Учебное пособие по языку ФОРТ

Содержание

1. ВВЕДЕНИЕ ⬆️ К содержанию

Язык программирования Форт (от английского FORTH) был изобретен Чарльзом Муром в 70-х годах для создания программного обеспечения управляющих устройств. В настоящее время Форт широко используется при решении следующих задач:

```
разработка и тестирование встроенного оборудования;
управление станками, роботами, медицинскими приборами;
разработка трансляторов и операционных систем;
системы управления базами данных;
задачи машинной графики;
экспертные системы, в том числе экспертные системы реального времени.
```

В отличие от других языков высокого уровня, Форт обеспечивает программисту полный доступ к машине и не пытается оградить его от ошибок. Однако, модульность, а также расширяемость языка, позволяющая программисту вводить конструкции со встроенными средствами контроля, дает возможность создавать высоконадежные программы.

Форт использует обратную польскую (постфиксную) запись, при которой операнды предшествуют операции. Хотя такая запись непривычна и может показаться неудобной, она существенно уменьшает затраты на организацию вызовов подпрограмм.

Код, получаемый компилятором Форта, исключительно компактен, даже по сравнению с машинным языком. Особенно это заметно на больших программах.

Форт-система, в основном, написана на самом языке Форт. Она занимает от 8 до 16 Кбайтов в зависимости от предоставляемых возможностей (таких, как встроенный ассемблер, экранный редактор, взаимодействие с файловой системой).

Программы на языке Форт реентерабельны, допускают рекурсию. Программист может написать программу в машинных командах на встроенном в Форт-систему ассемблере и в дальнейшем использовать ее как обычную подпрограмму. Вследствие этого, Форт можно применять для создания программ непосредственного управления аппаратурой.

Форт-система - автономная система. Она может работать как на "голом" оборудовании, так и под управлением операционной системы (например, CP/M, MS-DOS).

Форт является диалоговым языком, то есть команды выполняются Форт-системой сразу, как только Вы их введете с клавиатуры и нажмете клавишу ввода. Ответ "ok" является подтверждением того, что запрос выполнен, и приглашением продолжать работу.

Как правило, термин "печатает" ("выводит на печать") далее в тексте документа будет означать вывод на экран дисплея, если иное не оговорено специально.

Содержание

Основной структурной единицей языка является Форт-слово (или просто слово), которое может быть составлено из любых символов, кроме пробела. Маленькая и большая буквы - это два разных символа. Пробел разделяет слова в Форт-программе. Максимально допустимое количество символов в слове зависит от конкретной реализации Форт-системы. Программирование на языке Форт сводится к определению новых, специализированных для целей пользователя, слов в терминах ранее введенных слов. Некоторый начальный набор слов написан непосредственно в терминах машинного языка конкретного компьютера.

Программист может определять свои собственные операции, создавать специальные типы данных и, таким образом, развивать язык в сторону конкретных приложений. Естественным результатом программирования является создание приспособленного для решения задач пользователя языка.

Обычно текст определения нового слова помещается в двух-трех строках экрана. После ввода такое определение компилируется (это называется режимом компиляции). Успешно скомпилированное слово заносится в так называемый словарь. Теперь можно вводить с терминала его имя, что задает ИСПОЛНЕНИЕ СЛОВА, то есть выполнение указанных им действий. Система при этом переходит в

режим исполнения. Обычный режим диалога - это режим исполнения. При попытке исполнить еще не определенное слово система печатает его имя с комментарием '-?'. Скомпилированное новое слово можно использовать в дальнейших определениях.

Далее появляющиеся в тексте документа Форт-слова будут заключаться в " (двойная кавычка).

Настоящее пособие ориентировано на последний принятый международный стандарт - Стандарт-83. Стандарт фиксирует определенный набор слов, их имена и функции. Таким образом, программы, удовлетворяющие стандарту, могут использоваться с любыми стандартными Форт-системами. Конкретная реализация может отличаться от стандарта, что обычно указывается в документации. Список слов в конкретной Форт-системе, в том числе и вновь определенных, выводит на экран слово "WORDS". Вместе с именами слов иногда даются их адреса в словаре. Последнее определенное слово будет верхним в распечатке.

Содержание

В языке Форт арифметическим или основным стеком (или просто стеком) называется участок оперативной памяти, используемый для размещения ЦЕЛЫХ чисел - аргументов и результатов операций языка.

На каждое число в стеке отводится 2 байта. Числа на стеке могут восприниматься различным образом в зависимости от того, какое слово их использует. Обычно они трактуются как числа из диапазона от -2**15 до 2**15-1, но есть слова, которые воспринимают их как числа от 0 до 2**16-1.

Хранимые в стеке числа упорядочены по положению. Стек функционирует по принципу "последним занесен - первым выбран" (LIFO). Будем говорить, что при добавлении числа оно заносится СПРАВА от имеющихся, начиная от ДНА стека; при удалении снимается крайнее правое число с ВЕРШИНЫ стека.

При пользовании стеком вся ответственность за его состояние ложится на программиста. Для того чтобы зрительно согласовать стековые изменения при выполнении слов, удобно применять так называемую стековую нотацию. Действие слов будем показывать на диаграмме

```
    стек ДО операции  -->  стек ПОСЛЕ операции
```

причем не затрагиваемую операцией часть стека будем изображать многоточием. Для содержимого стека введем следующие обозначения:

малая латинская буква - значение в общем смысле n - число d - число двойной длины c - символ addr - адрес памяти

Выполнение слова, представляющего собой запись числа, добавляет в стек это число. Слово "." (точка) снимает число с вершины стека и печатает его. Очень полезное слово ".S" печатает весь стек, оставляя его неизменным.

Некоторые операции со стеком:

DUP ... a --> ... a a DROP ... a --> ... SWAP ... a b --> ... b a OVER ... a b --> ... a b a ROT ... a b e --> ... e e a -ROT ... a b e --> ... e a b NIP ... a b --> ... b TUCK ... a b --> ... b a b 2DROP ... a b --> ... 2DUP ... a b --> ... a b a b 2SWAP ... a b e f --> ... e f a b 2OVER ... a b e f --> ... a b e f a b PICK ... a {n чисел} n --> ... a {те же n чисел} a ROLL ... a {n чисел} n --> ... {те же n чисел} a

В частности, "0 PICK" эквивалентно "DUP", а "1 PICK" слову "OVER"; "1 ROLL" эквивалентно "SWAP", а "2 ROLL" слову "ROT".

Слово "DEPTH" кладет в стек число, равное глубине стека - количеству элементов, находившихся в стеке перед исполнением этого слова.

Проследим, например, выполнение следующего текста:

```
1  2       3  DUP       DEPTH -ROT  2OVER  .S
```

Выпишем текст в столбик, сопровождая каждое слово состоянием стека после его исполнения:

```
1       (  1  )
2       (  1  2  )
3       (  1  2  3  )
DUP     (  1  2  3  3  )
DEPTH   (  1  2  3  3  4 )
-ROT    (  1  2  4  3  3  )
2OVER   (  1  2  4  3  3  2  4 )
```

Иногда данные на стеке, сформированные одним словом и используемые потом другим словом, будем называть параметрами.

Содержание

4. АРИФМЕТИЧЕСКИЕ-ОПЕРАЦИИ

Арифметический стек - основное поле для выполнения арифметических действий и хранения промежуточных результатов вычислений. Надо только помнить, что знак операции (точнее слово, обозначающее операцию) пишется ПОСЛЕ того, как аргументы в стеке уже размещены.

Текст

13 3 -

помещает в стек число 10, так как слово "-" (минус) извлекает из стека два числа, сперва вычитаемое, потом уменьшаемое, и помещает в стек их разность:

- ```
  ...  a  b   -->       ...  a-b
  ```

С другими операциями все обстоит аналогично
  ```

- ```
  ...  a  b   -->       ...  a+b
  ```
- ```
  ...  a  b   -->       ...  a*b
  ```

ABS ... a --> ... |a| NEGATE ... a --> ... -a / ... a b --> ... целая часть MOD ... a b --> ... остаток /MOD ... a b --> ... остаток целая часть

В трех последних словах имеются в виду остаток и целая часть частного от деления a на b. Так, при делении 26 на 7 имеем:

```
26  7        /    -->  3
26  7       MOD   -->  5
26  7       /MOD  -->  5  3
```

Имеются специальные слова для действий с 1 и 2 (они выполняются немного быстрее)

1+ ... a --> ... a+1

Аналогично работают "1-", "2+", "2-", "2*", "2/".

Следующие слова выполняют поразрядные логические операции над двоичным представлением чисел; в этих операциях числа трактуются как наборы из шестнадцати битов.

AND ... a b --> ... a AND b (И) OR ... a b --> ... a OR b (ИЛИ) NOT ... a --> ... NOT a (НЕ) XOR ... a b --> ... a XOR b (ИСКЛЮЧАЮЩЕЕ ИЛИ)

Содержание

5. ОПРЕДЕЛЕНИЕ-НОВЫХ-СЛОВ

Одно из главных достоинств языка Форт заключается в его расширяемости, то есть программист может расширять базовый набор слов Форт-системы, определяя новые слова через уже определенные.

Слова, которые указывают Форт-системе, что пользователь заводит новое слово, называются ОПРЕДЕЛЯЮЩИМИ словами. Наиболее употребительное определяющее слово - это ":" (двоеточие). Формально соответствующее определение (или описание) выглядит следующим образом:

```
                    : имя тело ;

"имя" как раз и есть новое придуманное слово,
"тело" представляет собой перечень через пробелы уже имеющихся в Форт-
системе слов; совокупность их функций образует те действия, которые будут
выполнены при исполнении данного слова,
наличие слова ";" (точка с запятой) обязательно, оно завершает определение.
```

Например, текст

: S2 DUP * SWAP DUP * + ;

определяет слово "S2", вычисляющее сумму квадратов двух чисел из стека

S2 ... a b --> ... a*a+b*b

Если в теле определения встретятся слова, которых нет в словаре, система напечатает ошибочное слово со знаком '-?'. При этом вся наработанная информация о новом слове исчезает.

При разработке новых слов нужно внимательно следить за изменениями стека. Рекомендуется писать комментарии. Комментарий начинается словом "(" (открывающая скобка), и система пропускает следующий за ним текст до первого символа ")" (закрывающая скобка).

Скомпилированные слова сразу же могут использоваться и в вычислениях и в определении других слов. Например, сумму четырех квадратов можно определить так:

: S4 (a b c d --> a*a+b*b+c*c+d*d) S2 -ROT S2 + ;

Можно отменить уже определенное слово ("забыть" его), но при этом забываются также и все слова, определенные позже него. Для этого используется слово "FORGET". Например, действие

```
FORGET S2
```

"забудет" S2 и все определенные позже слова.

Прежде, чем заводить новое слово, стоит убедиться, что его еще нет в словаре. Одному и тому же слову можно дать несколько определений с разным смыслом, но выполняться будет только последнее введенное. Однако прежнее определение не уничтожается. Если теперь выполнить слово "FORGET" с этим словом, то снова будет действовать прежнее определение. При отладке больших программ полезно иногда применять слово "FORGET", чтобы избежать переполнения словаря.

Надо помнить, что при вводе нового слова с клавиатуры его исходный текст пропадает. В словаре запоминается только скомпилированная форма. Чтобы внести изменения в уже определенное слово для перекомпиляции, приходится перенабирать его определение полностью или использовать внешнюю память.

При завершении сеанса работы с Форт-системой, что обычно задается словом "BYE", из словаря исчезают все новые слова, определенные в этом сеансе. Способ сохранения наработанной версии Форт-системы зависит от конкретной реализации.

Приведем еще пару примеров. Слово "8MOD" эквивалентно тексту "8 MOD" , но использует логические операции. Слово "LAST1" выделяет в двоичном разложении числа младшую единицу.

: 8MOD 7 AND ; : LAST1 DUP DUP 1- XOR AND ;

Содержание

Кодофайлом будем называть участок памяти, в котором располагаются набор слов Форт-системы и новые скомпилированные слова, написанные пользователем. Здесь же размещаются константы и переменные. Память занимается в направлении возрастания адресов, при этом свободная память находится в конце словаря. Иногда два соседних байта называют ячейкой. Тогда адресом ячейки считается адрес младшего байта (то есть байта с меньшим адресом). Мы будем называть ВЕРШИНОЙ СЛОВАРЯ первый свободный байт памяти. От программиста требуется особая осторожность при работе с памятью: изменения, записанные в ячейку с ошибочным адресом, могут нарушить функционирование Форт-системы так, что потребуется ее перезагрузка!

Вот некоторые стандартные слова для работы с кодофайлом:

```
HERE              ...      -->  ...  addr
```

На стек кладется адрес вершины кодофайла. С помощью этого слова можно определить, какой объем памяти требуется для любого фрагмента Вашей программы - надо сравнить значения "HERE" до компиляции и после нее.

```
ALLOT             ...  n   -->  ...
```

Резервируются n байтов свободной памяти: адрес вершины кодофайла увеличивается на n (а при n<0 уменьшается).

```
,                     ...  n   -->  ...
```

Занятие двух байтов в кодофайле и запись туда n.

```
!               ...  a        addr   -->  ...
```

Это слово (восклицательный знак, читается "запомнить") служит для записи значения по данному адресу.

```
@               ...  addr     -->  ...  a
```

Слово "@" (читается "взять") кладет в стек значение, хранящееся по адресу, лежащему на стеке. Сам адрес из стека при этом убирается.

```
+!              ...  a   addr    -->  ...
```

К числу, расположенному по адресу addr, прибавляется значение a. Результат сохраняется там же.

Содержание

6.2. КОНТЕКСТНЫЕ-СЛОВАРИ

Контекстный словарь (или просто словарь) - это связанный список слов, относящихся к некоторой единой теме. Новые определения записываются в один и тот же кодофайл в порядке их компиляции независимо от того, к какому контекстному словарю они объявлены принадлежащими. Однако поиск слова для его исполнения происходит только в пределах того контекстного словаря, на который в данный момент ориентирована Форт-система. Во-первых, это существенно ускоряет поиск. Во-вторых, слово с одним и тем же именем может быть определено в разных словарях с разными действиями.

Изначально в Форт-системе имеются три контекстных словаря: словарь языка Форт "FORTH", словарь редактора "EDITOR" и словарь ассемблера "ASSEMBLER". За их переключением между собой следит сама система. Два последних пока не обсуждаем. Обычно словарь "FORTH" является текущим в обоих смыслах:

```
в контексте поиска слов; такой словарь определяет переменная с именем
"CONTEXT";
в контексте присоединения новых слов; на этот словарь указывает переменная
с именем "CURRENT".
```

Для задания своего нового словаря надо ввести текст

```
                    VOCABULARY  имя
```

Теперь исполнение слова "имя" будет устанавливать этот словарь в контекст поиска до тех пор, пока явно не будет исполнено слово, являющееся именем другого словаря.

Если ввести текст

```
                  имя  DEFINITIONS
```

то указанный в нем словарь становится текущим для присоединения новых слов на период до явного аналогичного переобъявления.

Манипулирование со словарями требует от программиста аккуратности и бдительности.

Содержание

6.3. СЛОВАРНАЯ-СТАТЬЯ

Для каждого слова при его определении в кодофайле создается СЛОВАРНАЯ СТАТЬЯ. Она состоит из СИСТЕМНОЙ части, служащей для хранения и поиска слова, и ПРОГРАММНОЙ части, описывающей действия и информацию, связанные с этим словом.

В СИСТЕМНОЙ части выделяются:

```
ПОЛЕ ИМЕНИ - содержит имя слова.
ПОЛЕ СВЯЗИ - адрес словарной статьи предыдущего (по появлению определения)
слова в контекстном словаре; служит для организации цепного списка
словарных статей; поле связи первого определения в каждом словаре содержит
нуль.

В ПРОГРАММНУЮ часть включаются:
ПОЛЕ КОДА - иногда называется исполняемая часть; содержит (в той или иной
форме) вызов специальной программы - адресного интерпретатора - для
выполнения предписанных определением действий.
ПОЛЕ ПАРАМЕТРОВ - содержимое этой области зависит от типа слова; для данных
(переменные и т.п.) здесь находятся сами значения или просто
зарезервированная под них память; либо сюда помещаются адреса полей кодов
слов из тела определения данного слова.
```

Содержание

Для передачи данных от слова к слову можно использовать стек. Но для длительного хранения информации применяются переменные и константы.

Определяющее слово "CONSTANT" в тексте

```
                    CONSTANT  имя
```

определяет новое слово "имя" как константу со значением, равным числу на вершине стека, и удаляет из стека это число. В дальнейшем выполнение слова "имя" помещает это число в стек.

Специфическое для языка Форт преимущество использования в программе констант состоит в следующем. Скомпилированное определение, содержащее константу, занимает меньший объем памяти, чем то же определение с изображением самого числа. При неоднократном появлении числа в программе это становится существенным.

Как правило, в базовом наборе слов определены константы:

```
        0 CONSTANT FALSE    и    1 CONSTANT TRUE
        0 CONSTANT 0        и    1 CONSTANT 1
```

Определяющее слово "VARIABLE", которое используется в тексте

```
                    VARIABLE  имя
```

резервирует в словаре 2 байта под значение переменной "имя". Исполнение слова "имя" кладет в стек число - адрес зарезервированного места. Этот адрес может использоваться другими словами.

Пример. Текст

A @ 5 + B ! (A и B - переменные)

соответствует оператору "B:=A+5" других языков программирования.

Специального слова для организации привычной по другим алгоритмическим языкам конструкции массива в языке Форт нет. Ниже приводится один из возможных способов:

```
слово "2ALLOT" резервирует в кодофайле память под n чисел (число n берется
со стека) и кладет в стек адрес начала зарезервированного места

    : 2ALLOT             (  ... n        -->  ... a     )
      HERE  SWAP
      2*  ALLOT ;


слово для формирования элемента массива:

    : [I]       (  ... i        a  -->        ... a[i]  )
      OVER + + ;


заводится нужный массив

    N              (  N - константа - число элементов массива   )
    2ALLOT         (  на стеке будет адрес начала массива       )
    CONSTANT B     (  B - имя массива                           )


Если, например, поместить в стек номер нужного элемента в массиве B, то при
выполнении текста "B [I]" на стеке окажется адрес этого элемента.

Обратите внимание, что проверку корректности номера элемента массива этот
способ не обеспечивает и что элементы массива надо нумеровать с нуля.
```

Содержание

8. СИМВОЛЫ

Для представления символьной информации отводится по одному байту памяти на каждый символ. Таким образом, каждому символу сопоставляется число от 0 до 255, которое называется его КОДОМ. В разных ЭВМ используются разные кодировки.

Имеются слова для работы с отдельными символами. Надо учитывать, что код символа в стеке хранится в младшем байте ячейки.

```
C@                ...        addr          -->  ...  c
```

В стек помещается число, равное содержимому байта по адресу addr.

```
C!                ...        c  addr   -->  ...
```

В байт по адресу addr записывается символ "c".

```
C,                ...        c          -->  ...
```

Слово, аналогичное слову "," (запятая), но резервирующее (и записывающее) только один байт.

```
KEY           ...                 -->  ...  c         (ожидание)
```

При выполнении этого слова Форт-система переходит в режим ожидания, пока не будет нажата клавиша какой-либо литеры на клавиатуре дисплея. Код этой литеры и кладется в стек.

```
EMIT          ...        c          -->  ...
```

Символ "c" будет напечатан.

Константа "BL" помещает в стек код пробела.

Слово "C"" помещает в стек код первой следующей за ним литеры, не являющейся пробелом. Слово "C"" делает текст более наглядным, чем при непосредственном использовании кодов. Например, чтобы напечатать знак плюс, нужно выполнить текст

```
                    C" +  EMIT
```

Содержание

9. РАБОТА-С-УЧАСТКАМИ-ПАМЯТИ

Часто приходится выполнять действия сразу над большими участками памяти. Участок памяти в таких действиях определяется адресом его начального байта и длиной.

FILL ... addr n c --> ...

Содержимое n байтов начиная с адреса addr заполняется кодом "c".

BLANK ... addr n --> ...

Эквивалентно "FILL" с заполнением кодом пробела.

ERASE ... addr n --> ...

Эквивалентно тексту "0 FILL".

CMOVE ... addr1 addr2 n --> ...

Побайтное копирование участка в n байтов с началом addr1 по адресу addr2 в сторону увеличения адресов.

Слово "CMOVE>" отличается от "CMOVE" тем, что начинает запись с ПОСЛЕДНЕГО байта участков. Различие этих слов существенно при перекрытии участков. Вот небольшой пример. Пусть на вершине стека лежит адрес участка памяти в 12 байтов, где записан текст "Форт-система". Тогда исполнение

```
DUP  4  +  8  CMOVE
```

превратит этот текст в "ФортФортФорт", а если выполнить "CMOVE>" вместо "CMOVE", то получится "ФортФорт-сис". С помощью этих слов удобно размножать маленькие участки памяти внутри больших (например, числа внутри массивов).

Для заполнения участка памяти информацией непосредственно с клавиатуры имеется слово:

EXPECT ... addr n --> ...

Участок памяти от адреса addr длиной n байтов заполняется в сторону увеличения адресов вводимыми с клавиатуры символами до тех пор, пока не заполнится весь участок или пользователь не завершит ввод (нажав клавишу ввода). Переменная "SPAN" содержит число фактически введенных символов.

Содержание

10. СТРОКИ

СТРОКОЙ СО СЧЕТЧИКОМ (в дальнейшем просто СТРОКОЙ) называется участок памяти, в первом байте которого находится СЧЕТЧИК - байт, хранящий длину участка (без учета байта под счетчик). Но адресом строки считается адрес счетчика.

Слово " (двойная кавычка) употребляется в конструкции

" текст " ... --> ... addr

Текст текст будет превращен в строку, размещенную в памяти от адреса "HERE", адрес этой строки кладется в стек. Слово "COUNT" преобразует адрес строки в адрес и длину ее текста:

COUNT ... addr --> ... addr+1 n

а слово "TYPE" выводит текст (участок памяти) по его адресу и длине:

TYPE ... addr n --> ...

В качестве примеров работы со строками рассмотрим тексты:

" МОЛОДЕЦ" COUNT TYPE (напечатается МОЛОДЕЦ) " МОЛОДЕЦ" COUNT 3 - TYPE (напечатается МОЛО)

Разберите следующие примеры: слово "S," размещает в кодофайле строку с данным адресом, а слово "T," - ее текст. Оба слова оставляют на стеке адрес получившегося объекта.

: T, (a1 - адрес строки --> a2 - адрес текста в кодофайле) HERE SWAP (a2 a1) COUNT (a2 a1+1 n) HERE OVER ALLOT (a2 a1+1 n a2) SWAP CMOVE ; (a2)

: S, (a1 - адрес строки --> a2 - адрес строки в кодофайле) HERE SWAP DUP C@ 1+ (a2 a1 n+1) HERE OVER ALLOT (a2 a1 n+1 a2 - отведено n+1 байт) SWAP CMOVE ;

Слово "."" употребляется в конструкции

```
                ." текст "
```

при выполнении которой текст текст будет выведен на экран.

Упомянем еще несколько возможностей при выводе:

```
слово "CR" переводит строки,
слово "SPACE" вставляет в выходной текст пробел (т.е. оно эквивалентно "BL
EMIT").
```

Имя слова в поле имени его словарной статьи хранится в виде строки со счетчиком.

Содержание

11. УСЛОВНОЕ-ИСПОЛНЕНИЕ

При определении нового слова могут потребоваться знакомые Вам из других языков конструкции, организующие условное и циклическое исполнение. Имеются и соответствующие логические величины, принимающие традиционные значения "ИСТИНА" и "ЛОЖЬ". Эти значения представлены целыми числами, причем "ИСТИНА" соответствует числу -1 (двоичные разряды этого числа состоят из 16 единиц), а "ЛОЖЬ" соответствует числу 0 (16 двоичных нулей).

Логические значения получаются при выполнении специальных слов, предназначенных для сравнения чисел.

Слова арифметического сравнения:

```
      ...  a  b     -->  ...  a>b    т.е. при a>b   ИСТИНА
```

```
                                                иначе  ЛОЖЬ
```

< ... a b --> ... a<b = ... a b --> ... a=b 0= ... a --> ... a=0 0> ... a --> ... a>0 0< ... a --> ... a<0

Над логическими значениями можно совершать логические операции, описанные в п.4. Для того, чтобы можно было эффективно их использовать, управляющие конструкции на самом деле воспринимают числа из стека как логические значения таким образом:

0 - это "ЛОЖЬ", любое другое значение - "ИСТИНА".

Для организации условного исполнения в языке Форт предусмотрены слова "IF", "ELSE" и "THEN". Они используются в постфиксной форме в конструкциях:

```
          IF <часть-if> ELSE <часть-else> THEN
    и
          IF <часть-if> THEN
```

Слово "IF" берет из стека логическое значение, и в случае, если это "ИСТИНА", исполняет текст <часть-if>; в противном же случае исполняется <часть-else>, если она есть. Дальше управление передается на текст, следующий за "THEN". Заметим, что использование управляющих слов требует состояния компиляции; то есть их можно применять только в определениях новых слов и при этом ЦЕЛИКОМ ВНУТРИ ОДНОГО определения.

Пример. Стандартное слово "ABS" можно было бы определить так:

: ABS (... a --> ... |a|) DUP 0< IF NEGATE THEN ;

Пример другой конструкции разберите сами и постарайтесь улучшить:

: MAX (... a b --> max{a,b}) 2DUP > IF DROP ELSE SWAP DROP THEN ;

Слово "MAX" и аналогичное ему "MIN" входят в стандарт языка Форт.

Конструкции условного исполнения теоретически могут быть любой степени вложенности; ограничения зависят от конкретной Форт-системы; контроль за скобочными соответствиями "IF", "ELSE" и "THEN" оставлен за программистом.

Содержание

12. ЦИКЛЫ

Циклы любого типа также должны использоваться только в пределах одного определения. Глубина вложенности обычно зависит от конкретной Форт-системы; контроля соответствия нет. 12.1. ЦИКЛЫ-С-УСЛОВИЕМ

Для организации циклов такого типа в языке Форт предусмотрены слова "BEGIN" , "WHILE" , "REPEAT" и "UNTIL", используемые в постфиксной форме в конструкциях

```
      BEGIN <часть-begin> WHILE <часть-while> REPEAT     (1)
или
```

```
        BEGIN <часть-begin> UNTIL                                   (2)
```

В конструкции (2) после исполнения текста <часть-begin> слово "UNTIL" берет из стека оставленное этим текстом логическое значение; в том случае, если это значение "ЛОЖЬ", снова исполняется <часть-begin>, потом "UNTIL", и так далее; итерации прекращаются, когда "UNTIL" возьмет из стека значение "ИСТИНА".

Пример вычисления факториала может выглядеть так:

: FACT (... n --> ... n!) DUP 2 < IF DROP 1 (1 если n<2, то n!=1 ---) ELSE DUP (n n s=n k=n I) (Теперь лежащие в стеке числа будут представлять) (s - накопленное произведение и k - множитель) BEGIN 1- (s k'=k-1 <--- I) SWAP OVER * SWAP (s' k' s'= s*k I I) DUP 1 = (s k k=1 если k=1, то s=n! I I) UNTIL (n! 1 иначе повторить --- I) DROP THEN ; (n! <---)

Конструкция цикла типа (1) используется, когда в цикле есть действия, которые в заключительной итерации выполнять не надо: первоначально выполняется <часть-begin>, слово "WHILE" снимает со стека логическое значение и, если это "ИСТИНА", то выполняются тексты <часть-while>, <часть-begin>, снова слово "WHILE" и так далее. Когда слово "WHILE" снимет со стека "ЛОЖЬ", выполнение цикла закончится и начнет выполняться текст, следующий после "REPEAT".

Отметьте, что значение "ИСТИНА" здесь задает ПРОДОЛЖЕНИЕ вычислений, в отличие от циклов типа "UNTIL".

Можете разобрать два примера.

1. Наибольший общий делитель двух положительных чисел:

: НОД (a b --> НОД/a,b/ по Евклиду) 2DUP < IF SWAP THEN (теперь a>=b) BEGIN DUP (a b b) WHILE (пока b>0 2DUP MOD (a b aMODb) ROT (b aMODb a) DROP (a' b' - новые значения a и b) REPEAT DROP ; (НОД(a,b) /все/)

2. Подсчет числа единиц в двоичном разложении числа:

: UNITS (a --> число единиц в a) 0 SWAP (0 a) (В стеке лежат два числа: счетчик числа единиц s) (и постепенно изменяемое число a) BEGIN (s a') DUP (s a' a') LAST1 DUP (s a' d d) WHILE (пока d>0) - (s a'') SWAP 1+ SWAP (s' a'') REPEAT 2DROP ; (s)

Слово "LAST1" было определено в п.5.

Содержание

12.2. ЦИКЛЫ-СО-СЧЕТЧИКОМ

В циклах со счетчиком (перечислительных циклах) пользователь сам определяет число повторений цикла. В таких циклах имеется целочисленная переменная, пробегающая нужное множество значений - ПАРАМЕТР ЦИКЛА. Для организации перечислительных циклов используется конструкция:

```
                DO  <тело-цикла>  +LOOP
        или
```

```
                    DO  <тело-цикла>  LOOP
```

Слово "DO" берет из стека два значения

DO -- ... b a --> ...

где a - начальное значение параметра цикла, а b - пороговое. Слово "+LOOP" прибавляет в каждой итерации к параметру цикла число, которое берет из стека. Текст <тело-цикла> исполняется до тех пор, пока очередное значение параметра цикла не "перепрыгнет" через границу, проходящую между b-1 и b. Конструкция

```
                      DO  ...  +LOOP
```

допускает и отрицательные приращения параметра; именно с учетом этого и дано такое правило завершения цикла.

Используемое чаще слово "LOOP" эквивалентно "1 +LOOP". В конструкции с этим словом <тело-цикла> исполнится b-a раз при b>a. При b Слово "I" помещает в стек текущее значение параметра цикла. Значение параметра объемлющего цикла помещается в стек словом "J".

Пример. Сумма квадратов первых n натуральных чисел:

```
: SS2    (  n   -->  сумма  )
  0  SWAP 1+  1  DO  I  DUP  *  +  LOOP  ;
```

Существует способ завершить выполнение цикла ранее заданного числа повторений. Слово "LEAVE" обеспечивает немедленный выход из того цикла, в котором оно находится. В одном цикле можно употребить несколько слов "LEAVE". Цикл завершается при первом его исполнении.
Предупреждение: слово "LEAVE" должно явно находиться в том же определении, что и цикл, выход из которого оно задает.

Текущее и граничные значения параметра цикла в процессе работы хранятся в СТЕКЕ ВОЗВРАТОВ (п.15).

Содержание

13. АРИФМЕТИКА-ДВОЙНОЙ-ТОЧНОСТИ

Наряду с обычным представлением целых чисел, в котором на каждое число отводится по два байта, Форт допускает представление чисел с двойной точностью - число размещается в четырех байтах и, следовательно, может принимать значения от -2**31 до 2**31-1. Чтобы отличать числа двойной точности от обычных чисел, надо в запись таких чисел (в любом месте) включить точку.

Например: 12345678.

При размещении таких чисел в стеке и, аналогично, в памяти в четырех идущих подряд байтах старшая половина располагается правее младшей. Слов, работающих с числами двойной точности, в

языке не очень много; разумеется, к ним относятся и уже известные Вам слова "2DROP", "2DUP", "2OVER", "2SWAP".

Слово "D." выводит число двойной длины со знаком:

D. ... d --> ...

Слово "2@" аналогично слову "@", оно кладет на стек число двойной длины, находящееся по данному адресу:

2@ ... addr --> ... d

Слово "2!" аналогично слову "!":

2! ... d addr --> ...

Из арифметических операций можно назвать:

D+ ... d1 d2 --> ... d1+d2 D- ... d1 d2 --> ... d1-d2 DABS ... d --> ... |d| DNEGATE ... d --> ... -d D< ... d1 d2 --> ... d1 ... d1=d2

Операции умножения и деления являются "смешанными", то есть умножение двум числам обычной длины сопоставляет их произведение двойной длины; в делении делимое имеет двойную длину, а делитель, частное и остаток - обычную.

M* ... n1 n2 --> ... d=n1*n2 M/MOD ... d n --> ... n1 n2 (остаток частное)

Для перевода числа в двойное используется слово "S>D"

: S>D (n --> d) DUP 0< ;

Обратный перевод требует тщательной проверки его корректности, но в простейшем случае эквивалентен "DROP".

Наконец, имеются слова "2CONSTANT" и "2VARIABLE", вполне аналогичные своим прообразам для чисел обычной длины.

Содержание

14. ФОРМАТНЫЙ-ВЫВОД-ЧИСЕЛ

В языке Форт можно просто менять систему счисления, используемую при вводе и выводе информации. Переменная "BASE" хранит основание текущей системы счисления. Для установки текущей системы имеются слова:

"HEX" - шестнадцатиричная, "DECIMAL" - десятичная.

Иногда к ним добавляют:

"BINARY" - двоичная, "OCTAL" - восьмиричная.

Например, слово "HEX" определено так

```
: HEX  16 BASE ! ;
```

Установленная система счисления остается текущей до следующего изменения. При загрузке Форт-системы устанавливается десятичная система.

Описываемые ниже слова работают с буфером вывода, в котором формируется внешнее представление числа в виде строки символов. Основным преобразователем разрядов числа в символы является слово "##". Оно делит двойное число с вершины стека на основание текущей системы счисления, заменяет его на стеке получившимся частным (тоже двойной длины), а остаток переводит в литеру и записывает в буфер при помощи слова "HOLD". При этом указатель буфера продвигается на одну позицию. Форматное преобразование должно начинаться словом "<##", которое устанавливает указатель на конец буфера, так как формирование строки идет от конца. Слово "HOLD" можно использовать и для вставки во внешнее представление числа желаемых дополнительных символов. Слово "##>" завершает преобразование и помещает в стек адрес сформированной в буфере строки литер и ее длину.

Полный перевод числа сразу выполняет слово "##s", которое оставляет на стеке двойной нуль - результат последнего деления.

Для примера можно разобрать определение слова "D." :

: ##s (d --> 0 0) BEGIN ## 2DUP 0 0 D= UNTIL ; : SIGN (n -->) (вывод знака минус) 0< IF C" - HOLD THEN ; : D. (d -->) 2DUP DABS <## ##s ROT SIGN ##> TYPE SPACE DROP ;

Слово "TYPE" выводит символы, которые составляют число. Для того, чтобы вставить пробел между числом и приглашением ok, добавлено слово "SPACE".

Для примера создадим два слова форматного вывода.

Первое печатает номер телефона в стандартном виде:

: .PHONE (d -->) <## ## ## C" - HOLD ## ## C" - HOLD ##s ##> TYPE ;

То есть при вводе

```
2333410.  .PHONE
```

получим 233-34-10

При помощи второго слова ".TABLEAU" можно выводить результаты марафонского забега, замеренного с точностью до сотых долей секунды; например, при вводе

```
          946293.  .TABLEAU
```

получим 2ч37м42.93с

Введем два вспомогательных слова. Слово "SIXI" устанавливает шестиричную систему счисления. Слово "##ms" выдает минуты или секунды

: SIXI (-->) 6 BASE ! ; : ##ms (d --> d/60) ## SIXI ## DECIMAL ;

Слово ".TABLEAU" собственно и выводит результаты забега

: .TABLEAU (d -->) <## C" с HOLD ## ## C" . HOLD ##ms C" м HOLD ##ms C" ч HOLD ##s ##> TYPE ;

Содержание

15. СТЕК-ВОЗВРАТОВ

Кроме арифметического стека в системе используется еще один стек, называемый СТЕКОМ ВОЗВРАТОВ. В основном в нем хранятся указатели, используемые Форт-системой при обработке вложенных структур.

Стек возвратов тоже организован по принципу LIFO. Пользователь может временно хранить в нем свою информацию, но с учетом следующего. Данные, внесенные в стек возвратов, надо выбрать из него прежде, чем закончится соответствующее определение. Опасно передавать на этом стеке параметры от одного слова к другому.

Работать со стеком возвратов можно с помощью слов:

> R ... a --> ... | ... --> ... a

число a снимается со стека и кладется в стек возвратов (справа от черты),

R> ... --> ... a | ... a --> ...

число a снимается со стека возвратов и кладется в арифметический стек,

R@ ... --> ... a | ... a --> ... a

число a с вершины стека возвратов копируется в арифметический стек

Пример. Описание слова "3DUP"

: 3DUP (a b c --> a b c a b c) >R 2DUP R@ -ROT R> ;

Еще одно предупреждение. При использовании операций со стеком возвратов внутри перечислительного цикла слова "I" и "J" могут выдавать неправильные значения, если эти операции не сбалансированы. Это происходит потому, что на стеке возвратов хранятся текущие и граничные значения параметров цикла. Слово "I" просто снимает нужное значение со стека возвратов в соответствии с выбранным в реализации форматом.

Содержание

16. КОСВЕННОЕ-ИСПОЛНЕНИЕ

Итак, введенное слово ищется в контекстном словаре в обратном направлении, начиная с самого последнего введенного слова. Если слово найдено, оно исполняется. Для реализации каждого из этих двух действий по отдельности служат следующие слова:

' имя ... --> ... addr

Слово "имя" должно быть уже определено. Слово "'" (апостроф) кладет на стек адрес поля кода слова "имя".

EXECUTE ... addr --> ...

Адрес поля кода некоторого слова снимается со стека, и это слово исполняется. Таким образом, текст

```
' имя EXECUTE
```

просто эквивалентен тексту "имя".

Предлагаемое средство - это способ выполнять слова не непосредственно, вводя их имена, а косвенно. Заводится какая-нибудь переменная, например "УКАЗАТЕЛЬ", в которой с помощью апострофа можно запомнить адрес поля кода некоторого слова и в дальнейшем даже менять ее содержимое; а исполнение слова задается текстом:

```
УКАЗАТЕЛЬ @ EXECUTE
```

Подменяя адреса поля кода, можно изменить выполняемые действия некоторого слова уже после того, как оно скомпилировано. Обычно на этом принципе основаны определения слов для интерфейса с внешними устройствами. Только учтите, что слово "EXECUTE" не проверяет, допустим ли заданный на стеке адрес. А неверный адрес почти всегда влечет нарушение работы системы.

Если надо указать внутри определения через ":", что действие апострофа должно относиться к следующему слову в теле определения, то используют слово "[']". Иначе во время исполнения того определенного через двоеточие слова апостроф 'займется' очередным словом из входного потока.

Содержание

17. СЛОВА-ГЕНЕРАТОРЫ

В отличие от других языков Форт позволяет программисту создавать собственные определяющие слова. Это дает возможность создавать семейства слов с похожими свойствами. Одинаковые признаки задаются не в каждом члене, а в самом определяющем слове. Благодаря этому сокращается текст программ, они легче читаются и модифицируются. При умелом использовании таких конструкций эффект может быть очень значительным. Определяющие слова будем также называть словами-генераторами или мета-определяющими словами. С помощью собственных генераторов можно образовывать новые структуры данных, например, многомерные массивы.

Слова-генераторы описываются с помощью слов "CREATE" и "DOES>" в конструкции

: имя-генератора CREATE <часть-create> DOES> <часть-does> ;

и употребляются в конструкции

```
имя-генератора        имя                    (*)
```

для определения слова "имя".

Слово "CREATE" в определении генератора указывает начало действий, выполняемых в период компиляции будущего определяемого слова; слово "DOES>" - начало действий во время исполнения этого нового слова.

При выполнении конструкции (*) слово "CREATE" создает в кодофайле словарную статью для слова "имя", но при этом память под поле параметров не выделяется. После этого исполняется текст <часть-create>, который может создать поле параметров определяемого слова. В дальнейшем при исполнении слова "имя" на стек кладется адрес его поля параметров и выполняется текст <часть-does>.

Например, генератор для определения констант выглядит так:

: CONSTANT CREATE , (слово "," резервирует 2 байта) (и кладет в них число из стека) DOES> (на стеке адрес этих двух байтов) @ ; (значение помещается в стек)

В качестве упражнения разберите устройство генератора одномерных массивов. Слово "ARRAY" берет из стека целое число n и резервирует в кодофайле место для n чисел, связывая с ними следующее за "ARRAY" слово как имя этого массива. В дальнейшем при выполнении имени массива из стека берется индекс, проверяется его принадлежность диапазону от 1 до n и, если все в порядке, в стек помещается адрес соответствующего элемента массива.

: ARRAY (в стеке лежит число элементов) CREATE DUP , (это число помещается в поле параметров) 2* ALLOT (захват места для массива) DOES> (при вызове в стеке лежит индекс) (и помещается адрес захваченной памяти) OVER 1 < IF ." ИНДЕКС МЕНЬШЕ 1" 2DROP ELSE 2DUP @ > IF ." ИНДЕКС БОЛЬШЕ ЧЕМ НАДО" 2DROP ELSE 1+ SWAP 2* + THEN THEN ;

Содержание

18. УПРАВЛЕНИЕ-РЕЖИМАМИ

В каждый момент Форт-система может находиться в одном из двух состояний - ИСПОЛНЕНИЯ или КОМПИЛЯЦИИ. При загрузке системы устанавливается режим исполнения. Появление во входном тексте определяющего слова ":" переводит систему в режим компиляции на период обработки определения. Слово ";" завершает компиляцию и возвращает систему в прежний режим. То есть, само это слово исполняется при режиме компиляции. Дело в том, что слово ";" является словом НЕМЕДЛЕННОГО ИСПОЛНЕНИЯ. Признак немедленного исполнения - это специальный бит в поле имени словарной статьи каждого слова (либо он выставлен, либо нет). Словами немедленного исполнения являются и все управляющие конструкции.

Предусмотренное в системе слово "IMMEDIATE" присваивает признак немедленного исполнения последнему определенному к моменту его появления слову. Таким образом, программист может создавать новые управляющие слова, которые будут исполняться в период компиляции.

С другой стороны, слово "[COMPILE]" заказывает принудительную компиляцию следующего за ним слова независимо от наличия у того признака немедленного исполнения. Само оно также имеет

признак немедленного исполнения.

Можно поступать и иначе. Два слова "[" и "]" немедленно переключают режимы даже внутри определения: "[" - перевод в режим исполнения; "]" - перевод в режим компиляции.

В стандарте имеется удобное слово "FIND", служащее для поиска слова в словаре с проверкой признака немедленного исполнения.

FIND ... addr1 --> ... addr2 n

Здесь addr1 - адрес строки со счетчиком, содержащей имя слова. Число n принимает значение 0, если слово не найдено; 1, если слово найдено и имеет признак немедленного исполнения; -1, если этого признака нет. Значение addr2 в первом случае остается прежним, в двух других является адресом поля кода слова.

В отличие от "'" слово "FIND" использует строку со счетчиком, это позволяет формировать образец поиска программным путем.

Специальная переменная "STATE" (флаг состояния) имеет значение ИСТИНА для режима компиляции. Можно написать слова, которые будут выполнять разные действия в зависимости от значения этого флага. Их называют зависимыми от состояния и на первых порах использовать не рекомендуют.

В выполняемые словом действия может входить компиляция других слов. Само слово, включающее такое действие, имеет обычно признак немедленного исполнения. В общем виде соответствующее определение можно представить так:

: имя1 ... COMPILE имя2 ... ; IMMEDIATE

При компиляции слова "имя1" слово "COMPILE" запоминает адрес следующего в определении слова "имя2". В дальнейшем при исполнении слова "имя1" внутри определения некоторого другого слова "имя3" этот запомненный адрес записывается в поле параметров создаваемой словарной статьи. То есть слово "имя2" будет исполняться во время исполнения слова "имя3", а не "имя1".

Для примера посмотрите, как можно реализовать слова "IF"и "THEN".

: IF COMPILE ?BRANCH HERE 0 , ; IMMEDIATE : THEN HERE SWAP ! ; IMMEDIATE

При своем будущем исполнении слово "?BRANCH" снимает со стека число и, если оно - 0, подменяет адреса передачи управления, обеспечивая условный переход. Обратите внимание, что число на стеке будет проверяться не в момент исполнения "IF" при компиляции определения некоторого слова, а в момент исполнения самого этого слова. Таким образом, исполнение слова "IF" компилирует адрес слова "?BRANCH", резервирует место для ссылки вперед (на обход ветви-IF) и оставляет адрес зарезервированного места на стеке. Слово "THEN" вписывает по этому адресу текущий адрес в кодофайле, сбрасывая сам адрес со стека.

Учтите, что стек активно используется в процессе компиляции слов системой; поэтому изменять его во время обработки определений (например, текстом "[DROP]") не рекомендуется!

Этот пример демонстрирует, как пользователь может создавать собственные структуры управления, повышая эффективность программ.

Содержание

19. АДРЕСНЫЙ-ИНТЕРПРЕТАТОР

Сведения данного параграфа предназначены для более углубленного понимания работы Форт-системы. Они относятся к исполнению слов, определенных через ":", в словарной статье которых поле параметров содержит адреса полей кодов слов, составлявших соответствующее определение.

В Форт-системе имеется специальная программа - АДРЕСНЫЙ ИНТЕРПРЕТАТОР, которая занимается исполнением слов, не записанных в машинных командах. Интерпретатор представляет собой программу с тремя режимами. Так как в определениях через ":" налицо вложенность слов, можно говорить об уровнях вложенности и, соответственно, об уровнях интерпретации.

Исполнение некоторого слова начинается вызовом режима "CALL", который устанавливает указатель интерпретации "IP" на начало программной части этого слова. Прежнее значение "IP", которое указывало на следующее слово предыдущего уровня интерпретации, запоминается в стеке возвратов. Режим "CALL" завершается первым вызовом режима "NEXT".

Режим "NEXT" передает управление по адресу, записанному в поле параметров, на которое указывает "IP", одновременно передвигая "IP" на следующий элемент поля (т.е. прибавляя к нему 2).

Такие вложенные вызовы продолжаются до тех пор, пока управление не передано на машинную подпрограмму, которая и будет исполнена. В конце каждой машинной программы записан вызов режима "NEXT" адресного интерпретатора.

В конце поля параметров каждой словарной статьи записан вызов третьего режима интерпретатора - режима "RETURN" (его компилирует туда слово ";"). Режим "RETURN" обеспечивает выход на предыдущий уровень: загружает "IP" значением, снимаемым со стека возвратов, и вызывает режим "NEXT".

После возврата на самый верхний уровень интерпретации специальное системное слово восстанавливает диалог.

В системе имеется слово "EXIT", которое сразу вызывает режим "RETURN". То есть "EXIT" можно использовать внутри определения некоторого слова, чтобы задать немедленное прекращение его исполнения (это удобно в условных операторах). Однако внутри перечислительного цикла использовать "EXIT" нельзя !

Содержание

20. ПЕРЕМЕННЫЕ-ТИПА-"QUAN"-И-"VECT"

Переменная типа "QUAN" отличается от стандартной переменной типа "VARIABLE" тем, что объединяет свойства константы и переменной. При исполнении переменная типа "VARIABLE" оставляет на стеке адрес ячейки, в которой хранится ее значение. Но этот адрес обычно используется либо для получения самого значения с помощью слова "@", либо для засылки в переменную нового значения с помощью слова "!". Так вот слово "QUAN" создает переменную, исполнение имени которой в зависимости от контекста имеет один из трех результатов: получение текущего значения, засылка нового значения и получение адреса значения.

Переменная типа "QUAN" определяется следующим образом:

```
                    QUAN  имя
```

Слово "IS", употребленное перед именем переменной, засылает в переменную новое значение, взятое с вершины стека.

Слово "AT", употребленное перед именем переменной, записывает в стек адрес ее значения.

Слова "IS" и "AT" назовем префиксами к имени переменной.

При использовании имени переменной без префиксов на стек будет помещено ее текущее значение (свойство константы). Заметим, что использование "чистого" имени эквивалентно: AT имя @ а засылка при помощи "IS" эквивалентна выражению: AT имя !

Слово для реализации переменной типа "QUAN" имеет три поля кода:

```
поле кода для получения значения (2/3 байта)
поле кода для засылки значения (2/3 байта)
поле кода для получения адреса значения (2/3 байта)
значение переменной (2 байта)
```

Длина поля кода зависит от типа шитого кода, но в любом случае требуются два дополнительных поля кода. Эти затраты памяти на описание переменной компенсируются при ее использовании, так как каждое обращение к переменной занимает два байта и к тому же работает быстрее, чем обращение к переменной типа "VARIABLE". Это объясняется тем, что на месте использования переменной типа "QUAN" компилируется адрес одного из трех полей ее кода в зависимости от указанного префиксом (или его отсутствием) действия:

действие	**для QUAN**	**для VARIABLE**
значение	имя (2 байта)	имя @ (4 байта)
присваивание	IS имя (2 байта)	имя ! (4 байта)
адрес	AT имя (2 байта)	имя (2 байта)

По аналогии со словом "QUAN" используется слово "VECT", создающее слово с векторизованным исполнением. Его можно рассматривать как переменную типа "QUAN", значением которой являются другие слова. Например, после выполнения текста

```
          VECT имя
          ' DUP  IS имя
```

последующее исполнение слова "имя" равносильно исполнению "DUP". Таким образом, созданные при помощи "VECT" слова можно рассматривать как слова со сменной семантикой. Отличие реализаций "VECT" и "QUAN" состоит лишь в следующем: исполнение имени переменной типа

"VECT" вызывает выполнение слова, адрес поля кода которого содержит переменная. Префиксы используются аналогичным образом.

Содержание

21. ВНЕШНЯЯ-ПАМЯТЬ

Памяти микрокомпьютера обычно недостаточно для запоминания всех необходимых данных и программ. Для этого используется память на внешних носителях. В основном, это память на магнитном диске.

Дисковая память Форт-системы разделена на БЛОКИ. Считается, что каждый блок с исходным текстом хранит 1024 символа и состоит из 16 строк по 64 символа в каждой. Иногда такой блок называют экраном. Для ввода информации в блок и ее изменения при Форт-системе имеется собственный редактор.

Блоки собраны на диске в один или несколько файлов по желанию программиста. Слово "USING" устанавливает текущий файл внешней памяти; используется в виде

```
USING  имя-файла
```

Все описываемые ниже слова работают с блоками текущего файла. Блоки пронумерованы, начиная с 0. Количество блоков в файле системой не ограничивается.

Слово "LIST" распечатывает на экране содержимое блока, номер которого находится на вершине стека, в 16 строк по 64 символа. При этом слева добавляются числа - нумерация строк с 0 по 15. Переменная "SCR" хранит номер последнего распечатанного по "LIST" блока. Например, можно написать такие слова:

: LL SCR @ 1- LIST ; (распечатать предыдущий блок) : LN SCR @ 1+ LIST ; (распечатать следующий блок)

Рекомендуется в нулевой строке блока писать комментарий к его содержимому. Слово "INDEX" служит для распечатывания нулевых строк последовательности блоков. Например, выражение

```
12  24  INDEX
```

распечатает начальные строки блоков с 12 по 24 включительно.

Текст Форт-программы может вводиться как с клавиатуры, так и с диска. ЗАГРУЗКА блока (обработка его содержимого Форт-системой) задается словом "LOAD". Номер блока берется с вершины стека. Заметим, что текст загружаемого блока в свою очередь может содержать слово "LOAD". После загрузки "внутреннего" блока произойдет возврат и "дозагрузка" остатка текущего блока.

Программа обычно занимает больше одного блока. Слово "THRU" загружает последовательность блоков в диапазоне номеров, взятых со стека. Например, выражение:

```
        12          24   THRU
```

загрузит блоки с 12 по 24 включительно.

Слово "-->" вызывает загрузку следующего по номеру блока. Слово ";S" прекращает загрузку блока, что позволяет загружать только его часть.

Переменная "BLK" содержит номер блока, который загружается в данный момент. Если "BLK" содержит ноль, это означает, что ввод текста идет с клавиатуры. Таким образом, БЛОК С НОМЕРОМ 0 НЕ МОЖЕТ СОДЕРЖАТЬ ПРОГРАММУ. Еще при вводе текста используется переменная ">IN". Она содержит смещение в байтах относительно начала блока, откуда в настоящий момент идет ввод текста. Если в "BLK" ноль, то ">IN" указывает на смещение от начала буфера ввода.

Любая работа с содержимым блока выполняется не на диске, а в оперативной памяти. Слово "BLOCK" переписывает блок с указанным на стеке номером в 1024-байтный БЛОЧНЫЙ БУФЕР и оставляет на стеке адрес начала этого буфера. (Распечатка и загрузка блоков также используют это слово.) Теперь для работы с блоком можно применять любые слова, работающие с оперативной памятью. Обычно Форт-система предоставляет пользователю право задавать количество блочных буферов для одновременного хранения нескольких блоков. Это позволяет многократно модифицировать их содержимое, не обращаясь каждый раз к диску, что существенно экономит время. Слово "BLOCK" сначала проверяет, нет ли уже нужного блока в некотором буфере. Если надо перекачивать блок, а все буферы заполнены, то обычно заменяется блок с самым давним доступом. Если его содержимое подвергалось изменениям, оно предварительно копируется обратно на диск (без ведома пользователя).

С каждым буфером связан флаг наличия изменений. Слово "UPDATE" помечает как измененный блок, к которому осуществлялся самый последний доступ.

Можно заставить систему записать измененный блок на диск, не дожидаясь, пока его вытеснит другой блок. Для этого используются слова "FLUSH" и "SAVE-BUFFERS". "SAVE-BUFFERS" копирует все помеченные по "UPDATE" блоки на диск, не освобождая буферы; "FLUSH" после перекачки еще и очищает буферы, заполняя их пробелами.

Слово "EMPTY-BUFFERS" сбрасывает все флаги изменений без записи на диск. Это слово можно применить, если Вы случайно испортили содержимое какого-то блока и не хотите, чтобы изменения попали на диск. Надо только помнить, что ПРИ ЭТОМ ОЧИЩАЮТСЯ ВСЕ БУФЕРЫ. Далее придется восстанавливать их словом "BLOCK".

На основе описанных средств можно строить собственные файловые системы или организовывать базы данных.

Содержание

22. АССЕМБЛЕР

Форт является одним из самых быстрых и эффективных языков программирования и широко используется в системах реального времени и специализированных приложениях. Такие программы обычно пишутся на "высокоуровневом" Форте. Однако можно значительно ускорить их выполнение,

переписав интенсивно используемые слова в машинных кодах. Для этой цели Форт-система имеет встроенный Форт-ассемблер, который вдобавок позволяет непосредственно обращаться к аппаратуре и операционной системе. Рекомендуем именно переписать критичные по времени участки программы после того, как она будет отлажена.

Новое определение создается по форме

CODE <имя-слова> <ассемблерная программа> END-CODE

Ассемблерная программа представляет собой запись операторов машинного кода в обратной польской записи. Определенное таким образом слово вызывается и выполняется подобно Форт-слову. Оно может работать со значениями из стека, что позволяет передавать аргументы так же, как в Форт-словах. Основное отличие состоит в том, что слово "CODE" устанавливает контекст словаря ассемблерных мнемоник "ASSEMBLER". В этом же словаре имеются ассемблерные версии структур управления (условных операторов и циклов).

Любое ассемблерное определение должно завершаться вызовом адресного интерпретатора. То есть ассемблерная программа должна заканчиваться словами "NEXT JMP".

Слово "END-CODE" восстанавливает контекст словаря "FORTH".

Рассмотрим реализацию операции "SWAP" на Форт-ассемблере микропроцессора К1810:

CODE SWAP AX POP BX POP AX PUSH BX PUSH NEXT JMP END-CODE

Преимуществом Форт-ассемблера является его расширяемость и "встроенность в Форт". Внутри ассемблерного определения можно воспользоваться определением через ":" как макрокомандой; можно обратиться к переменной.

Примеры использования машинных слов дает сама Форт-система. На Форт-ассемблере написаны слова для обмена с терминалом и диском (обычно через обращения к операционной системе) и основные слова ядра.

Содержание

ЛИТЕРАТУРА


```
Л.Броуди. Начальный курс программирования на языке ФОРТ. "Финансы и
статистика", М, 1990.
А.Ю.Бураго, В.А.Кириллин, И.В.Романовский. Форт - язык для
микропроцессоров. Ленинградская организация общества "Знание" РСФСР, Л,
1989.
С.Н.Баранов, Н.Р.Ноздрунов. Язык Форт и его реализация. "Машиностроение",
Л, 1988.
```