с большими числами при установке этого параметра, можно использовать трюк с битовым сдвигом:

```
1 NUL PIO-CFG@
1 15 <<< OR
1 NUL PIO-CFG!
```

Первая строка возвращает на вершине стека текущие значения параметра для всех выводов порта PA, а выполнение последовательности

```
1 15 <<<
```

помещает на вершину стека ячейку, в которой только бит 15 установлен в 1. Последующее выполнение слова **OR** складывает два значения по ИЛИ, что дает нам новое значение параметра. Третьей строкой мы записываем новое значение типа выхода PA15.

12.1.1.3 Параметр OUT-SPEED

Параметр OUT-SPEED позволяет регулировать скорость переключения для каждого вывода порта, настроенного для работы в режиме выхода или функции встроенного устройства ввода-вывода, использующего этот вывод как выход.

Формат этого параметра аналогичен PIO-MODE – каждому выводу назначается значение из двух битов. Значение 0 или 2 соответствует низкой скорости переключения, 1 – средней, 3 – высокой скорости.

От скорости переключения зависит максимальная допустимая частота выходного сигнала, уровень создаваемых помех и потребляемая мощность.

Подробности применения этого параметра можно найти в техническом описании используемого вами модуля AFM.

12.1.1.4 Параметр PULL-UP-DOWN

В цифровой электронике часто возникает необходимость зафиксировать определенный логический уровень на входе, когда источник сигнала отключен. Самый простой способ решения этой задачи – подключение "подтягивающего" резистора между входом и питанием (называется pull-up) или входом и "землей" (соответственно, pull-down).

В большинстве современных микроконтроллеров такие резисторы встроены в микросхему вместе с ключами, управляя которыми можно подключать и отключать резисторы.

Параметр PULL-UP-DOWN позволяет управлять "подтягивающими" резисторами на выводах порта. Эта настройка действует не только для входов, но и для выходов и при подключении встроенных устройств ввода-вывода. Для аналоговых входов АЦП резисторы лучше все-таки отключать.

Формат этого параметра аналогичен рассмотренным ранее PIO-MODE и OUT-SPEED.

Для каждого вывода выделены два бита. Все варианты управления резисторами перечислены в таблице.

Таблица 12.3 Значения параметра настройки порта PULL-UP-DOWN

Значение	Название	Описание режима работы
0	NO-PULL	"Подтягивающие" резисторы отключены
1	PULL-UP	Подключен резистор между выводом и питанием (3.3V)
2	PULL-DOWN	Подключен резистор между выводом и "землей" (GND)
3		Значение зарезервировано, не использовать

Такие подробности, как сопротивление резисторов или токи нагрузки, можно найти в техническом описании используемого вами модуля AFM.

Для примера установим для входов PA7 и PA6 режим pull-up, а для PA4 и PA5 – режим pull-down.

Сделаем мы это уже проверенным методом, используя таблицу значений и основание счисления 4:

```
4 RADIX!
3 NUL PIO-CFG@ DROP
11220000 3 NUL PIO-CFG!
DECIMAL
```

Обратите внимание – сначала мы получили текущее значение параметра, но удалили младшую ячейку значения. В старшей ячейке содержатся значения для выводов PA14 и PA13 (AFM-Link). Затем мы ввели новые значения для PA7-PA4 (младшая ячейка) и записали новое значение параметра конфигурации.

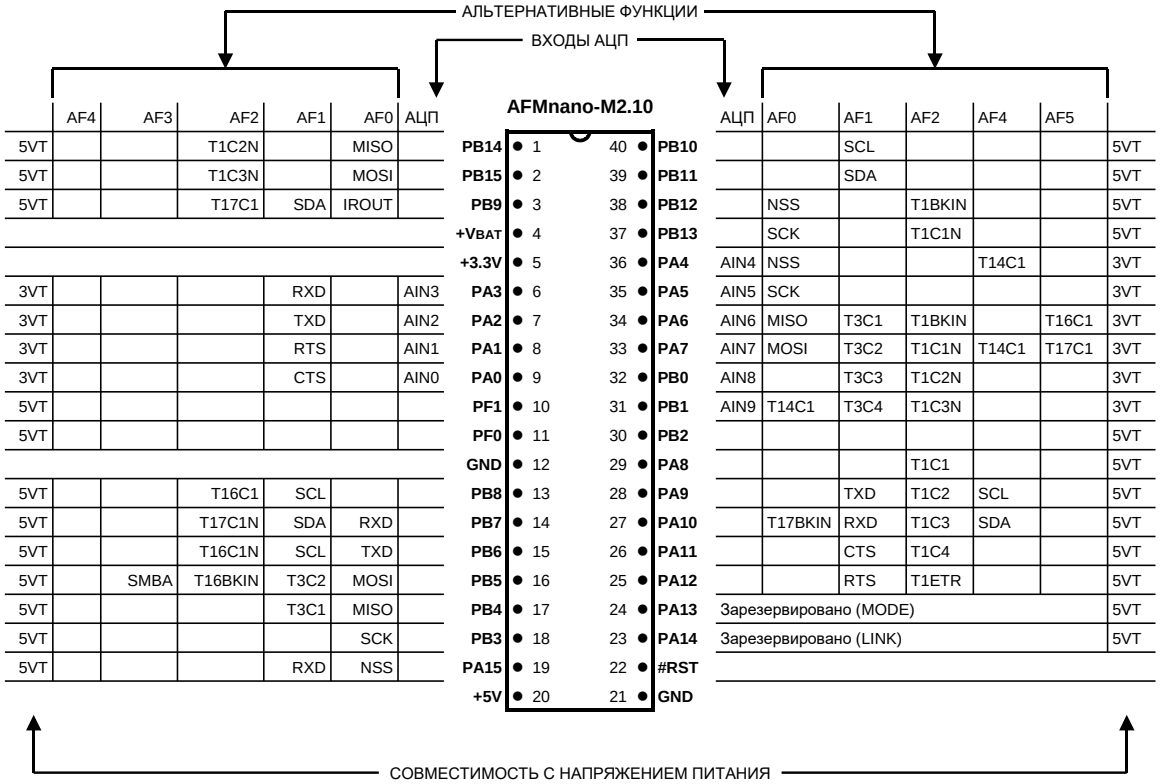
12.1.1.5 Параметры FUNCTION-LOW и FUNCTION-HIGH

Как вы помните, параметр PIO-MODE предусматривает установку вывода микросхемы в режим FUNCTION для подключения одного из встроенных устройств ввода-вывода. Какое именно устройство будет подключено (и какая именно электрическая цепь), определяется параметром FUNCTION-LOW или FUNCTION-HIGH.

В современных микроконтроллерах может иметься значительное количество встроенных устройств, поэтому под код устройства выделено 4 бита, что дает нам 16 возможных значений. Эти значения также называются кодами альтернативных функций (основная функция – порт ввода-вывода, для нее код не требуется).

Поскольку для настройки 16 выводов портов потребуется 64 бита, этот параметр разделен на два управляющих регистра – старший и младший. Соответственно, применяется два различных номера параметров для установки.

Параметр FUNCTION-LOW устанавливает коды подключенных устройств для выводов порта с 0 по 7, параметр FUNCTION-HIGH – для выводов порта с 8 по 15. Как всегда, младшие биты ячеек соответствуют выводам с меньшими номерами.



	AF4	AF3	AF2	AF1	AF0	АЦП				АЦП	AF0	AF1	AF2	AF4	AF5	
5VT			T1C2N		MISO		PB14	1	40	PB10			SCL				5VT
5VT			T1C3N		MOSI		PB15	2	39	PB11			SDA				5VT
5VT			T17C1	SDA	IROUT		PB9	3	38	PB12		NSS		T1BKIN			5VT
							+VBAT	4	37	PB13		SCK		T1C1N			5VT
							+3.3V	5	36	PA4	AIN4	NSS			T14C1		3VT
3VT				RXD		AIN3	PA3	6	35	PA5	AIN5	SCK					3VT
3VT				TXD		AIN2	PA2	7	34	PA6	AIN6	MISO	T3C1	T1BKIN		T16C1	3VT
3VT				RTS		AIN1	PA1	8	33	PA7	AIN7	MOSI	T3C2	T1C1N	T14C1	T17C1	3VT
3VT				CTS		AIN0	PA0	9	32	PB0	AIN8		T3C3	T1C2N			3VT
5VT							PF1	10	31	PB1	AIN9	T14C1	T3C4	T1C3N			3VT
5VT							PF0	11	30	PB2							5VT
							GND	12	29	PA8				T1C1			5VT
5VT			T16C1	SCL			PB8	13	28	PA9			TXD	T1C2	SCL		5VT
5VT			T17C1N	SDA	RXD		PB7	14	27	PA10		T17BKIN	RXD	T1C3	SDA		5VT
5VT			T16C1N	SCL	TXD		PB6	15	26	PA11			CTS	T1C4			5VT
5VT		SMBA	T16BKIN	T3C2	MOSI		PB5	16	25	PA12			RTS	T1ETR			5VT
5VT				T3C1	MISO		PB4	17	24	PA13	Зарезервировано (MODE)						5VT
5VT					SCK		PB3	18	23	PA14	Зарезервировано (LINK)						5VT
5VT				RXD	NSS		PA15	19	22	#RST							
							+5V	20	21	GND							

Рисунок 12.1 Альтернативные функции выводов модуля AFMnano-M2.10

Коды встроенных устройств (альтернативных функций) приводятся в технической документации, прилагаемой к модулю AFM, поскольку определяются особенностями применяемого микроконтроллера и могут меняться от модели к модели.

Для примера приведем альтернативные функции выводов PIO для модели AFMnano-M2.10 (см. Рисунок 12.1). Обозначения альтернативных функций указаны в виде AFx, где x – числовой код встроенного устройства ввода-вывода микроконтроллера.

На рисунке присутствуют не все возможные коды, поскольку большинство из них зарезервированы для более сложных микросхем той же серии.

Если вы установите неиспользуемый код, вывод микросхемы окажется в неопределенном состоянии, что может привести к отказу.

Для удобства программирования четырехбитных полей параметров лучше использовать шестнадцатеричную систему счисления.

Для примера переключим выводы PA6 и PA7 на таймер-счетчик T3. Для дальнейшей работы с выводами таймера T3C1 и T3C2 необходимо установить соответствующий режим в параметре PIO-MODE.

```
HEX 1100 NUL 4 NUL PIO-CFG!
```

Описание встроенных устройств ввода-вывода вы найдете в техническом руководстве используемого модуля AFM.

12.1.2. Чтение входных данных PIO. Слово PIO-IN

После того, как вы сконфигурировали некоторые выводы PIO как дискретные цифровые входы и выходы, вам необходимо каким-то образом получать данные о состоянии этих электрических цепей.

Слово PIO-IN (port -- x) получает на вершине стека номер порта, а возвращает состояние входов порта. Каждый бит полученного числа соответствует одному дискретному входу.

Слово **PIO-IN** возвращает актуальное значение логического сигнала на соответствующем выводе микросхемы, как для входов, так и для выходов.

Если вы используете для опытов платы, имеющие кнопки или датчики, подключенные к PF0 и PF1, вы можете получать их состояние:

```
5 PIO-IN U.
```

Выводимый результат будет зависеть от состояния кнопок или датчиков в момент выполнения слова **PIO-IN**.

12.1.3. Управление выходами PIO. Слова PIO-OUT! и PIO-OUT@

Если вывод PIO сконфигурирован как выход, его состояние управляется специальным регистром выходных данных. Каждый бит этого регистра соответствует одному выходу PIO.

Запись логической 1 в бит регистра приводит к появлению сигнала высокого логического уровня на соответствующем выходе. Для сброса выходного сигнала в низкий уровень в бит регистра надо записать 0.

Если соответствующий вывод не сконфигурирован как цифровой выход, значение в соответствующем ему бите регистра выходных данных никак не влияет на актуальное состояние выходного сигнала.

Для записи значения в регистр выходных данных используется слово PIO-OUT! (x port --). На вершине стека слово получает номер порта, а под ним – значение для записи в регистр.

Узнать текущее значение, записанное в регистр выходных данных, можно с помощью слова PIO-OUT@ (port -- x). Слово получает номер порта на вершине стека и возвращает значение из регистра.

Установим выход PF0 в 1, а остальные сбросим в 0:

```
1 5 PIO-OUT!
```

Проверим, что записалось в регистр выходных данных порта PF:

```
5 PIO-OUT@ U.
1 Ok
```

Если теперь выполнить слово **PIO-IN**, то с большой вероятностью вы увидите другое значение. Слово **PIO-IN** возвращает актуальное состояние всех сигналов на выводах PIO микросхемы, а содержимое регистра выходных данных влияет только на выходы.

По умолчанию вывод PF0 сконфигурирован как вход, поэтому изменение бита в регистре выходных данных никак не влияет на возвращаемое значение.

12.1.4. Слово PIO-BIT-SR

Слово **PIO-OUT!** изменяет состояние всех выходов порта одновременно. Это не всегда удобно, так как требует нескольких операций, если требуется управлять только отдельными выходами, не затрагивая остальные.

Для управления выходами порта на уровне отдельных линий предназначено слово PIO-BIT-SR (x1 x2 port --). Слово получает номер порта на вершине стека, за ним следуют два числа, управляющие синхронной установкой и сбросом логических уровней на выходах порта.

Верхнее число (x2) отвечает за установку в 1 битов регистра выходных данных. Установка производится только для тех битов, значение которых в x2 равно 1. Биты, значение которых равно 0, в регистре данных не изменяются.

Аналогично нижнее число (x1) отвечает за сброс в 0 битов регистра. Сбрасываются только те биты, значение которых в x1 равно 1. Для битов равных 0 состояние в регистре данных не изменяется.

Для примера сбросим в 0 ранее взведенный бит PF0 и установим в 1 бит PF1:

```
1 2 5 PIO-BIT-SR
```

Проверим, что получилось:

```
5 PIO-OUT@ U.
2 Ok
```

Сброс и установка отдельных битов при выполнении слова **PIO-BIT-SR** производятся одновременно.

Если вы используете для опытов плату, на которой имеются светодиоды, подключенные к выводам PF0 и PF1 (если нет, вы можете подключить их самостоятельно), сконфигурируйте эти выводы порта как выходы:

```
0.5 NUL 5 PIO-CFG!
```

Теперь вы можете включать и выключать светодиоды, используя приведенные выше примеры.

При самостоятельном подключении светодиодов к выводам PFx, соединяйте их с общим проводом (GND) последовательно с резистором сопротивлением не менее 330 ом.

12.1.5. Типовые конфигурации PIO

Ранее мы упоминали о системном запросе конфигурации HW_CONFIG (см. стр. 127). Фактически, при выполнении этого запроса происходит загрузка регистров управления PIO значениями, заранее записанными в памяти системы. Использование типовых конфигураций значительно упрощает разработку программ.

Типовые конфигурации и присвоенные им номера описываются в технической документации на Форт-машину.

12.1.6. Автоматическая инициализация PIO

При использовании AFM в качестве управляющей машины конфигурация всех выводов микросхемы микроконтроллера должна производиться сразу после старта системы. Настройка конфигурации выводов PIO с помощью отдельных команд порождает довольно громоздкий код, который отнимает часть памяти программы, но выполняется всего лишь раз.

В Advanced Forth System предусмотрена автоматическая загрузка параметров конфигурации PIO перед запуском программы пользователя.

Когда вы сохраняете свою программу для последующего автономного выполнения, AFS записывает в постоянную память параметры настройки конфигурации PIO, включая состояние регистров выходных данных портов PIO.

При старте в автономном режиме AFS загружает эти значения в соответствующие регистры, после чего запускает программу пользователя.

Автоматическая инициализация PIO выполняется быстрее и занимает меньше памяти программы, чем выполнение тех же операций с помощью отдельных слов. При этом гарантируется корректная установка всех выводов в требуемое состояние.

Из соображений безопасности автоматическая инициализация PIO не производится при восстановлении сохраненного сеанса программирования.

При сохранении программы пользователя записываются текущие значения регистров конфигурации. Перед записью установите необходимые состояния регистров выходных данных и другие параметры в состояние, требуемое при старте.

12.2. Аналогово-цифровой преобразователь

Аналогово-цифровой преобразователь (АЦП, ADC), как следует из названия, переводит напряжение на входе в цифровое значение, пригодное для дальнейшей обработки программой.

В модулях AFMnano используется блок АЦП с мультиплексором входных сигналов. Общее число доступных аналоговых входов зависит от модели модуля. Входы АЦП обозначаются AINx, где x – номер входа.

Перед началом работы с АЦП необходимо подключить используемые аналоговые входы мультиплексора к выводам микроконтроллера с помощью установки режима PIO или выбора типовой конфигурации.

Кроме внешних сигналов, АЦП измеряет температуру кристалла микросхемы и опорное напряжение преобразователя (если такая возможность предусмотрена в микроконтроллере). Эти данные используются для калибровки самого АЦП и улучшения точности измерения, а также доступны для выполняемой программы.

12.2.1. Инициализация АЦП. Слово ADC-INIT

Работа с АЦП начинается с инициализации словом ADC-INIT (mode dev --).

На вершине стека передается системный номер АЦП (параметр dev). В простых Форт-машинах имеется один блок АЦП с системным номером 0, но в более сложных может быть несколько АЦП.

Следующий параметр (mode) определяет режим работы устройства. Вы можете использовать значение ноль для установки режима АЦП. Другие значения используются в системах с несколькими АЦП и специальными функциями.

При выполнении слова ADC-INIT драйвер выполняет включение блока АЦП, настройку частоты синхронизации и первичную калибровку.

12.2.2. Управление АЦП. Слово ADC-CTRL

Для управления АЦП предназначено слово ADC-CTRL (param n dev --).

На вершине стека находится номер устройства АЦП, под ним – номер запроса управления, далее – параметр запроса.

Допустимые номера запросов приведены в таблице.

Таблица 12.4 Запросы управления АЦП

Номер	Запрос	Описание выполняемого действия
0	ADC_ONOFF	Разрешение и запрещение работы АЦП
1	ADC_REFUPD	Обновление значений опорного напряжения и температуры кристалла

12.2.2.1 Запрос ADC_ONOFF

После инициализации АЦП находится в активном рабочем состоянии. Если возникнет необходимость повторной инициализации АЦП, например, для рекалибровки, сделать это можно только после запрещения работы АЦП.

Запрещение и разрешение работы АЦП осуществляется запросом ADC_ONOFF.

Стековая диаграмма запроса: ADC-CTRL (param 0 dev --) .

Параметр запроса определяет, в какое состояние переводится устройство и драйвер.

Значение 0 запрещает работу блока АЦП и переводит его в пассивное состояние. В этом состоянии возможно изменение параметров и рекалибровка АЦП (повторная инициализация).

Значение 1 разрешает работу АЦП (переводит в активное состояние) без повторной инициализации.

12.2.2.2 Запрос ADC_REFUPD

Чтобы обновить данные о текущем значении опорного напряжения и температуры кристалла микросхемы, используйте запрос ADC_REFUPD.

Стековая диаграмма запроса: ADC-CTRL (param 1 dev --) .

Параметр запроса не используется в простых Форт-машинах и должен быть задан нулем.

В процессе инициализации АЦП также обновляет значения опорного напряжения и температуры.

12.2.3. Получение результатов измерений. Слово ADC-CHAN

Произвести измерение напряжения в одном канале АЦП и получить результат можно с помощью слова ADC-CHAN (n dev -- x).

На входе слову требуется номер канала АЦП и номер самого АЦП, возвращается результат измерения.

В качестве примера произведем замер в канале AIN0:

```
NUL NUL ADC-CHAN .
```

Данные возвращаются в условных единицах шкалы АЦП. Для пересчета в значения измеряемого напряжения необходимо знать количество разрядов данных АЦП. Эта характеристика указана в документации на модуль AFM.

12.2.4. Слово ADC-SCAN

Для получения результатов измерения нескольких каналов АЦП используется слово ADC-SCAN (a-addr mask dev --). Кроме номера устройства требуются еще два входных параметра. Параметр mask - битовая маска тех входов мультиплексора, напряжение на которых вы хотите измерить. Для выбора входа соответствующий бит должен быть установлен в 1. Младший бит соответствует входу AIN0.

Третий параметр содержит адрес буфера в памяти данных программы, куда будут сохранятся результаты измерений.

Адрес буфера должен быть выровнен на границу ячейки, а сам буфер иметь достаточный размер. Каждому каналу АЦП в буфере соответствует одна ячейка.

Драйвер АЦП производит переключение входов и измерение в автоматическом режиме, от входа с наименьшим номером в сторону увеличения номера. Результаты измерений записываются в буфер. Данные записываются последовательно, в порядке выполнения измерений.

Предположим, что мы используем для измерений 8 каналов.

Напомним, как создать именованный массив ячеек нужного размера:

```
ALIGN CREATE BufADC 8 CELLS ALLOT
```

Слово **BufADC** возвращает адрес первой ячейки, он же – адрес буфера. Передадим это значение драйверу АЦП и выполним сканирование 8 входов:

```
HEX
BufADC FF NUL ADC-SCAN
```

Если в битовой маске входов имеются пропуски, данные в буфер все равно поступают непрерывно. Другими словами, если вы выполните

```
BIN
BufADC 10101 NUL ADC-SCAN
```

в буфере АЦП будут размещены данные трех измерений. В первой ячейке – для канала AIN0, во второй – для канала AIN2, в третьей – для канала AIN4.

Драйвер АЦП попытается провести измерения для всех указанных в маске каналов. Если вы создали буфер недостаточного размера, последние измерения могут быть записаны в случайные ячейки и испортят другие данные.

Будьте внимательны при определении размеров буфера и указании маски сканируемых входов.

12.2.5. Состояние АЦП. Слово ADC-STAT

При работе с АЦП возникает необходимость получения данных о его состоянии. Например, после запуска сканирования всех каналов необходимо знать момент его завершения.

Слово ADC-STAT (n dev – x) предназначено для получения различной информации о состоянии АЦП. Кроме номера устройства слову передается номер запроса (n).

Возвращаемые значения, в зависимости от номера запроса, приведены в таблице. Если вы передадите слову ADC-STAT некорректный номер запроса, в стеке вернется ноль.

Таблица 12.5 Номера запросов и возвращаемые данные состояния АЦП

Номер	Запрос	Описание возвращаемых данных
0	ADC_BUSY	Если АЦП свободен, возвращается ноль
1	ADC_TEMP	Температура кристалла микросхемы в градусах Кельвина
2	ADC_VREF	Значение опорного напряжения, данные АЦП без обработки

12.2.5.1 Запрос ADC_BUSY

Запрос ADC_BUSY позволяет определить момент завершения цикла сканирования входов и сохранения результатов измерения.

Если АЦП занят выполнением измерений, на вершине стека вернется значение "истина" (т.е. не ноль).

Если АЦП готов к выполнению следующего измерения, возвращается ноль.

12.2.5.2 Запросы ADC_TEMP и ADC_VREF

Как мы уже упоминали, один из каналов АЦП может быть подключен к термодатчику, измеряющему температуру кристалла микропроцессора. Еще один канал используется для точного измерения опорного напряжения.

Оба канала не связаны с мультиплексором входных сигналов и не требуют отдельного буфера для хранения данных. Полученные значения используются для повышения точности измерения в основных каналах.

Для обновления значений температуры и опорного напряжения используется запрос ADC_REFUPD слова **ADC-CTRL**. При выполнении запроса обновляется внутренние переменные драйвера АЦП.

Для чтения полученного значения температуры кристалла используйте запрос ADC_TEMP слова **ADC-STAT**.

Для примера получим температуру микросхемы в градусах Цельсия.

Введите:

```
1 NUL ADC-STAT 273 - .
```

Получаем

```
23 Ok
```

У вас может получиться другое значение. Оно зависит от температуры в помещении, условий охлаждения модуля и других факторов.

Также следует учесть, что встроенный в микросхему термодатчик не предназначен для точных измерений, его задача – отслеживать значительные изменения температуры, поэтому погрешность измерения может достигать 2-3 градусов.

Запрос ADC_VREF возвращает текущее значение опорного напряжения, используемого АЦП для преобразования. Значение возвращается в "сыром" виде (как его принимает АЦП) и может быть использовано при обработке данных для определения погрешности измерения.

12.3. Последовательный (синхронный) периферийный интерфейс (SPI)

Последовательный синхронный интерфейс периферийных устройств используется для скоростного обмена с внешними устройствами, расположенными на небольшом удалении от процессора, а также для межпроцессорного обмена.

Встроенный в микроконтроллер блок синхронного периферийного интерфейса называется SPI (Serial Peripheral Interface). Для подключения внешних устройств не требуется каких-либо дополнительных формирователей сигналов, входы и выходы микросхем соединяются непосредственно.

Для обмена данными интерфейс имеет следующие электрические цепи:

SCK (Serial Clock) – выход импульсов синхронизации ведущего устройства, вход синхронизации ведомого;

MOSI (Master Output/Slave Input) – выход данных ведущего устройства, вход данных ведомого устройства;

MISO (Master Input/Slave Output) – вход данных ведущего устройства, выход данных ведомого устройства;

-CS (Chip Select) – выход выбора микросхемы ведущего устройства, вход выбора ведомого (активный уровень низкий).

Последнее обозначение имеет синоним NSS (Negative Slave Select), который используется в документации AFM, если такое обозначение принято у изготовителя микросхемы микроконтроллера.

Интерфейс SPI поддерживается практически всеми производителями микросхем и развивается на протяжении многих десятилетий. Мы не можем обстоятельно рассказать здесь о всех вариантах применений SPI и тонкостях его программирования, для этого существуют специальные руководства. Остановимся на обзоре интерфейса драйвера SPI и описании основных режимов работы.

12.3.1. Инициализация SPI. Слово SPI-INIT

Слово SPI-INIT (data speed mode dev --) выполняет начальную инициализацию блока и драйвера SPI.

На вершине стека передается системный номер SPI (параметр dev).

Следующий параметр (mode) определяет режим работы устройства.

Ключевыми параметрами режима работы SPI являются размер передаваемых данных и значения полярности и фазы сигнала синхронизации. Устройство SPI может быть ведущим (выдает сигналы выбора устройства и синхронизации) или ведомым (получает сигналы управления от ведущего).

Под полярностью сигнала синхронизации понимается его логическое состояние в момент, когда происходит изменение сигнала данных (появляется значение следующего бита данных).

Фаза сигнала синхронизации определяет, по какому изменению сигнала синхронизации происходит захват сигнала данных.

Эти параметры всегда указываются в описании микросхем (устройств), которые вы будете подключать к вашему модулю AFM. Вам нужно только правильно установить их при инициализации SPI.

Код режима работы задается отдельными битами.

Бит 0 кодирует фазу сигнала синхронизации. Бит 1 кодирует полярность сигнала синхронизации.

Если бит 2 кода режима установить в 1, устройство инициализируется как ведущее. При нулевом значении бита 2 устройство будет ведомым.

Бит 7 устанавливается в 1, если данные должны передаваться и приниматься начиная с младшего бита. Если бит 7 кода режима равен 0, первым передается и принимается старший бит данных.

Все остальные биты параметра mode должны быть равны 0.

Для представления кода режима SPI удобно использовать шестнадцатеричную систему. Получить необходимый код режима можно простым сложением шестнадцатеричных чисел:

Код режима SPI = (старший/младший бит первый) * 80_{16} + (бит ведущий/ведомый) * 4 + (полярность синхронизации) * 2 +

+ (фаза синхронизации) * 1

Предположим, нам нужен код для передачи данных начиная с младшего бита, в режиме ведущего, полярность синхронизации 1, фаза равна 0.

Вычисляем:

```
HEX 80 4 + 2 + .
```

Получаем код режима:

```
86 Ok
```

Параметр speed определяет скорость обмена данными в последовательном синхронном интерфейсе.

Сигнал синхронизации данных SPI получает из сигнала синхронизации процессора. Частота синхронизации процессора понижается в делителе частоты SPI. Вы можете указать коэффициент деления частоты из ряда 2^N, где N – целое число от 1 до 8. Другими словами, частота синхронизации SPI равна частоте процессора, деленной на 2, 4, 8 …128 или 256.

Параметр speed должен содержать значение N-1. Для деления частоты процессора на 2 укажите параметр 0, для деления на 4 – параметр равен 1 и так далее. Максимальное значение параметра равно 7 (деление на 256).

Если в качестве параметра задать значение, превышающее 7, новый коэффициент не будет установлен (будет установлено деление на 2).

Если вам требуется изменить частоту синхронизации процессора, делайте это до установки частоты синхронизации SPI.

Параметр data определяет размер передаваемых и принимаемых данных.

Поскольку SPI передает данные по одному биту, сопровождая каждый бит сигналом синхронизации, длина кадра данных может быть любой от 4 до 16 бит.

Размер данных занимает младшие 4 бита параметра data. В стеке передается значение, на 1 меньше размера данных (значение 3 – для 4 бит, значение 4 – для 5 бит и т.д.). Если в качестве размера данных вы укажете недопустимое значение (0, 1, 2 или больше 16), размер будет установлен случайным образом, что приведет к нарушениям в работе блока SPI.

12.3.2. Управление SPI. Слово SPI-CTRL

Слово SPI-CTRL (param n dev --) предназначено для управления блоком SPI и его драйвером.

На вершине стека передается номер устройства, под ним – номер запроса управления, ниже – параметр запроса.

В простых системах используется только один управляющий запрос – запрещение и разрешение, но в более сложных системах могут присутствовать и другие.

Таблица 12.6 Запросы управления SPI

Номер	Запрос	Описание выполняемого действия
0	SPI_ONOFF	Разрешение и запрещение работы SPI

12.3.2.1 Запрос SPI_ONOFF

Изменение параметров SPI возможно только при пассивном состоянии устройства. Управление разрешением и запрещением работы SPI осуществляется запросом SPI_ONOFF.

Стековая диаграмма запроса: SPI-CTRL (param 0 dev --) .

Параметр запроса определяет, в какое состояние переводится устройство и драйвер.

Значение 0 запрещает работу блока SPI и переводит его в пассивное состояние. В этом состоянии возможно изменение параметров и режимов работы SPI (в том числе повторная инициализация).

Значение 1 разрешает работу SPI. В этом состоянии осуществляется обмен данными по синхронному интерфейсу. Изменения режимов запрещены.

Для начала работы с SPI (с системным номером ноль) введите:

```
1 NUL NUL SPI-CTRL
```

Если ранее вы установили режим работы, скорость обмена и размер данных, SPI готов к приему и передаче.

12.3.3. Прямой доступ к регистрам. Слова SPI-REG! и SPI-REG@

Интерфейс SPI имеет простое устройство, поэтому наиболее эффективным способом работы с ним является прямой доступ к его регистрам для записи и чтения передаваемых данных, битов управления и состояния.

Слово SPI-REG! (x reg dev --) записывает значение в указанный регистр указанного блока SPI. На вершине стека передается номер устройства, далее – номер регистра, а под ним – записываемое значение.

Слово SPI-REG@ (reg dev -- x) считывает значение указанного регистра заданного блока SPI. На вершине стека передается номер устройства, далее – номер регистра. Возвращается прочитанное значение регистра.

Номер регистра и его назначение указаны в технической документации на модуль AFMnano. При обращении к регистру с несуществующим номером запись не производится, а при чтении возвращается ноль.

12.4. Интерфейс Inter-integrated Circuit (I2C)

Интерфейс I2C широко применяется в самой разнообразной электронике – от промышленных систем управления до бытовой техники.

Физически интерфейс представляет собой общую шину, имеющую всего два проводника. По одному передаются данные (цепь обозначается SDA), по другому – синхронизация (цепь SCL). Оба проводника подтянуты к напряжению питания через резисторы, что позволяет нескольким устройствам одновременно изменять логические уровни в цепях интерфейса.

К интерфейсу может быть подключено большое количество устройств. Всем устройствам на шине присваивается свой уникальный адрес, который передается в начале каждого пакета данных.

Практически все современные микроконтроллеры имеют в своем составе хотя бы один блок I2C. Этот интерфейс весьма удобен для передачи небольших объемов информации на небольшом расстоянии и не требует значительных ресурсных затрат.

Контроллер (и драйвер) I2C может работать в режиме ведущего устройства на шине или ведомого. Режим ведущего устройства поддерживается во всех вариантах AFS и его мы рассмотрим далее. В режиме ведомого устройства I2C требует более сложной настройки и управления, возможны несколько вариантов реализации режима, доступные не во всех Форт-машинах и AFS.

12.4.1. Инициализация I2C. Слово I2C-INIT

Слово I2C-INIT (ownadr speed mode dev --) выполняет начальную инициализацию блока и драйвера I2C.

На вершине стека передается системный номер устройства I2C (параметр dev).

Следующий параметр (mode) определяет режим работы устройства.

Параметр speed определяет частоту синхронизации интерфейса, от которой напрямую зависит скорость обмена данными. Передаваемое в стеке значение должно содержать частоту в килогерцах.

Оба параметра могут быть заданы значением ноль, в этом случае I2C настраивается для работы в распространенной типовой конфигурации – частота синхронизации 100 кГц, режим ведущего устройства.

Возможность установки той или иной частоты синхронизации зависит от тактовой частоты процессора и особенностей реализации блока I2C в микроконтроллере. В простых контроллерах драйвер устанавливает одно из трех стандартных значений: 10 кГц, 100 кГц или 400 кГц.

Параметр ownadr содержит значение собственного адреса блока I2C на общей шине. Если работа в режиме ведомого не требуется, значение параметра можно задать нулем.

Подробное описание возможностей I2C вы найдете в технической документации на используемый модуль AFMnano.

Для большинства приложений достаточно инициализировать I2C значениями по умолчанию:

```
NUL NUL NUL NUL I2C-INIT
```

После выполнения этой строки устройство I2C с системным номером 0 будет настроено для работы ведущим с частотой синхронизации 100 кГц.

В качестве примера попробуем установить частоту синхронизации 400 кГц:

```
DECIMAL

NUL 400 NUL NUL I2C-INIT
```

Изменяя скорость обмена на шине I2C вы должны учитывать максимальную допустимую скорость работы самого медленного устройства. Многие устройства I2C работают только на частоте 100 кГц.

12.4.2. Управление I2C. Слово I2C-CTRL

Слово I2C-CTRL (param n dev --) предназначено для управления блоком I2C и его драйвером.

На вершине стека передается системный номер устройства, под ним – номер запроса управления, ниже – параметр запроса.

Допустимые номера запросов приведены в таблице.

Таблица 12.7 Запросы управления I2C

Номер	Запрос	Описание выполняемого действия
0	I2C_ONOFF	Включение и выключение I2C

12.4.2.1 Запрос I2C_ONOFF

Все настройки и изменения параметров I2C возможны только при пассивном состоянии устройства ввода-вывода. Управление разрешением и запрещением работы, а также полным выключением блока I2C осуществляется запросом I2C_ONOFF.

Стековая диаграмма запроса: I2C-CTRL (param 0 dev --) .

Параметр запроса определяет, в какое состояние переводится устройство и драйвер.

Значение 0 запрещает работу блока I2C и переводит его в пассивное состояние. В этом состоянии возможно изменение параметров и режимов работы I2C (в том числе повторная инициализация).

Значение 1 разрешает работу I2C. В этом состоянии осуществляется обмен данными по общей шине. Изменения режимов запрещены.

В большинстве случаев после инициализации I2C в режиме по умолчанию требуется только включить его:

```
NUL 100 NUL NUL I2C-INIT

1 NUL NUL I2C-CTRL
```

Теперь можно передавать и принимать данные в режиме ведущего устройства на шине I2C.

12.4.3. Обмен данными по I2C. Слово I2C-RECV

Наиболее распространенное применение I2C – связь с различными периферийными устройствами ввода-вывода, выполненными в виде микросхем. Эти устройства обычно работают в режиме ведомого, а ведущим для них будет блок I2C в нашей Форт-машине.

Для чтения данных из ведомого необходимо передать на шину его адрес, а затем сохранить принимаемые данные в буфер. Часто требуется предварительно передать ведомому дополнительную информацию, например, номер регистра или максимальное количество принимаемых байтов.

Таким образом, чтение данных из ведомого устройства может состоять из трех фаз: адресации, передачи параметров и собственно чтения.

Слово I2C-RECV (addr1 n1 addr2 n2 dest dev --) существенно упрощает эту задачу. Параметры addr1 и n1 – адрес и размер буфера для приема данных, параметры addr2 и n2 – адрес и размер предварительно передаваемых данных, параметр dest – адрес ведомого устройства на шине I2C. На вершине стека, как всегда, номер используемого блока I2C.

Если предварительная передача данных не требуется, просто замените адрес и длину данных константами **NUL** .

О результатах выполнения слова можно узнать с помощью запроса состояния (см. далее). Если никаких проблем не возникло, буфер приема содержит полученные данные.

12.4.4. Обмен данными по I2C. Слово I2C-SEND

Передача данных ведомому устройству выполняется несколько проще. Используйте слово I2C-SEND (addr n dest dev --) . Параметры аналогичны слову **I2C-RECV**, но их меньше. Надо только указать адрес начала передаваемых данных, их длину и адрес ведомого устройства на шине I2C. Далее вы найдете пример использования **I2C-SEND**.

12.4.5. Прямой доступ к регистрам. Слова I2C-REG! и I2C-REG@

Для более тонкой настройки и расширения возможностей использования интерфейса I2C предусмотрен прямой доступ к регистрам для записи и чтения.

Обратите внимание: регистры блока I2C имеют размер 32 бита. Для представления их содержимого требуется пара ячеек.

Слово I2C-REG! (xx reg dev --) записывает двойное число в указанный регистр указанного блока I2C. На вершине стека передается номер устройства, далее – номер регистра, а под ним – записываемое значение в паре ячеек.

Слово I2C-REG@ (reg dev -- xx) считывает значение указанного регистра указанного блока I2C. На вершине стека передается номер устройства, далее – номер регистра. Возвращается прочитанное значение регистра в паре ячеек.

Номер регистра и его назначение указаны в технической документации на модуль AFMnano. При обращении к регистру с несуществующим номером запись не производится, а при чтении возвращается ноль.

12.4.6. Практический пример программирования I2C

Попрактикуемся в программировании интерфейса I2C на примере программы для вывода информации на жидкокристаллический семисегментный индикатор. Один из наиболее распространенных контроллеров ЖКИ с интерфейсом I2C – микросхема PCF8576. Он применяется, например, в индикаторе MT-10T11. Используем его в качестве примера.

Прежде чем начать программирование аппаратуры, подготовим необходимые данные.

В процессе настройки контроллера PCF8576 необходимо установить параметры индикатора с помощью 4 команд и очистить память изображения от случайных данных. Необходимые коды команд можно взять из справочного руководства по PCF8567.

При передаче данных в индикатор сначала передается команда установки начального адреса памяти изображения, а сразу за ней – данные изображения. Наш индикатор имеет 10 разрядов, следовательно, потребуется буфер изображения на 10 байт плюс 1 байт команды.

Объединим команды инициализации и буфер изображения:

```
HEX CREATE InitCmd CE C, E0 C, F8 C, F0 C,
```

Мы создали массив, содержащий 4 байта команд инициализации. Если создать следующий массив, его первый байт будет "пристыкован" к последнему байту предыдущего массива, чем мы воспользуемся при инициализации индикатора.

```
CREATE DataLCD 0 C, 0 C, 0 C, 0 C, 0 C, 0 C, 0 C, 0 C, 0 C, 0 C,
```

Теперь у нас есть буфер данных, заполненный нулями.

Изображение в каждом разряде индикатора формируется из семи сегментов цифрового знака и десятичной точки (на рисунке ниже). Каждому элементу изображения соответствует один бит в байте данных памяти изображения. При установке этого бита в единицу сегмент включается.

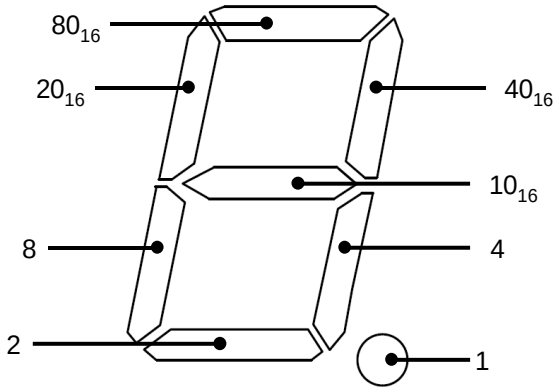


Рисунок 12.2 Маски битов, управляющих сегментами символа ЖКИ

На рисунке показаны маски битов, включающие соответствующий сегмент.

Если вы хотите получить изображение цифры 3, надо включить сегменты с масками 2, 4, 10, 40, 80 (значения шестнадцатеричные). Чтобы получить значение байта, надо сложить маски, получаем D6.

Мы уже приготовили все необходимые значения для изображения цифр от 0 до 9. Создадим массив данных:

```
ALIGN HERE EE C, 44 C, DA C, D6 C, 74 C, B6 C, BE C, C4 C, FE C,
F6 C, 0 C,
```

В конце массива мы добавили нулевой элемент, заменяющий пробел.

Теперь преобразуем массив в таблицу констант:

```
DECIMAL 11 CTABLE CodeLCD
```

Освобождаем память в секции данных:

```
-11 ALLOT
```

Теперь можно заняться программированием аппаратуры.

Сначала надо подключить электрические цепи блока I2C к контактам модуля AFM. Шина I2C использует выводы PA9 (сигнал SCL) и PA10 (сигнал SDA).

Для нормальной работы сигналы шины должны быть подтянуты резисторами к напряжению питания, а выходы I2C настроены как "открытый сток".

Мы не будем подключать к шине I2C ничего, кроме ЖКИ индикатора, поэтому можно использовать встроенные подтягивающие резисторы для PA9 и PA10.

Чтобы не отвлекаться от основной цели (программирование I2C), мы уже определили, какие значения надо установить в качестве параметров PIO.

Просто введите следующие строки:

```
HEX
0100.0440 5 NUL PIO-CFG!
NUL 6600 1 NUL PIO-CFG!
4 RADIX!
0110 NUL 3 NUL PIO-CFG!
02100220 NUL NUL NUL PIO-CFG!
DECIMAL
```

Далее инициализируем I2C как ведущее устройство:

```
NUL NUL NUL NUL I2C-INIT
```

Все готово к работе. Необходимо передать в контроллер ЖКИ команды настроек и заполнить память изображения нулями. Мы можем сделать это одной операцией, поскольку предусмотрительно соединили два массива данных.

Контроллер индикатора имеет адрес 38_{16} на шине I2C. Передавать будем 4 байта команд, 1 байт начального адреса памяти изображения и 10 байт данных изображения (нули). Итого 15 байт (шестнадцатеричное 0F).

Передаем начиная с адреса массива команд:

```
HEX
InitCmd 0F 38 NUL I2C-SEND
```

С этого момента индикатор готов к приему данных изображения. Мы будем обновлять все разряды одновременно. Для удобства создадим новое слово:

```
: UpdateLCD  InitCmd 0F 38 NUL I2C-SEND ;
```

Для отображения на индикаторе мы должны получить байт данных изображения, используя таблицу кодировки:

```
5 CodeLCD
```

возвращает в стеке данные для отображения символа "5".

Если мы хотим увидеть его в третьей позиции слева, нам надо записать его в байт буфера со смещением 3 (в нулевом байте буфера лежит код команды):

```
DataLCD 3 + C!
```

Теперь выполняем обновление ЖКИ:

```
UpdateLCD
```

Если все сделано правильно, на экране появится пятерка.

Если вы хотите добавить десятичную точку к изображению цифры, добавьте 1 к данным изображения перед записью в буфер:

```
7 CodeLCD 1+ DataLCD 2 + C! UpdateLCD
```

Теперь на индикаторе получится "7.5".

Вы можете самостоятельно доработать таблицу кодировки для изображения шестнадцатеричных цифр, символов "минус", "градус" и других.

12.5. Последовательный асинхронный интерфейс (UART)

Последовательный асинхронный интерфейс появился в компьютерах одним из первых. В "больших" ЭВМ первых поколений он служил средством связи между машиной и терминалом пользователя (вы также используете этот интерфейс для связи с Форт-машиной).

Благодаря своей простоте и гибкости последовательные каналы связи (COM-порты) долгое время были обязательным оборудованием персональных компьютеров.

Сейчас асинхронная последовательная связь широко применяется в средствах автоматизации, а соответствующие устройства ввода-вывода можно обнаружить практически в каждом микроконтроллере.

Встроенный в микроконтроллер блок, обеспечивающий поддержку асинхронного последовательного канала связи, называется Universal Asynchronous Receiver Transmitter, то есть "последовательный асинхронный приемник-передатчик", или UART. Физический интерфейс формируется дополнительными внешними схемами, задающими напряжения логических сигналов нуля и единицы, но с точки зрения программирования - UART и есть "последовательный интерфейс".

Асинхронный последовательный интерфейс не требует синхронизации данных отдельным сигналом. В минимальном варианте используется две электрических цепи:

TXD (Transmitted Data) – выход передаваемых данных;

RXD (Received Data) – вход принимаемых данных.

При передаче больших объемов информации с высокой скоростью возникает необходимость управления потоком данных. С этой целью могут быть задействованы еще две цепи:

RTS (Request To Send) – выход приемника, разрешающий передачу данных;

CTS (Clear To Send) – вход передатчика, разрешающий передачу.

Выход RTS также используется для управления преобразователями физического уровня интерфейсов RS-485 (и других).

Будем считать, что вы уже знакомы с асинхронной последовательной передачей данных и программированием UART на базовом уровне (например, COM-портов в ПК). Если это не так, рекомендуем обратиться к дополнительной литературе.

12.5.1. Инициализация UART. Слово UART-INIT

Слово UART-INIT (baud mode dev --) предназначено для установки в начальное состояние блока последовательного асинхронного интерфейса и его драйвера.

На вершине стека передается системный номер UART (параметр dev).

Блокам UART присваиваются условные номера устройств, которые могут не совпадать с обозначениями изготовителя микроконтроллера.

Узнать системный номер устройства и соответствующее ему обозначение в документации изготовителя микроконтроллера можно в техническом описании вашего модуля AFMnano.

Слово UART-INIT выполняет включение указанного блока асинхронного последовательного интерфейса и настройку его драйвера.

При обращении к отсутствующему в системе устройству запрос игнорируется. Системные номера начинаются с нуля. В Форт-машине AFMnano всегда есть хотя бы один асинхронный интерфейс, следовательно, устройство с номером 0 всегда имеется.

Следующий параметр, передаваемый в стеке, определяет режим работы UART (параметр mode).

Код режима работы задается отдельными битами, которые определяют такие характеристики, как размер передаваемых данных, число стоп-битов, использование бита контроля четности и другие.

Допустимых комбинаций битов, как и режимов работы UART, может быть много. Подробное описание возможностей UART используемого модуля AFMnano вы найдете в техническом руководстве.

Стандартные драйверы устройств ввода-вывода в AFS по возможности используют код режима ноль для установки часто используемой конфигурации.

Коды широко используемых стандартных режимов для модулей AFMnano перечислены в таблице.

Таблица 12.8 Коды стандартных режимов UART

Код десятичный (HEX)	Режим	Размер данных	Стоп-биты	Контроль ошибок
00	8-N-1	8 бит	1 бит	Отключен
02	7-N-1	7 бит	1 бит	Отключен
08	8-O-1	8 бит	1 бит	Нечетность
16 (10)	8-E-1	8 бит	1 бит	Четность
192 (C0)	8-N-1, однопроводный полудуплекс (AFM-Terminal)	8 бит	1 бит	Отключен

Обычно связь осуществляется по раздельным электрическим цепям для передаваемых и принимаемых данных (TXD и RXD соответственно). Такой канал связи называется дуплексным и в нем одновременно может происходить и передача, и прием.

Если канал связи полудуплексный, то есть в любой момент времени осуществляется либо передача, либо прием, достаточно одного вывода микросхемы для приема и передачи данных. Такой режим работы поддерживают многие современные UART в микроконтроллерах. Переключение режима вход-выход на выводе TXD осуществляет сам UART. Физический интерфейс AFM-Link использует именно такой тип канала – однопроводной полудуплексный.

Если в полудуплексном канале приемник остается включенным во время передачи, он "прослушивает" линию и получает копию передаваемых данных. Эта возможность используется для контроля линии, к которой могут быть подключены сразу несколько передатчиков. Если передачу ведут два или более UART одновременно, их приемники получат искаженные данные.

Если вы хотите использовать однопроводной полудуплексный канал, установите бит 7 кода в единицу (то есть добавьте к коду режима шестнадцатеричное 80).

Если вам не нужно, чтобы приемник оставался все время включенным, установите бит 6 кода в единицу (то есть добавьте к коду режима шестнадцатеричное 40).

Если вы хотите установить режим для UART, подключенного к AFM-Link и терминалу, используйте режим 8-N-1 с однопроводным полудуплексным каналом и отключением приемника во время передачи. Код режима должен быть 80_{16} + 40_{16} = $C0_{16}$ (или 192 десятичное).

Параметр baud устанавливает скорость обмена последовательными данными в сотнях бит в секунду (bps). Для установки скорости 4800 bps укажите значение 48, для скорости 115200 bps укажите значение 1152 и т.д.

По умолчанию UART настраивается для обмена данными со скоростью 9600 bps (если вы укажете слишком большую или слишком маленькую скорость, например, ноль).

Минимальная скорость обмена, которую можно установить, равна 300 bps, максимальная – 230400 bps.

Для примера инициализируем UART с системным номером 0, для обмена символами размером 8 бит, с одним стоп-битом, без контроля четности, со скоростью 38400 bps:

```
384 NUL NUL UART-INIT
```

Многим приложениям этого вполне достаточно, осталось только разрешить работу UART. Дополнительные возможности управления предоставляет следующее слово.

12.5.2. Управление UART. Слово UART-CTRL

Слово UART-CTRL (param n dev --) предназначено для управления блоком UART и его драйвером.

На вершине стека передается номер устройства, под ним – номер запроса управления, ниже – параметр запроса.

Допустимые номера запросов приведены в таблице.

Таблица 12.9 Запросы управления UART

Запрос	Название	Описание выполняемого действия
0	UART_ONOFF	Разрешение и запрещение работы, выключение UART
1	UART_FLOW	Режим управления потоком данных и физическим интерфейсом

12.5.2.1 Запрос UART_ONOFF

Изменение режимов работы аппаратуры и драйвера UART возможно только при пассивном состоянии устройства.

Управление разрешением и запрещением работы, а также полным выключением блока UART, осуществляется запросом UART_ONOFF.

Стековая диаграмма запроса: UART-CTRL (param 0 dev --) .

Параметр запроса определяет, в какое состояние переводится устройство и его драйвер.

Значение 0 запрещает работу блока UART и переводит его в пассивное состояние. В этом состоянии возможно изменение параметров и режимов работы UART (в том числе повторная инициализация).

Значение 1 разрешает работу UART. В этом состоянии осуществляется обмен данными по каналу связи. Изменения режимов запрещены.

Для начала работы с UART (с системным номером ноль) введите:

```
1 NUL NUL UART-CTRL
```

Если ранее вы установили режим работы и скорость обмена, UART готов к приему и передаче данных.

12.5.2.2 Запрос UART_FLOW

Запрос UART_FLOW определяет способ управления потоком ввода-вывода последовательных данных и физическим интерфейсом канала связи.

Стековая диаграмма запроса: UART-CTRL (param 1 dev --) .

При приеме данных может возникнуть ситуация, когда данные из канала связи поступают быстрее, чем их успевает обрабатывать принимающая программа. При полном заполнении буфера приемника могут быть потеряны вновь пришедшие байты. Чтобы этого не произошло, необходимо сигнализировать передающей стороне о необходимости остановки потока данных. После освобождения буфера приемника должен быть послан сигнал возобновления передачи.

Такая сигнализация осуществляется на физическом уровне с помощью дополнительных цепей последовательного интерфейса – RTS (Request To Send) и CTS (Clear To Send).

Активный уровень выхода RTS устанавливается, когда приемник готов к поступлению данных. Соответственно, вход CTS используется передающей стороной для останова и возобновления передачи данных.

Использование этих дополнительных цепей чаще всего происходит автоматически, большинство UART и драйверы поддерживают такую возможность. Необходимо указать соответствующий способ управления потоком при инициализации.

Отдельный выход UART также может быть использован для переключения схем физического интерфейса полудуплексного канала связи из режима приема в режим передачи. Этот способ управления часто используется в работе интерфейса RS-485. В этом случае вывод называется DE (обычно это тот же вывод, что и RTS).

Для такого управления, в принципе, может быть использован любой выход PIO. Переключение выхода может осуществлять прикладная программа или драйвер UART. Например, для управления схемой сопряжения AFM-Link с терминалом используется выход PA13. Драйвер UART управляет им автоматически.

Для выбора способа управления укажите в качестве параметра запроса UART_FLOW одно из значений в таблице.

Таблица 12.10 Управления потоком данных и физическим интерфейсом UART

Код режима	Управление	Описание
0	Нет	Управление потоком и интерфейсом не осуществляется
1	DE	Управление интерфейсом типа RS-485 с помощью выхода DE/RTS
2	CTS/RTS	Управление интерфейсом типа RS-232 по цепям CTS и RTS
3	PA13	Управление интерфейсом с помощью выхода PA13 (AFM-Link/Terminal)

В таблице приведены основные режимы, поддерживаемые драйверами UART. По умолчанию управление потоком и интерфейсом не осуществляется (режим с кодом 0).

В сложных или заказных системах могут быть добавлены специальные режимы, описание которых приводится в документации Форт-машины.

12.5.3. Передача данных в UART. Слово UART-PUTC

Слово UART-PUTC (char dev --) передает символ драйверу UART, номер которого лежит на вершине стека. Драйвер записывает символ в буфер передачи или непосредственно в регистр данных UART.

Если буфер полон и UART не может немедленно отправить символ, слово ожидает освобождения буфера.

12.5.4. Прием данных из UART. Слово UART-GETC

Прием данных также может осуществляться по одному символу. Поступающие из канала связи символы драйвер UART помещает в буфер, из которого они извлекаются с помощью слова UART-GETC (dev -- char). В качестве параметра слову передается номер UART, а возвращается принятый символ.

Если в момент вызова в буфере приемника UART ничего нет, слово ожидает поступления хотя бы одного символа.

12.5.5. Прямой доступ к регистрам. Слова UART-REG! и UART-REG@

Драйвер UART реализует типовые функции приема и передачи данных. Для более сложного программирования UART может потребоваться прямой доступ к его регистрам для записи и чтения.

Обратите внимание: регистры блока UART имеют размер 32 бита. Для представления их содержимого требуется пара ячеек.

Слово UART-REG! (xx reg dev --) записывает двойное число в указанный регистр указанного UART. На вершине стека передается номер устройства, далее – номер регистра, а под ним – записываемое значение в паре ячеек.

Слово UART-REG@ (reg dev -- xx) считывает значение указанного регистра указанного UART. На вершине стека передается номер устройства, далее – номер регистра. Возвращается прочитанное значение регистра в паре ячеек.

Номер регистра и его назначение указаны в технической документации на модуль AFMnano. При обращении к регистру с несуществующим номером запись не производится, а при чтении возвращается ноль.

12.5.6. Разделение ресурсов между UART и AFM-Link

Возможно использование UART, поверх которого работает системная текстовая консоль, для приема и передачи данных программой пользователя.

Если программа обменивается через UART только текстовой информацией, используйте слова консольного ввода-вывода, например, **KEY**, **ACCEPT**, **EMIT**, **TYPE**. В этом случае, как правило, необходимости в использовании драйвера UART не возникает.

Возможно, вам потребуется использовать другие выводы микросхемы Форт-машины, установить другие режимы работы канала связи и скорость обмена. Выполните соответствующие настройки UART, на работе консоли это никак не отразится.

Разумеется, на другой стороне канала связи вы будете получать не только данные из программы, но и сообщения об ошибках.

Если вы не хотите использовать консольный ввод-вывод, отключите системную текстовую консоль с помощью слова **CON-CTRL**.

Вы также можете изменить скорость передачи данных в канале связи AFM-Link, выполнив инициализацию используемого UART. Разумеется, вам придется изменить соответствующие настройки терминала, иначе вы не сможете продолжить работу с текстовой консолью.

В простых моделях AFM с ограниченным размером памяти устанавливается версия AFS-L0 с сокращенными функциями ввода-вывода. Если в такой системе имеется только один UART, он используется исключительно для текстовой консоли системы. Некоторые слова управления драйвером UART могут отсутствовать в словаре такой системы.

12.6. Таймеры-счетчики (TIM)

Таймеры-счетчики используются для точного измерения временных интервалов в программах, формирования импульсов заданной длительности, подсчета числа входящих импульсов, измерения длительности входных импульсов.

Это самый разнообразный класс устройств. Если ранее рассмотренные интерфейсы имеют примерно одинаковую реализацию в микроконтроллерах различных изготовителей, то таймеры-счетчики каждый делает по-своему.

Даже самая простая модель AFMnano-M0 имеет не менее 5 таймеров-счетчиков различного устройства и назначения. Число регистров, необходимых для настройки, управления и работы одного таймера-счетчика доходит до 20.

Упрощенная структура типового таймера-счетчика выглядит следующим образом.

Базовое времязадающее устройство (таймер) делит входную частоту синхронизации на фиксированный коэффициент деления. Пониженная частота используется для тактирования счетчика импульсов.

При достижении счетчиком заданного значения выдается сигнал, который может использоваться различными способами внутри контроллера или выдаваться на внешний вывод.

Дополнительно могут иметься несколько независимых счетных каналов, которые используют значение счетчика таймера для измерения длительности входных импульсов либо для генерации импульсных сигналов различной длительности.

В простых таймерах счетные каналы могут отсутствовать.

В описании модулей AFMnano обозначение выводов счетных каналов таймеров состоит из буквы T с номером блока в микросхеме и буквы C с номером счетного канала. В конце может присутствовать буква N, что обозначает дополнительный вывод того же канала.

Таким образом, T1C1 – вывод канала 1 таймера-счетчика TIM1, а T1C1 – его дополнительный вывод.

Как именно будут использоваться выводы счетных каналов, зависит от настроек конфигурации и выполняемых функций таймера-счетчика. Это могут быть входы, выходы и комплиментарные пары выходов.

Разработать простой и компактный драйвер, обеспечивающий контроль за всеми допустимыми режимами работы таймера-счетчика невозможно, поэтому основной способ взаимодействия с этими устройствами ввода-вывода – прямой доступ к регистрам.

Стандартный интерфейс драйвера также присутствует. Как и другим устройствам ввода-вывода, таймеру-счетчику необходима начальная инициализация и другие настройки, общие для всех устройств класса.

12.6.1. Инициализация таймера-счетчика. Слово TIM-INIT

Слово TIM-INIT (ARR PSC mode dev --) предназначено для установки в начальное состояние таймера-счетчика и его драйвера.

На вершине стека передается системный номер таймера-счетчика (параметр dev).

Таймерам-счетчикам присваиваются условные номера устройств, которые могут не совпадать с обозначениями изготовителя микроконтроллера. Некоторые таймеры-счетчики есть в одних моделях, но отсутствуют в других.

Узнать системный номер устройства и соответствующее ему обозначение в документации изготовителя микроконтроллера можно в техническом описании вашего модуля AFMnano.

При обращении к отсутствующему в системе устройству запрос игнорируется.

Слово TIM-INIT выполняет включение блока указанного таймера-счетчика и подключение его входа синхронизации.

Следующий параметр, передаваемый в стеке, определяет режим работы таймера-счетчика и его драйвера (параметр mode).

Параметр PSC (prescaler) содержит значение счетчика делителя входной частоты синхронизации. Далее мы рассмотрим использование этого параметра на практическом примере.

В следующем параметре задается начальное значение счетчика импульсов. Это значение сохраняется в специальном регистре (Auto Reaload Register – ARR), из которого автоматически загружается в регистр счетчика импульсов в начале каждого цикла счета.

12.6.2. Установка режима счетного канала. Слово TIM-CHAN-SET

Если таймер-счетчик имеет счетные каналы, их режимы работы устанавливаются словом TIM-CHAN-SET (mode chan dev --).

На вершине стека передается системный номер таймера-счетчика (параметр dev). Второе значение в стеке – номер программируемого счетного канала (параметр chan). Первому каналу соответствует значение ноль.

Третье значение в стеке – код режима работы счетного канала (параметр mode).

Полное описание всех допустимых режимов и соответствующие им коды приводятся в документации на Форт-машину.

Вот некоторые из них, используемые в примерах ниже и полезные для самостоятельных опытов.

Код режима 0 используется для отключения неиспользуемого счетного канала.

При установке режима 40_{16} выход счетного канала принудительно переводится в низкий логический уровень.

При установке режима 50_{16} выход счетного канала принудительно переводится в высокий логический уровень.

В режиме с кодом 30_{16} выход счетного канала изменяет свое состояние на противоположное при каждом совпадении счетчика импульсов с заданным значением (см. стр. 168).

Режимы широтно-импульсной модуляции (ШИМ) имеют коды 60_{16} и 70_{16} (см. стр. 163).

12.6.3. Управление таймером-счетчиком. Слово TIM-CTRL

Слово TIM-CTRL (param n dev --) предназначено для управления блоком таймера-счетчика и его драйвером.

На вершине стека передается номер устройства, под ним – номер запроса управления, ниже – параметр запроса.

В простых системах используется только один управляющий запрос – включение и выключение, но в более сложных системах могут присутствовать и другие.

Таблица 12.11 Запросы управления таймера-счетчика

Номер	Запрос	Описание выполняемого действия
0	TIM_ONOFF	Включение и выключение таймера-счетчика

12.6.3.1 Запрос TIM_ONOFF

Изменение ключевых параметров таймера-счетчика и его счетных каналов, задаваемых словами TIM-INIT и TIM-CHAN-SET, возможно только при пассивном состоянии устройства. Управление разрешением, запрещением и полным выключением таймера-счетчика осуществляется запросом TIM_ONOFF.

Стековая диаграмма запроса: TIM-CTRL (param 0 dev --) .

Параметр запроса определяет, в какое состояние переводится устройство и драйвер.

Значение 0 останавливает работу таймера-счетчика и переводит его в пассивное состояние. В этом состоянии возможно изменение параметров и режимов работы устройства.

Значение 1 переводит таймер-счетчик в активное рабочее состояние. В этом состоянии осуществляется работа времязадающего устройства и счетных каналов. Изменения режимов запрещены.

Значение 2 используется для прекращения работы таймера-счетчика и полного отключения соответствующего блока в микросхеме. Такая возможность необходима для повышения энергоэффективности. После отключения устройства возможна только его инициализация.

Дальнейшее управление устройством осуществляется через его регистры. Мы не можем рассказать обо всех тонкостях программирования таймеров-счетчиков, это слишком обширная тема для вводного курса. Для детального изучения этого вопроса обратитесь к специальному Руководству по применению.

12.6.4. Прямой доступ к регистрам. Слова TIM-REG! и TIM-REG@

Для дальнейшей работы с таймером-счетчиком потребуется прямой доступ к его регистрам для записи и чтения битов управления и состояния.

Слово TIM-REG! (x reg dev --) записывает значение в указанный регистр указанного таймера-счетчика. На вершине стека передается номер устройства, далее – номер регистра, а под ним – записываемое значение.

Слово TIM-REG@ (reg dev -- x) считывает значение указанного регистра заданного таймера-счетчика. На вершине стека передается номер устройства, далее – номер регистра. Возвращается прочитанное значение регистра.

Номер регистра и его назначение указаны в технической документации на модуль AFMnano. При обращении к регистру с несуществующим номером запись не производится, а при чтении возвращается ноль.

12.6.5. Пример программирования таймера-счетчика

Запрограммируем устройство TIM14 (системный номер 5) для вывода повторяющихся импульсов на контакт PB1 нашего модуля AFMnano.

Вы можете подключить светодиод с рабочим током 5 mA между выводом PB1 и напряжением питания +3.3V через резистор сопротивлением не менее 330 Ом.

Учтите, что вывод PB1 не предназначен для работы с большими напряжениями, подключать светодиод к питанию 5V нельзя!

Разумеется, надо переключить вывод PB1 в режим функции встроенного устройства ввода-вывода и отключить подтягивающие резисторы. Как мы помним, это делается в настройках конфигурации PIO (см. 12.1.1. Настройка конфигурации PIO. Слова PIO-CFG! и PIO-CFG@).

Нам нужно задать номер функции 0 для PB1. Устанавливаем параметр FUNCTION-LOW (стр. 137):

```
NUL NUL 4 1 PIO-CFG!
```

Отключаем подтягивающие резисторы в PB (стр. 136):

```
NUL NUL 3 1 PIO-CFG!
```

Для вывода PB1 надо задать режим 2 (стр. 133). Значение параметра MODE для порта PB будет равно 1000 (двоичное) или 8 десятичное:

```
0.8 NUL 1 PIO-CFG!
```

Приступаем к программированию таймера.

12.6.5.1 Настройка режима работы таймера

Для начальной инициализации таймера-счетчика и его драйвера воспользуемся словом TIM-INIT (ARR PSC mode dev --). Ему требуется передать номер устройства, код режима и два параметра настройки. Номер режима пока установим равным нулю. Это типовой режим, позволяющий реализовать большинство интересующих нас функций.

Мы хотим получить импульсы, включающие светодиод с частотой 1 Гц, но с управляемой длительностью периодов свечения и гашения. Фактически мы программируем режим широтно-импульсной модуляции (ШИМ или PWM – Pulse Width Modulation).

Входная частота синхронизации (24 МГц, если вы ее не изменили) слишком высока для нашей задачи. Чтобы привести ее к необходимому значению на входе таймера имеется делитель (prescaler).

Значение в регистре делителя соответствует числу импульсов входного сигнала синхронизации, приходящихся на один импульс выходного сигнала делителя:

Выходная частота делителя = Входная частота синхронизации / (PSC+1),

где PSC = значение в регистре делителя.

Если регистр сброшен в 0, деление не происходит (тактовый сигнал проходит на выход без изменений).

Мы собираемся регулировать длительность свечения светодиода, поэтому выходная частота делителя должна быть больше, чем требуемый нам 1 Гц (позже мы объясним причину).

Выберем выходную частоту 1000 Гц и посчитаем требуемый коэффициент деления:

PSC = (Входная частота / Выходная частота) - 1 = (24 000 000 Гц / 1000 Гц) – 1 = 23999

Это число мы и должны передать в качестве параметра PSC для слова **TIM-INIT**.

Импульсы с выхода делителя поступают в счетчик времязадающего устройства. Значение счетчика изменяется при поступлении каждого импульса.

Используемый нами таймер-счетчик TIM14 имеет только один режим работы времязадающего устройства: счетчик обнуляется в начале цикла и увеличивается на 1 каждым входным импульсом до достижения базового значения, после чего выдается выходной импульс и счет начинается заново (с нуля).

В других таймерах-счетчиках имеется еще несколько режимов работы, но нам они сейчас не интересны. Нам вполне подойдет инкрементный (то есть от нуля вверх) циклический режим счета.

В качестве базового значения зададим число 1000. Мы знаем, что импульсы с выхода делителя приходят на счетчик с частотой 1000 Гц, значит, полный цикл от 0 до 1000 счетчик пройдет ровно за 1000 импульсов, т.е. за одну секунду, а это и есть требуемая нам частота 1 Гц.

Это не объясняет, почему все-таки выбрано значение 1000 импульсов в секунду, но разгадка уже близко. Пока же передаем базовое значение счетчика в предназначенном для этого параметре ARR:

```
1000 23999 NUL 5 TIM-INIT
```

Времязадающее устройство настроено, теперь необходимо настроить счетный канал.

12.6.5.2 Настройка канала ШИМ

Широтно-импульсная модуляция осуществляется следующим образом. В счетном канале таймера-счетчика имеется специальный регистр сравнения (номер 13_{10}). Его содержимое сравнивается с счетчиком импульсов времязадающего устройства после каждого импульса синхронизации (выход делителя). Возможны два варианта генерации выходного сигнала:

- выход активен, пока счетчик импульсов меньше регистра сравнения (ШИМ1);
- выход активен, пока счетчик импульсов больше регистра сравнения (ШИМ2).

Мы выберем режим ШИМ1. Задавая значение в регистре сравнения от 1 до 999 мы можем регулировать время активности выходного сигнала внутри цикла длительностью 1 секунда с точностью до 1 мс.

Другими словами, если весь период генерации выходного сигнала составляет 1000 импульсов синхронизации (1000 мс), а мы записали в регистр сравнения число 700, то первые 700 мс, пока счетчик набирает значение от 0 до 700, выход активен (светодиод горит), а последующие 300 мс, когда счетчик больше 700, выход неактивен (светодиод погашен).

Соответствующие режимы работы канала устанавливаются словом **TIM-CHAN-SET**.

Нам нужно установить режим сравнения ШИМ1 с генерацией импульса на выходе канала.

Требуемый режим имеет шестнадцатеричный код 60 (для ШИМ2 код 70_{16}):

```
HEX 60 NUL 5 TIM-CHAN-SET
```

Номер канала задан нулем. В регистр сравнения надо не забыть записать длительность активного уровня импульса. Для начала пусть будет 500 мс:

```
DECIMAL 500 13 5 TIM-REG!
```

Нам осталось подключить выход канала к выводу модуля AFMnano. Для этого надо установить бит разрешения выходного сигнала и выбрать полярность активного уровня выходного сигнала в регистре номер 8.

Бит разрешения в регистре 8 самый младший (бит 0), его надо установить в 1. Мы подключаем светодиод между выходом и питанием, поэтому активный уровень сигнала должен быть логическим нулем. За полярность выхода отвечает бит 1, его также надо установить в 1.

Записываем нужное значение в регистр 8:

```
3 8 5 TIM-REG!
```

Все готово к работе, осталось разрешить работу таймера:

```
1 NUL 5 TIM-CTRL
```

Светодиод должен начать загораться и гаснуть с равными интервалами один раз в секунду.

12.6.5.3 Управление ШИМ

Если все сделано без ошибок, после разрешения работы таймера подключенный к выходу PB1 светодиод начнет вспыхивать. Время свечения должно быть равно времени гашения, так как мы записали в регистр сравнения 500, а полный цикл равен 1000 импульсов счета.

Мы можем изменить соотношение времени свечения и гашения, установив другое значение в регистре сравнения (номер 13):

```
250 13 5 TIM-REG!
```

Теперь светодиод светится меньшее время и значительно дольше остается выключенным.

Изменим значение в регистре на большее:

```
750 13 5 TIM-REG!
```

Картина изменилась – светодиод светится дольше, выключаясь на короткое время.

13. Примеры программ

Вы можете скопировать приведенные здесь примеры программ в отдельные текстовые файлы для последующей загрузки в Форт-машину или копировать построчно в ходе экспериментов. Все тексты программ проверены на практике и гарантированно работают при соблюдении указанных условий.

13.1. Генерация сигнала ШИМ для сервопривода

Эта небольшая программа предназначена для демонстрации способа управления стандартным микромощным сервоприводом типа SG-90 или аналогичным. Подобные сервоприводы, также известные как рулевые машинки, широко применяются в моделях с дистанционным управлением и любительской автоматике.

Подключение сервопривода довольно простое – необходимо подать напряжение питания 5V и сигнал управления. На платах AFM Evo-I и AFMcustom-M1.10 имеются специальные разъемы для подключения стандартных рулевых машинок.

Угол поворота вала сервопривода управляется сигналом с широтно-импульсной модуляцией (ШИМ, PWM – Pulse Width Modulation).

Мы используем все каналы таймера TIM1, что позволит управлять одновременно 4 сервоприводами.

Сигналы ШИМ генерируется встроенным таймером TIM1 (системный номер 0) на выводах T1C1, T1C2, T1C3, T1C4. Вы можете использовать типовую конфигурацию PIO или самостоятельно настроить подходящие выводы модуля AFM.

Программа работает при частоте процессора 24 МГц (по умолчанию).

Слово **SERVO-INIT** выполняет все необходимые настройки таймера-счетчика. Вы можете проследить по тексту программы (ниже) выполнение следующих операций.

Для генерации сигнала ШИМ частота синхронизации таймера устанавливается равной 1 МГц, что позволяет управлять длительностью импульсов с шагом 1 мкс.

Коэффициент деления 23 устанавливается при инициализации таймера-счетчика..

Период ШИМ для управления сервоприводом должен быть равен 20 мс. Для получения нужного времени с шагом 1 мкс базовое значение счетчика устанавливается равным 20000.

Для установки режима ШИМ в каждом счетном канале задается режим с кодом 60_{16}.

Среднему положению вала сервопривода соответствует длительность высокого уровня импульса ШИМ равная 1.5 мс.

Слово SERVO1 (u --) записывает новое значение длительности высокого уровня импульса ШИМ в регистр сравнения счетного канала №1 – при инициализации устанавливается начальное значение 1500, что соответствует 1.5 мс с шагом 1 мкс. Для управления остальными каналами предназначены слова **SERVO2**, **SERVO3** и **SERVO4**.

Для подключения выходов каналов таймера к выводам модуля в регистре 8 устанавливаются в единицу биты разрешения. Полярность выходного сигнала не меняется.

Таймер TIM1 имеет расширенные функции управления, поэтому для подключения выходов счетных каналов потребуется дополнительно записать значение $8C00_{16}$ в регистр 17_{10} (11_{16}).

Подключите сервоприводы к соответствующим выводам модуля AFM и включите питание.

Загрузите текст программы в Форт-машину и убедитесь, что компиляция прошла без ошибок.

Выполните слово **SERVO-INIT**. Вал каждого сервопривода должен установиться в среднее положение, если до этого он был смещен.

Используя слово **SERVO1** изменяйте положение вала. Слово **SERVO1** получает в качестве параметра значение длительности высокого уровня импульса ШИМ в микросекундах. Для других каналов используйте соответствующие слова SERVOx.

Среднему положению вала сервопривода соответствует длительность 1500 мкс, вращению по часовой стрелке – длительности менее 1500 мкс, вращению против часовой стрелки – длительности более 1500 мкс.

Примеры управления углом поворота вала сервопривода:

```
2000 SERVO1 \\ Против часовой стрелки

1000 SERVO1 \\ По часовой стрелке

1500 SERVO1 \\ Среднее положение
```

Используя различные значения, передаваемые слову **SERVO1**, устанавливайте вал сервопривода в нужные положения.

Для большинства сервоприводов полный допустимый диапазон длительности высокого уровня импульса ШИМ составляет от 500 мкс до 2500 мкс.

```
\\ Генерация ШИМ для управления сервоприводом
DECIMAL

: SERVO1 \ ( u -- ) Изменение угла поворота вала сервопривода \
  13 NUL TIM-REG! ;
: SERVO2 14 NUL TIM-REG! ;
: SERVO3 15 NUL TIM-REG! ;
: SERVO4 16 NUL TIM-REG! ;

\\ Настраиваем TIM1 для периода генерации 2 ms с шагом 1 us

: SERVO-INIT
  20000 23 NUL NUL TIM-INIT
\\ Начальная длительность активного уровня ШИМ = 1500 us
  1500 DUP 2DUP SERVO1 SERVO2 SERVO3 SERVO4
\\ Устанавливаем режим ШИМ-1 в каналах таймера
  HEX
  60 NUL NUL TIM-CHAN-SET
  60 1 NUL TIM-CHAN-SET
  60 2 NUL TIM-CHAN-SET
  60 3 NUL TIM-CHAN-SET
\\ Подключаем выходы таймера к выводам модуля
  1111 8 NUL TIM-REG!
  8C00 11 NUL TIM-REG!
\\ Разрешаем таймер
  1 NUL NUL TIM-CTRL
;
DECIMAL
```

13.2. Генерация звуковых сигналов

Эта программа позволяет генерировать простейшие звуковые сигналы с помощью пьезоизлучателя (бипера). Сигнал звуковой частоты генерируется встроенным таймером TIM14 (системный номер 5) на выводе T14C1. Вы можете использовать типовую конфигурацию PIO или самостоятельно настроить подходящие выводы модуля AFM.

На плате AFMcustom-M1.10 звуковой излучатель установлен и подключается автоматически в типовой конфигурации 1 или 2. При самостоятельной сборке схемы соедините один контакт излучателя с цепью питания 3.3V, а второй – с соответствующим выводом модуля AFM.

Программа работает при частоте процессора 24 МГц (по умолчанию).

Слово **BEEP-INIT** выполняет все необходимые настройки таймера-счетчика. Вы можете проследить по тексту программы (ниже) выполнение следующих операций.

Для генерации звукового сигнала счетный канал таймера устанавливается в режим, при котором состояние выхода канала меняется на противоположное при каждом совпадении регистра сравнения и значения счетчика импульсов.

Таким образом, период следования импульсов определяется базовым значением счетчика времязадающего устройства, а скважность равна 50% (форма выходного сигнала - меандр).

Частота синхронизации таймера устанавливается равной 1 МГц, что позволяет управлять длительностью и скважностью импульсов с шагом 1 мкс.

Коэффициент деления 23 устанавливается при инициализации таймера-счетчика..

При начальной инициализации базовое значение счетчика устанавливается равным 250, что соответствует частоте выходного сигнала 4 кГц.

В счетном канале таймера устанавливается режим с кодом 30_{16}. В регистр сравнения записывается 50, что ограничивает максимальную частоту выходного сигнала 20 кГц.

Слово TONE (u --) записывает новое базовое значение счетчика в регистр автозагрузки (ARR). Таким образом обеспечивается генерация сигнала с изменяемым периодом при сохранении скважности 50%.

Для подключения выхода счетного канала таймера к выводу модуля в регистре 8 устанавливается в единицу бит разрешения. Полярность выходного сигнала инвертируется, поскольку излучатель подключен между выходом и цепью питания. Для этого также взводится бит полярности в том же регистре.

Подключите пъезоизлучатель к соответствующим выводам модуля AFM и включите питание. Настройте конфигурацию используемого вывода.

Загрузите текст программы в Форт-машину и убедитесь, что компиляция прошла без ошибок.

Выполните слово **BEEP-INIT**. Вы услышите звук с частотой 4 кГц.

Используя слово **TONE** изменяйте тональность сигнала.

```
\\ Программа выдачи звука на PA4/T14C1
DECIMAL

\\ Слово для установки периода выходного сигнала
: TONE \ ( u -- )  u = длительность периода в мкс \
  11 5 TIM-REG!
;

\\ Настраиваем TIM14 для генерации импульсов с шагом 1 мкс
: BEEP-INIT
  1 1 SYS-CFG!
  250 23 NUL 5 TIM-INIT
\\ Устанавливаем режим генерации меандра на выходе таймера
  50 13 5 TIM-REG!
  HEX 30 NUL 5 TIM-CHAN-SET
\\ Подключаем выход таймера к выводу модуля и инвертируем сигнал
  3 8 5 TIM-REG!
\\ Разрешаем таймер
  1 NUL 5 TIM-CTRL
;
DECIMAL
```

13.3. Управление двигателями постоянного тока

На плате AFMcustom-M1.10 установлена специальная схема – драйвер двигателей постоянного тока. С помощью этой схемы вы можете управлять включением, торможением и направлением вращения двух двигателей.

Входы управления драйвера подключены к следующим выводам портов:

PA5 – управление включением драйвера. При установке высокого логического уровня схема включена (каждый двигатель управляется независимо), при низком логическом уровне схема выключена и находится в режиме энергосбережения (двигатели могут вращаться свободно).

PA6 и PA7 – управление двигателем A (см. таблицу).

PB0 и PB1 – управление двигателем B (см. таблицу).

Таблица 13.1 Управление двигателями постоянного тока

Состояние PA6 (PB0)	Состояние PA7 (PB1)	Состояние двигателя A (B)
0	0	Свободное вращение двигателя
1	0	Вращение в прямом направлении
0	1	Вращение в обратном направлении
1	1	Торможение

Понятия "прямого" и "обратного" направления условные. Разъемы для подключения двигателей устроены таким образом, что при подключении двух двигателей с одинаковой полярностью соединительных кабелей и одинаковыми установками управляющих сигналов, вращаться они будут в противоположные стороны. Это весьма удобно при работе с ходовыми двигателями различных мобильных устройств, например, роботов.

Обратите внимание: при установке типовой конфигурации 1 указанные выводы PIO конфигурируются как выходы таймера-счетчика TIM3. Это позволяет управлять скоростью вращения двигателей с помощью ШИМ. Таким образом, возможно два варианта программы управления.

Рассмотрим текст простой программы без использования ШИМ (см. ниже).

Для начала работы необходимо сконфигурировать выводы PA5, PA6, PA7, PB0 и PB1 как дискретные выходы портов.

Слово **DRIVER-INIT** выполняет все необходимые настройки PIO.

Для включения и выключения микросхемы драйвера используются слова **DRV-ON** и **DRV-OFF** соответственно.

Слово **A-MOT-FREE** устанавливает режим свободного вращения для двигателя A. Это же состояние для двигателя B устанавливается словом **B-MOT-FREE**.

Слово **A-MOT-FWD** включает вращения двигателя A в прямом направлении. Для двигателя B прямое вращение включается словом **B-MOT-FWD**.

Реверс двигателей включается словами **A-MOT-REV** и **B-MOT-REV**.

Для торможения двигателей используются слова **A-MOT-STOP** и **B-MOT-STOP**.

Подключите двигатели к разъемам платы и включите питание.

Загрузите текст программы в Форт-машину и убедитесь, что компиляция прошла без ошибок.

Выполните слово **DRIVER-INIT**. Выполните слово **DRV-ON**. На этом этапе двигатели не должны включаться, если все сделано правильно.

Используя слова **A-MOT-FWD** и **B-MOT-FWD** включите двигатели. Если направление вращения не соответствует желаемому, поменяйте местами разъемы подключения к плате.

Проверьте реверс двигателей и торможение. По завершении работы выключите драйвер словом **DRV-OFF**.

```
\\ Простая программа управления двигателями постоянного тока
HEX

: DRV-OFF \ Выключение микросхемы драйвера \
20 NUL NUL PIO-BIT-SR
;
: DRV-ON \ Включение микросхемы драйвера \
NUL 20 NUL PIO-BIT-SR
;
: A-MOT-FREE \ Свободное вращение двигателя A \
C0 NUL NUL PIO-BIT-SR
;
: A-MOT-STOP \ Торможение двигателя A \
NUL C0 NUL PIO-BIT-SR
;
: A-MOT-FWD \ Прямое вращение двигателя A \
80 40 NUL PIO-BIT-SR
;
: A-MOT-REV \ Обратное вращение двигателя A \
40 80 NUL PIO-BIT-SR
;
: B-MOT-FREE \ Свободное вращение двигателя B \
3 NUL 1 PIO-BIT-SR
;
: B-MOT-STOP \ Торможение двигателя B \
NUL 3 1 PIO-BIT-SR
;
```

```
: B-MOT-FWD \ Прямое вращение двигателя B \
2 1 1 PIO-BIT-SR
;
: B-MOT-REV \ Обратное вращение двигателя B \
1 2 1 PIO-BIT-SR
;

: DRIVER-INIT \ Начальные установки PA5 PA6 PA7 PB0 PB1 \
NUL NUL PIO-CFG@
DROP 5400 NUL PIO-CFG!
DRV-OFF
NUL 5 NUL 1 PIO-CFG!
;

DECIMAL
```

Теперь рассмотрим более сложный вариант программы.

Для начала работы необходимо сконфигурировать выводы PA6, PA7, PB0 и PB1 как выходы счетных каналов таймера-счетчика TIM3, а вывод PA5 – как дискретный выход порта PIO. Все необходимые настройки PIO выполняются при выборе типовой конфигурации 1 (на плате AFMcustom-M1.10).

Для включения и выключения микросхемы драйвера используются слова **DRV-ON** и **DRV-OFF** соответственно.

Инициализация таймера выполняется словом **INIT-PWM-AB**.

Слово **A-MOT-FREE** устанавливает режим свободного вращения для двигателя A. Это же состояние для двигателя B устанавливается словом **B-MOT-FREE**.

Слово **A-MOT-FWD** включает вращения двигателя A в прямом направлении. Для двигателя B прямое вращение включается словом **B-MOT-FWD**.

Реверс двигателей включается словами **A-MOT-REV** и **B-MOT-REV**.

Для торможения двигателей используются слова **A-MOT-STOP** и **B-MOT-STOP**.

Скорость вращения двигателей устанавливается словами **PWMA** и **PWMB**.

Для управления двигателями используются комбинации режимов работы счетных каналов таймера-счетчика TIM3.

Подключите двигатели к разъемам платы и включите питание.

Загрузите текст программы в Форт-машину и убедитесь, что компиляция прошла без ошибок.

Выполните слово **INIT-PWM-AB**. На этом этапе двигатели не должны включаться, если все сделано правильно.

Используя слова **A-MOT-FWD** и **B-MOT-FWD** включите двигатели. Если направление вращения не соответствует желаемому, поменяйте местами разъемы подключения к плате.

Изменяйте скорость вращения с помощью слов **PWMA** и **PWMB**. Рекомендуемый диапазон значений параметров (скважность ШИМ) – от 70 до 100 процентов.

Проверьте реверс двигателей и торможение. По завершении работы выключите микросхему драйвера словом **DRV-OFF**.

```
\\ Программа управления двигателями постоянного тока
\\ с возможностью регулирования скорости вращения (ШИМ)
HEX
1 1 SYS-CFG!   \ Типовая конфигурация 1 \

: DRV-OFF \ Выключение микросхемы драйвера \
20 NUL NUL PIO-BIT-SR ;

: DRV-ON \ Включение микросхемы драйвера \
NUL 20 NUL PIO-BIT-SR ;

: A-MOT-FREE \ Свободное вращение двигателя A \
\\ Устанавливаем низкий уровень на выходе 1 и 2 канала таймера
  40 DUP
  NUL 4 TIM-CHAN-SET
  1 4 TIM-CHAN-SET
;
: A-MOT-STOP \ Торможение двигателя A \
\\ Устанавливаем высокий уровень на выходе 1 и 2 канала таймера
  50 DUP
  NUL 4 TIM-CHAN-SET
  1 4 TIM-CHAN-SET
;
: A-MOT-FWD \ Прямое вращение двигателя A \
\\ Устанавливаем низкий уровень на выходе 2 канала таймера
  40 1 4 TIM-CHAN-SET
\\ Устанавливаем режим ШИМ-1 в первом канале таймера
  60 NUL 4 TIM-CHAN-SET
;
```

```
: A-MOT-REV \ Обратное вращение двигателя A \
\\ Устанавливаем низкий уровень на выходе 1 канала таймера
  40 NUL 4 TIM-CHAN-SET
\\ Устанавливаем режим ШИМ-1 во втором канале таймера
  60 1 4 TIM-CHAN-SET
;

: B-MOT-FREE \ Свободное вращение двигателя B \
\\ Устанавливаем низкий уровень на выходе 3 и 4 канала таймера
  40 DUP
  2 4 TIM-CHAN-SET
  3 4 TIM-CHAN-SET
;

: B-MOT-STOP \ Торможение двигателя B \
\\ Устанавливаем высокий уровень на выходе 3 и 4 канала таймера
  50 DUP
  2 4 TIM-CHAN-SET
  3 4 TIM-CHAN-SET
;

: B-MOT-FWD \ Прямое вращение двигателя B \
\\ Устанавливаем низкий уровень на выходе 4 канала таймера
  40 3 4 TIM-CHAN-SET
\\ Устанавливаем режим ШИМ-1 в третьем канале таймера
  60 2 4 TIM-CHAN-SET
;

: B-MOT-REV \ Обратное вращение двигателя B \
\\ Устанавливаем низкий уровень на выходе 3 канала таймера
  40 2 4 TIM-CHAN-SET
\\ Устанавливаем режим ШИМ-1 в четвертом канале таймера
  60 3 4 TIM-CHAN-SET
;
```

```
DECIMAL

\\ Слово для установки длительности активного уровня ШИМ
\\ двигателя A (PA6 и PA7). Соответствует скважности в %%.
: PWMA \ ( x -- )  x = скважность в процентах (1..99) \
  DUP
  13 4 TIM-REG!
  14 4 TIM-REG!
;
\\ Слово для установки длительности активного уровня ШИМ
\\ двигателя B (PB0 и PB1). Соответствует скважности в %%.
: PWMB \ ( x -- )  x = скважность в процентах (1..99) \
  DUP
  15 4 TIM-REG!
  16 4 TIM-REG!
;

\\ Слово для инициализации
: INIT-PWM-AB
  100 239 NUL 4 TIM-INIT
\\ Двигатели в состоянии свободного вращения
  A-MOT-FREE B-MOT-FREE
\\ Устанавливаем по умолчанию 75% скважности ШИМ
  75 DUP PWMA PWMB
\\ Разрешаем работу микросхемы драйвера
  DRV-ON
\\ Подключаем выходы таймера к выводам модуля
  HEX 1111 8 4 TIM-REG!
\\ Разрешаем таймер
  1 NUL 4 TIM-CTRL
;
DECIMAL
```

Приложение А. Классификация AFS

В таблице приведена классификация вариантов реализации.

Таблица А.1 Классификация AFS

Параметры AFS	**Level-0**	**Level-1**	**Level-2**	**Level-3**
Размер ячейки, бит	**16**	**16**	**16**	**16, 32**
Прикладные процессы	**1**	**1**	**1**	**>1**[1]
Нити прикладного процесса	–	–	+	+
Обработка событий в приложении	–	+	+	–
Тиражирование приложений	+	+	+	–
Файловая система	–	–	±[2]	+
Перемещаемые исполняемые файлы	–	–	–	+
Расширение компилятора	–	–	±[3]	+
Аппаратная платформа	Микроконтроллеры[4]			Компьютеры[4]
Архитектура процессора	ARM (Cortex)	ARM (Cortex)	ARM (Cortex)	ARM, IA
Примеры применения	Умные сенсоры, простые контроллеры, учебные пособия	Универсальные контроллеры средней производительности	Высокопроизводительные контроллеры, сетевые агенты	Встраиваемые компьютеры, рабочие станции, серверы

Примечания.

1) Число одновременно выполняемых прикладных процессов определяется размером ОЗУ и настройками ядра ОС.

2) В однокристальных микроконтроллерах поддержка полноценной файловой системы на встроенной Flash практически невозможна, но при достаточном объеме Flash-ROM возможна организация хранения нескольких файлов. Микроконтроллеры с аппаратной поддержкой карт памяти могут использовать внешнюю файловую систему при наличии достаточного объема ОЗУ для работы программного обеспечения файловой системы.

3) Расширение компилятора подразумевает возможность создания пользователем собственных слов для управления компиляцией метакода. Такое расширение системного словаря требует более сложной организации интерактивной среды программирования и дополнительного объема Flash-ROM.

4) Микроконтроллеры – микросхемы со встроенными Flash-ROM и ОЗУ, обычно ограниченного объема, AFS интегрирована на кристалле. Компьютеры – электронные модули, имеющие большой объем постоянной памяти и ОЗУ. Ядро AFS должно быть интегрировано в процессор или постоянную память с шифрованием и обеспечением безопасной загрузки в интегрированную память процессора.

Уровень L0 предназначен для интеграции AFS в самые простые и дешевые микроконтроллеры. Такие микросхемы обычно выполняют одну программу и не требуют реализации сложных алгоритмов. В AFS-L0 все события, например, аппаратные прерывания, обрабатываются ядром системы. Прикладная программа получает информацию о событиях во время опроса состояния регистров ввода-вывода или драйверов устройств.

Уровень L1 отличается от L0 возможностью организации обработки событий в прикладной программе. Для реализации этой возможности требуется увеличенный объем постоянной памяти для хранения дополнительного кода ядра AFS и Форт-программы.

Микроконтроллеры с относительно большими объемами памяти могут оснащаться AFS уровня 2. От предыдущих уровней L2 отличается возможностью организации нескольких нитей выполнения кода внутри одной прикладной программы. Для каждой нити система поддерживает собственные стеки, что требует увеличения доступного объема ОЗУ. Прикладной процесс по-прежнему один, нити используют общую секцию данных.

Поскольку в AFS уровней L0-L2 применяется оптимизация метакода прикладной программы под конкретный микроконтроллер и отсутствует файловая система, выгрузить исполняемый файл программы из Форт-машины невозможно. Для тиражирования программы пользователя предусмотрены встроенные программно-аппаратные средства. Передача готовой программы от одной AFM к другой осуществляется по интерфейсу AFM-Link, обе машины должны быть одной модели и иметь одинаковые версии AFS.

Для AFS уровня 3 с поддержкой файловой системы осуществляется выгрузка исполняемого файла формата AFX. Такие исполняемые файлы могут выполняться любой AFM с установленной AFS-L3.

Разумеется, во всех случаях передачи программы от одной машины в другую должны совпадать размеры ячеек программы и AFS.

В документации на изделия используются следующие обозначения системного программного обеспечения (Рисунок А.1).

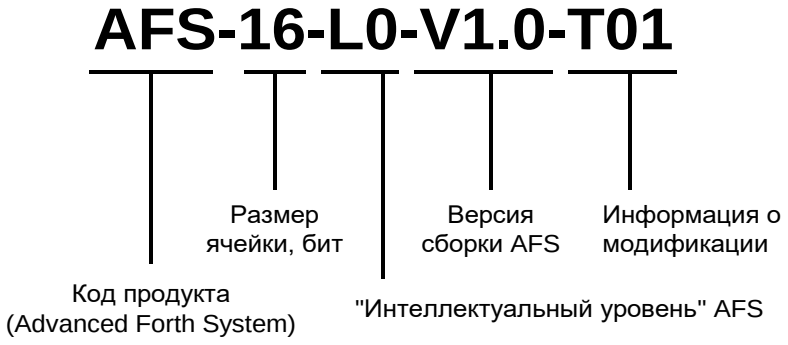


Рисунок А.1 Обозначение системного ПО семейства AFS

После кода продукта (AFS) указывается размер ячейки в битах. В настоящее время используются два размера ячеек – 16 бит или 32 бита.

Следующее поле содержит обозначение "интеллектуального уровня" системы (L0, L1, L2, L3). За ним следует поле версии сборки AFS.

Для серийных изделий код версии сборки начинается с буквы "V". Буква "R" обозначает "исследовательскую" версию системы. Такие сборки могут устанавливаться в Форт-машины для демонстрации и тестирования. После проведения испытаний и внесения необходимых изменений в код программного обеспечения ему присваивается версия с литерой "V".

После поля версии может быть добавлена информация о модификации стандартной AFS. Такое поле присутствует в обозначении AFS, установленной в специальных вариантах AFM. Обычно модификация заключается в добавлении к стандартному словарю слов для управления устройствами, размещенными на кастомизированном модуле AFM.

Обозначение установленной AFS выводится на текстовую консоль при каждом включении или перезагрузке Форт-машины. В выводимом обозначении допускается отсутствие дефисов, разделяющих поля.

Приложение B. Стандартные коды ASCII

В таблице представлены десятичные и шестнадцатеричные значения кода ASCII для алфавитно-цифровых символов и соответствующее изображение символа.

Таблица B.1 Алфавитно-цифровые символы ASCII

Код	HEX	Символ	Код	HEX	Символ	Код	HEX	Символ
32	20	(пробел)	64	40	@	96	60	` (обр. апостроф)
33	21	**!**	65	41	**A**	97	61	**a**
34	22	**"**	66	42	**B**	98	62	**b**
35	23	#	67	43	**C**	99	63	**c**
36	24	**$**	68	44	**D**	100	64	**d**
37	25	**%**	69	45	**E**	101	65	**e**
38	26	**&**	70	46	**F**	102	66	**f**
39	27	**'** (апостроф)	71	47	**G**	103	67	**g**
40	28	**(**	72	48	**H**	104	68	**h**
41	29	**)**	73	49	**I**	105	69	**i**
42	2A	*****	74	4A	**J**	106	6A	**j**
43	2B	**+**	75	4B	**K**	107	6B	**k**
44	2C	**,** (запятая)	76	4C	**L**	108	6C	**l**
45	2D	**-**	77	4D	**M**	109	6D	**m**
46	2E	**.** (точка)	78	4E	**N**	110	6E	**n**
47	2F	**/**	79	4F	**O**	111	6F	**o**
48	30	**0**	80	50	**P**	112	70	**p**
49	31	**1**	81	51	**Q**	113	71	**q**
50	32	**2**	82	52	**R**	114	72	**r**
51	33	**3**	83	53	**S**	115	73	**s**
52	34	**4**	84	54	**T**	116	74	**t**
53	35	**5**	85	55	**U**	117	75	**u**
54	36	**6**	86	56	**V**	118	76	**v**
55	37	**7**	87	57	**W**	119	77	**w**
56	38	**8**	88	58	**X**	120	78	**x**
57	39	**9**	89	59	**Y**	121	79	**y**
58	3A	**:**	90	5A	**Z**	122	7A	**z**
59	3B	**;**	91	5B	**[**	123	7B	**{**
60	3C	**<**	92	5C	****	124	7C	**|**
61	3D	**=**	93	5D	**]**	125	7D	**}**
62	3E	**>**	94	5E	**^**	126	7E	**~**
63	3F	**?**	95	5F	**_**	127	7F	(delete)

Примечание. Символ с кодом 127 в некоторых терминалах может использоваться как управляющий символ, соответствующий нажатию на клавиатуре клавиши Delete.

Приложение C. Словарь AFS Level-0

В этом приложении вы найдете все слова Форта, доступные в AFS уровня L0.

Для удобства слова разделены по группам операций.

Для каждого слова приводится его имя в словаре, стековая диаграмма, краткое описание и номер страницы, на которой приводится описание слова.

Таблица C.1 Слова для арифметических операций

Заголовок слова	Описание слова	Стр.
* (n1 n2 -- n3)	Перемножает числа на вершине стека	89
*/ (n1 n2 n3 -- n4)	Перемножает второе и третье число в стеке, произведение, двойное число, делит на число с вершины стека	92
*/MOD (n1 n2 n3 -- n4 n5)	Перемножает второе и третье число в стеке, произведение, двойное число, делит на число с вершины стека. Возвращает частное на вершине стека и остаток во втором числе стека	93
+ (n1 n2 -- n3)	Складывает числа на вершине стека	89
+! (n a-addr --)	Прибавляет число к содержимому указанной ячейки	90
- (n1 n2 -- n3)	Вычитает число на вершине стека из второго числа в стеке	89
/ (n1 n2 -- n3)	Делит второе число в стеке на число с вершины стека	89
/MOD (n1 n2 -- n3 n4)	Делит второе число в стеке на число с вершины стека, возвращает частное на вершине стека и остаток во втором числе стека	93
1+ (n1 -- n2)	Инкрементирует число на вершине стека	89
1- (n1 -- n2)	Декрементирует число на вершине стека	89
2* (n1 -- n2)	Умножает на 2 число на вершине стека	89
2/ (n1 -- n2)	Делит на 2 число на вершине стека	89
ABS (n -- u)	Вычисляет абсолютное значение числа на вершине стека	90
DABS (d -- ud)	Вычисляет абсолютное значение двойного числа на вершине стека	90
DNEGATE (d1 -- d2)	Изменяет знак двойного числа на вершине стека на противоположный	90
M* (n1 n2 -- d)	Перемножает два числа со знаком на вершине стека, возвращая результат в двойном числе со знаком	92
MOD (n1 n2 -- n3)	Вычисляет остаток от деления второго числа в стеке на число с вершины стека	93
NEGATE (n1 -- n2)	Изменяет знак числа на вершине стека на противоположный	90
S>D (n -- d)	Преобразует число со знаком в равное число двойной длины	91
U>D (u -- ud)	Преобразует число без знака в равное число двойной длины	91
UM* (u1 u2 -- ud)	Перемножает два числа без знака на вершине стека, возвращая результат в двойном числе без знака	92
UM/MOD (ud u1 -- u2 u3)	Делит двойное число на число с вершины стека. Возвращает частное на вершине стека и остаток во втором числе стека. Все числа без знака	93

Таблица C.2 Слова для логических операций

Заголовок слова	Описание слова	Стр.
<<< (x1 u -- x2)	Сдвигает влево второе число в стеке на число битов, лежащее на вершине стека	96
>>> (x1 u -- x2)	Сдвигает вправо второе число в стеке на число битов, лежащее на вершине стека	96
AND (x1 x2 -- x3)	Логическое умножение (операция И) двух чисел в стеке	94
CLR (x1 x2 -- x3)	Сбрасывает биты числа x1, если в маске x2 соответствующие биты установлены в 1	96
NOT (x1 -- x2)	Инвертирование битов числа на вершине стека	94
OR (x1 x2 -- x3)	Логическое сложение (операция ИЛИ) двух чисел в стеке	94
XOR (x1 x2 -- x3)	Исключающее ИЛИ двух чисел в стеке	94

Таблица C.3 Операторы сравнения

Заголовок слова	Описание слова	Стр.
0< (n1 -- flag)	Возвращает флаг "истина", если число на вершине стека меньше нуля (отрицательное)	100
0<> (n1 -- flag)	Возвращает флаг "истина", если число на вершине стека не ноль	100
0= (x1 -- flag)	Возвращает флаг "истина", если число на вершине стека - ноль	100
0> (n1 -- flag)	Возвращает флаг "истина", если число на вершине стека больше нуля (положительное)	100
< (n1 n2 -- flag)	Возвращает флаг "истина", если второе число в стеке меньше числа на вершине	99
<> (x1 x2 -- flag)	Возвращает флаг "истина", если числа в стеке не равны	99
= (x1 x2 -- flag)	Возвращает флаг "истина", если числа в стеке равны	99
> (n1 n2 -- flag)	Возвращает флаг "истина", если второе число в стеке больше числа на вершине	99
MAX (n1 n2 -- n1|n2)	Оставляет на вершине стека большее из двух чисел	99
MIN (n1 n2 -- n1|n2)	Оставляет на вершине стека меньшее из двух чисел	99
U< (u1 u2 -- flag)	Возвращает флаг "истина", если второе число в стеке меньше числа на вершине. Числа сравниваются без знака	100
U> (u1 u2 -- flag)	Возвращает флаг "истина", если второе число в стеке больше числа на вершине. Числа сравниваются без знака	100

Таблица С.4 Слова для управления выполнением программы

Описание слова	Описание слова	Стр.
+LOOP (S: n1 --) (R: n2 n3 -- n2 n4 |)	Получает в стеке параметров величину приращения индекса цикла. Приращение складывается с текущим значением индекса, результат сравнивается с пределом. Если новое значение индекса пересекло границу между значением предела и значением предела минус один, выполняется выход из цикла (из стека возвратов удаляются значения индекса и предела). Иначе выполняется переход в начало цикла, отмеченное словом DO. Значения приращения, индекса и предела - числа со знаком	108
ABORT (--)	Прекращает выполнение программы и передает управление Форт-системе	112
AGAIN (--)	Направляет выполнение в точку, отмеченную последним словом BEGIN, не проверяя каких-либо условий	104
BEGIN (--)	Отмечает начало цикла с неопределенным числом повторений	102
DO (S: x1 x2 --) (R: -- x1 x2)	Отмечает начало счетного цикла при компиляции. При выполнении извлекает с вершины стека параметров начальное значение и предел счетчика цикла, размещая их в стеке возвратов для последующего использования	107
ELSE (--)	Отмечает альтернативную ветвь структуры условного выполнения программы	101
EXECUTE (xt --)	Получает токен на вершине стека и выполняет соответствующее слово	123
EXIT (R: x --)	Досрочно завершает выполнение слова, извлекая адрес возврата из стека возвратов	111
I (-- x)	Возвращает текущее значение счетчика цикла на вершине стека параметров	109
IF (flag --)	Открывает структуру условного выполнения программы. Извлекает логический флаг из стека	101
J (-- x)	Возвращает текущее значение счетчика внешнего цикла на вершине стека параметров	109
K (-- x)	Возвращает текущее значение счетчика внешнего цикла второго уровня на вершине стека параметров	109
LEAVE (R: x1 x2 --)	Удаляет из стека возвратов предел и индекс цикла и продолжает выполнение программы со слова, следующего за ближайшим словом LOOP или +LOOP	110
LOOP (R: u1 u2 -- u1 u3 |)	Инкрементирует индекс и сравнивает его с пределом цикла. Если индекс меньше предела цикла, выполнение программы продолжается с первого слова после DO. Если индекс стал равен пределу цикла, выполняется выход из цикла (из стека возвратов удаляются значения индекса и предела)	107
QUIT (--)	Прекращает выполнение программы в интерактивном сеансе программирования и передает управление Форт-системе	112
REPEAT (--)	При компиляции отмечает точку выхода для WHILE. Направляет выполнение в точку, отмеченную последним словом BEGIN	103
THEN (--)	Закрывает структуру условного выполнения программы	101
UNLOOP (R: u1 u2 --)	Удаляет из стека возвратов предел и индекс цикла. Используется для последующего выполнения слова EXIT внутри счетного цикла.	112

Описание слова	Описание слова	Стр.
UNTIL (flag --)	Извлекает из стека логический флаг. Если флаг "истина", продолжает выполнение со следующего за ним слова. Если флаг "ложь", направляет выполнение в точку, отмеченную последним словом BEGIN	102
WHILE (flag --)	Извлекает из стека логический флаг. Если флаг "истина", продолжает выполнение со следующего за ним слова. Если флаг "ложь", передает выполнение слову, следующему за ближайшим словом REPEAT	103

Таблица C.5 Слова для операций в стеке параметров

Заголовок слова	Описание слова	Стр.
2DROP (x1 x2 --)	Удаляет пару ячеек с вершины стека	67
2DUP (x1 x2 -- x1 x2 x1 x2)	Создает копию пары ячеек на вершине стека	68
2OVER (x1 x2 x3 x4 -- x1 x2 x3 x4 x1 x2)	Создает на вершине стека копию второй пары ячеек	70
2SWAP (x1 x2 x3 x4 -- x3 x4 x1 x2)	Меняет местами две верхних пары ячеек стека	69
?DUP (x -- 0 | x x)	Создает копию ячейки на вершине стека, если число в ячейке не ноль	68
DEPTH (-- n)	Возвращает число занятых ячеек в стеке на момент выполнения	74
DROP (x --)	Удаляет ячейку с вершины стека	67
DUP (x -- x x)	Создает копию ячейки на вершине стека	68
OVER (x1 x2 -- x1 x2 x1)	Создает на вершине стека копию второй ячейки	70
PICK (xu .. x1 x0 u -- xu .. x1 x0 xu)	Создает копию указанной ячейки стека на вершине стека	72
ROLL (xu .. x1 x0 u -- xu-1 .. x1 x0 xu)	Циклически переставляет заданное количество верхних ячеек стека	73
ROT (x1 x2 x3 -- x2 x3 x1)	Циклически переставляет три верхние ячейки стека	71
SWAP (x1 x2 -- x2 x1)	Меняет местами две верхних ячейки стека	69

Таблица C.6 Слова для операций в стеке возвратов

Описание слова	Описание слова	Стр.
2>R (dx --) (R: -- dx)	Перемещает двойное число из стека параметров в стек возвратов	113
2R> (-- dx) (R: dx --)	Перемещает двойное число из стека возвратов в стек параметров	113
2R@ (-- dx) (R: dx -- dx)	Помещает копию двойного числа из стека возвратов в стек параметров	113
>R (x --) (R: -- x)	Перемещает ячейку из стека параметров в стек возвратов	113
R> (-- x) (R: x --)	Перемещает ячейку из стека возвратов в стек параметров	113
R@ (-- x) (R: x -- x)	Помещает копию ячейки из стека возвратов в стек параметров	113

Таблица С.7 Слова для операций с памятью данных

Заголовок слова	Описание слова	Стр.
! (x a-addr --)	Записывает число в ячейку с заданным адресом	77
, (x --)	Резервирует ячейку в области динамических данных программы и записывает в нее число из стека	86
2! (xd a-addr --)	Записывает число двойной длины в пару ячеек с указанным адресом	80
2@ (a-addr -- xd)	Возвращает в стеке содержимое пары ячеек с указанным адресом	80
2VARIABLE (--)	Создает в словаре запись для переменной двойной длины. Резервирует две ячейки в текущей позиции указателя данных и записывает в них 0. При выполнении слова-переменной адрес млашей ячейки помещается в стек	80
@ (a-addr – x)	Возвращает в стеке содержимое ячейки с указанным адресом	77
ALIGN (--)	Выравнивает значение указателя данных программы	85
ALIGNED (addr – a-addr)	Выравнивает указанный адрес на границу ячейки	85
ALLOT (n --)	Смещает указатель данных на заданное число байтов	78
C! (char addr --)	Записывает указанное значение в байт с заданным адресом	85
C, (char --)	Резервирует байт в области динамических данных программы и записывает в него значение из стека	86
C@ (addr – char)	Возвращает в стеке содержимое байта с указанным адресом	85
CELL+ (a-addr1 – a-addr2)	Прибавляет размер ячейки к адресу	86
CELLS (n1 – n2)	Возвращает размер указанного количества ячеек в байтах	86
CONSTANT (x --)	Создает в словаре запись константы, присваивая ей значение с вершины стека. При выполнении созданного слова-константы это значение помещается в стек	81
CREATE (--)	Создает в словаре запись слова и связывает с ним значение указателя данных на момент создания слова. При выполнении созданного слова это значение помещается в стек	78
HERE (-- addr)	Возвращает текущее значение указателя данных	77
IDATA! (addr --)	Устанавливает границу инициализированных данных	84
TO <name> (x --)	Записывает число из стека в переменную с указанным именем, созданную словом VALUE	83
VALUE (x --)	Создает в словаре запись для переменной. Резервирует ячейку в текущей позиции указателя данных и записывает в нее число с вершины стека. При выполнении слова-переменной в стек помещается ее текущее значение. Изменить значение переменной можно с помощью слова TO	83
VARIABLE (--)	Создает в словаре запись для переменной. Резервирует ячейку в текущей позиции указателя данных и записывает в нее 0. При выполнении слова-переменной адрес ячейки помещается в стек	80

Таблица С.8 Системные переменные и константы

Описание слова	Описание слова	Стр.
BIN (--)	Устанавливает двоичную систему счисления	49
BL (-- n)	Константа, помещает в стек код символа "пробел" (32_{10} , 20_{16})	81
CR (-- n)	Константа, помещает в стек код управляющего символа "возврат курсора" (13_{10} , $0D_{16}$)	81
DECIMAL (--)	Устанавливает десятичную систему счисления	49
HEX (--)	Устанавливает шестнадцатеричную систему счисления	49
LF (-- n)	Константа, помещает в стек код управляющего символа "перевод строки" (10_{10} , $0A_{16}$)	81
NUL (-- n)	Константа, помещает в стек ноль	81
RADIX! (n --)	Устанавливает значение основания системы счисления	49
RADIX@ (-- n)	Возвращает текущее значения основания системы счисления	49
TRUE (-- flag)	Константа, помещает в стек логический флаг "истина" (все биты ячейки установлены в значение 1)	81

Таблица С.9 Слова компилятора

Описание слова	Описание слова	Стр.
2LITERAL (xx --)	Компилирует двойное число из стека в текущее определение через двоеточие	121
: <name> (--)	Создает запись нового слова в словаре и переключает систему в режим компиляции	25
; (--)	Завершает компиляцию определения нового слова, начатого двоеточием	25
ALIAS <name1 . . nameN> (--)	Создает в словаре записи слов-псевдонимов последней созданной константы. Псевдонимы могут перечисляться списком	82
CTABLE <name> (a-addr n --)	Создает запись нового слова в словаре и компилирует в метакод массив символов, длина и начальный адрес которого переданы в стеке. При выполнении созданного слова на вершине стека передается индекс масиива, а возвращается соответствующий индексу символ	87
LITERAL (x --)	Компилирует число из стека в текущее определение через двоеточие	121
POSTPONE <name> (--)	Компилирует слово немедленного выполнения в определение через двоеточие	120
TABLE <name> (a-addr n --)	Создает запись нового слова в словаре и компилирует в метакод массив констант, длина и начальный адрес которого переданы в стеке. При выполнении созданного слова на вершине стека передается индекс масиива, а возвращается соответствующее индексу значение константы	87
[(--)	Переключает систему в режим интерпретации	122
[CHAR] (--)	Компилирует в новое определение код первого символа следующего в потоке ввода	60
] (--)	Переключает систему в режим компиляции	122

Таблица С.10 Слова для управления конфигурацией системы

Описание слова	Описание слова	Стр.
SYS-CFG! (x n --)	Устанавливает значения параметров конфигурации и режимов работы системы	126
SYS-CFG@ (n -- x)	Возвращает текущие значения параметров конфигурации и режимов работы системы	128

Таблица С.11 Служебные слова

Описание слова	Описание слова	Стр.
' <name> (xt --)	Находит слово в словаре и возвращает его токен в стеке	125
BYE (--)	Сохраняет текущий сеанс программирования и завершает работу интерактивной среды	28
MS (u --)	Приостанавливает выполнение программы на указанное число миллисекунд	105
QUIET-MODE (flag --)	Если флаг имеет значение "истина", включает "тихий" режим интерактивной среды программирования – система выводит на консоль только сообщения об ошибках. При значении флага "ложь" отключает "тихий" режим интерактивной среды программирования – на каждый ввод строки в режиме интерпретации система отвечает "Ok" (если нет ошибок)	43
SAVE-PROGRAM (--)	Сохраняет программу пользователя в формате для автономного выполнения и тиражирования. Завершает сеанс программирования	44
SAVE-SESSION (--)	Сохраняет текущий сеанс программирования	27
SMUDGE <name> (--)	Удаляет слово в словаре пользователя	27
SYS-TICK (-- ud)	Возвращает значение счетчика тиков системного таймера	129
SYS-TIME (-- ud)	Возвращает значение счетчика миллисекунд системного времени	129
WORDS (--)	Выводит на консоль словарь пользователя и словарь Форт	26
\ (--)	Начало комментария, завершающегося символом \	39
\\ (--)	Начало строки комментария (до конца строки)	38

Таблица C.12 Слова для ввода, обработки и вывода текста

Описание слова	Описание слова	Стр.
# (ud1 -- ud2)	Делит двойное число без знака на основание системы счисления. Частное от деления возвращается в стеке, остаток преобразуется в символ ASCII и помещается в буфер форматированного вывода (FOB). Указатель FOB сдвигается на один символ к началу строки	114
#> (ud -- c-addr u)	Удаляет из стека двойное число, возвращает длину строки (на вершине стека) и адрес первого символа строки в буфере форматированного вывода (FOB)	114
#S (ud1 -- ud2)	Делит двойное число без знака на основание системы счисления. Остаток преобразуется в символ ASCII и помещается в буфер форматированного вывода (FOB). Указатель FOB сдвигается на один символ к началу строки. Перечисленные операции повторяются, пока частное от деления не станет равно нулю	115
. (n --)	Преобразует число со знаком в текстовую строку и выводит на консоль. В конце строки выводится пробел	59
." (--)	Компилирует строку, завершенную символом "кавычка", внутри определения слова через двоеточие. Текст выводится на консоль во время выполнения созданного слова	60
<# (ud -- c-addr u)	Устанавливает в исходное состояние указатель в буфере форматированного вывода (перемещает его в конец буфера) и обнуляет длину строки в буфере	114
ACCEPT (addr n1 -- n2)	Получает в стеке длину и адрес буфера в секции данных для ввода текстовой строки. Принимает от консоли тестовую строку с редактированием. После приема управляющего символа возврата курсора (CR) завершает ввод текста и возвращает в стеке длину полученной строки	63
CHAR (-- n)	Помещает в стек числовое значение кода ASCII первого символа, следующего в потоке текстового ввода	60
COUNT (addr1 -- addr2 u)	Получает в стеке адрес счетной строки. Возвращает длину строки на вершине стека и адрес первого символа строки	60
D. (d --)	Преобразует двойное число со знаком в текстовую строку и выводит на консоль. В конце строки выводится пробел	59
EMIT (n --)	Получает код ASCII на вершине стека. Выводит соответствующий символ символ на консоль	60
HOLD (n --)	Получает на вершине стека код ASCII символа, который записывается в FOB в текущей позиции указателя. Указатель FOB смещается на один символ к началу строки	116
KEY (-- n)	Ожидает поступления символа от консоли. Помещает в стек код ASCII символа, введенного с клавиатуры	62
KEY? (-- flag)	Проверяет наличие данных в буфере ввода системной консоли. Возвращает логический флаг "ложь", если буфер пуст	62
NL (--)	Выводит на консоль управляющие символы возврата курсора и перевода строки	61
OMIT (char --)	Просматривает поток ввода и удаляет все символы, пока не встретится первый символ-разделитель (также удаляется)	63

Описание слова	Описание слова	Стр.
SIGN (n --)	Получает на вершине стека число и проверяет его знак. Если число отрицательное, в буфер FOB помещается символ "минус" в текущей позиции указателя. Указатель FOB смещается на один символ к началу строки	116
SPACE (--)	Выводит на консоль символ "пробел"	61
TO-NUMBER (ud1 c-addr1 u1 -- ud2 c-addr2 u2 u3)	Выполняет преобразование текстовой строки в число. См.описание по номеру страницы	117
TYPE (addr u --)	Получает на вершине стека длину строки и адрес первого символа строки. Выводит строку на консоль	60
U. (u --)	Преобразует число без знака в текстовую строку и выводит на консоль. В конце строки выводится пробел	59
UD. (d --)	Преобразует двойное число без знака в текстовую строку и выводит на консоль. В конце строки выводится пробел	59
WORD (char -- c-addr)	Выделяет в поток ввода слово, ограниченное символом-разделителем. Сохраняет полученную строку по адресу HERE. Возвращает адрес счетной строки	63

Таблица C.13 Слова для доступа к устройствам ввода-вывода

Заголовок слова	Описание слова	Стр.
ADC-CHAN (n dev -- x)	Возвращает результат измерения в заданном канале АЦП	142
ADC-CTRL (param n dev --)	Управляет аналогово-цифровым преобразователем (АЦП) и его драйвером	142
ADC-INIT (mode dev --)	Устанавливает исходное состояние АЦП и его драйвера	141
ADC-SCAN (a-addr mask dev --)	Выполняет измерения в задданных каналах АЦП и сохранят результаты в указанный буфер данных	143
ADC-STAT (n dev -- x)	Возвращает данные состояния АЦП	144
CON-CTRL (param n --)	Устанавливает значения параметров конфигурации и режимов работы текстовой консоли AFS	129
CON-STAT (n -- x)	Возвращает данные состояния и текущих настроек текстовой консоли AFS	130
I2C-CTRL (param n dev --)	Управляет блоком I2C и его драйвером	149
I2C-INIT (ownadr speed mode dev --)	Устанавливает исходное состояние блока I2C и его драйвера	148
I2C-RECV (addr1 n1 addr2 n2 dest dev --)	Принимает данные от ведомого устройства I2C	150
I2C-REG! (xx reg dev --)	Записывает число в регистр I2C	151
I2C-REG@ (reg dev -- xx)	Возвращает содержимое регистра I2C	151
I2C-SEND (addr n dest dev --)	Передает данные от ведущего устройства I2C к ведомому	150
PIO-BIT-SR (x1 x2 port --)	Изменяет состояние отдельных выходов порта PIO	140

Заголовок слова	Описание слова	Стр.
PIO-CFG! (xx param port --)	Записывает параметры настройки в регистры конфигурации портов ввода-вывода (PIO)	132
PIO-CFG@ (param port -- xx)	Возвращает значение параметра настройки PIO	132
PIO-IN (port -- x)	Возвращает состояние входов порта PIO	139
PIO-OUT! (x port --)	Записывает число в регистр выходных данных порта PIO	139
PIO-OUT@ (port -- x)	Возвращает текущее значение регистра выходных данных порта PIO	139
SPI-CTRL (param n dev --)	Управляет блоком SPI и его драйвером	147
SPI-INIT (data speed mode dev --)	Устанавливает исходное состояние блока SPI и его драйвера	146
SPI-REG! (x reg dev --)	Записывает число в регистр SPI	148
SPI-REG@ (reg dev -- x)	Возвращает содержимое регистра SPI	148
TIM-CHAN-SET (mode chan dev --)	Устанавливает режим работы счетного канала таймера-счетчика	160
TIM-CTRL (param n dev --)	Управляет блоком таймера-счетчика и его драйвером	161
TIM-INIT (ARR PSC mode dev --)	Устанавливает исходное состояние таймера-счетчика и его драйвера	160
TIM-REG! (x reg dev --)	Записывает число в регистр таймера-счетчика	161
TIM-REG@ (reg dev -- x)	Возвращает содержимое регистра таймера-счетчика	161
UART-CTRL (param n dev --)	Управляет блоком UART и его драйвером	156
UART-GETC (dev -- char)	Возвращает один символ из буфера приемника UART	158
UART-INIT (baud mode dev --)	Устанавливает исходное состояние UART и его драйвера	154
UART-PUTC (char dev --)	Передает один символ в UART	158
UART-REG! (xx reg dev --)	Записывает число в регистр UART	158
UART-REG@ (reg dev -- xx)	Возвращает содержимое регистра UART	158