Американский Национальный Стандарт
для Информационных Систем --
Языки Программирования ---

Forth

Секретариат
Компьютерная и Деловая Ассоциация Изготовителей Оборудования
Одобренный: 24 марта 1994
American National Standards Institute, Inc.

СОДЕРЖАНИЕ

Предисловие

(Это предисловие - не часть Американского Национального Стандарта X3.215-1994) Forth - язык для непосредственной связи между людьми и машинами. Используя стиль естественного языка и машинно-ориентированный синтаксис, Forth обеспечивает экономичную, производительную среду для интерактивной трансляции и выполнения программ. Forth также обеспечивает низкоуровневый доступ к управляемым компьютером аппаратным средствам, и возможность расширять язык непосредственно. Эта расширяемость позволяет быстро расширять язык и приспосабливать его к специфичным потребностям и различным аппаратным системам.

Forth был изобретен г. Чарльзом Муром, чтобы увеличить производительность программиста без принесения в жертву машинной эффективности. Forth - многоуровневая среда, содержащая элементы машинного языка, а также операционную систему и машинный монитор. Эта расширяемая, многоуровневая среда предусмотрена для высоко интерактивной разработки и отладки программ.

В интересах переносимости прикладного программного обеспечения, написанного в Forth, работы по стандартизации начались в середине 1970-ых международной группой пользователей и разработчиков, принявших название "Forth Standards Team". Эти работы завершились стандартом Forth-77. Поскольку язык продолжил развиваться, "Forth Standards Team" был издан промежуточный стандарт Forth-78. Следующая встреча "Forth Standards Team" была в 1979, а Forth-79 стандарт был издан в 1980. Главные изменения были сделаны "Forth Standards Team" в Forth-83 стандарте, который был издан в 1983.

Первая встреча Technical Committee по системам программирования на Forth была созвана Оргкомитетом X3J14 Forth Technical Committee 3 августа 1987, с последующими встречами 11-12 ноября 1987, 10-12 февраля 1988, 25-28 мая 1988, 10-13 августа 1988, 26-29 октября 1988, 25-28 января 1989, 3-6 мая, 1989, 26-29 июля 1989, 25-28 октября 1989, 24-27 января 1990, 22-26 мая 1990, 21-25 августа 1990, 6-10 ноября 1990, с 29 января по 2 февраля 1991, 3-4 мая 1991, 16-19 июня 1991, с 30 июля по 3 августа 1991, 17-21 марта 1992, 13-17 октября 1992, 26-30 января 1993, 28-30 июня 1993, и 21 июня 1994.

Запросы об интерпретации, предложения по исправлениям и дополнениям, или отчеты об ошибках принимаются. Они должны быть посланы в X3 Секретариат, Computer and Business Equipment Manufacturers Association, 1250 Eye Street, NW, Suite 200, Washington, DC 20005.

X3 Состав

Этот стандарт был разработан и одобрен для представления в ANSI при Accredited Standards Committee on Information Processing Systems,X3. Одобрение комитетом этого стандарта не подразумевает обязательно, что все члены комитета голосовали за его одобрение. Во время одобрения этого стандарта, X3 Комитет имел следующий состав:

James D. Converse, Chair
Donald C. Loughry, Vice-Chair
Joanne Flanagan, Secretary

Producer Group	Name of Representative

	AMP Incorporated	Edward Kelly
		Charles Brill (Alt.)
	AT&T/NCR Corporation	Thomas W. Kern
		Thomas F. Frost (Alt.)
	Apple Computer, Inc	Karen Higginbottom
	Compaq Computers	James Barnes
	Digital Equipment Corporation	Delbert Shoemaker
		Kevin Lewis
	Hitachi America Ltd	John Neumann
		Kei Yamashita (Alt.)
	Hewlett Packard	Donald C. Loughry
	Bull HN Information Systems Inc	William George
	IBM Corporation	Joel Urman
		Mary Anne Lawler (Alt.)
	Unisys Corporation	John Hill
		Stephen P. Oksala (Alt.)
	Sony Corporation of America	Michael Deese
	Storage Technology Corporation	Joseph S. Zajaczkowski
		Samuel D. Cheatham (Alt.)
	Sun Microsystems, Inc	Scott Jameson
		Gary S. Robinson (Alt.)
*	Xerox Corporation	Dwight McBain
		Roy Pierce (Alt.)
	3M Company	Edie T. Morioka
		Paul D. Jahnke (Alt.
	Consumers Group	
	Boeing Company	Catherine Howells
		Andrea Vanosdoll (Alt.)
	Eastman Kodak Company	James Converse
		Michael Nier (Alt.)
	General Services Administration	Douglas Arai
		Larry L. Jackson (Alt.)
	Guide International Inc	Frank Kirshenbaum
		Harold Kuneke (Alt.)
**	Hughes Aircraft Company	Harold Zebrack
	National Communications Systems	Dennis Bodson

	Northern Telecom Inc	Mel Woinsky
		Subhash Patel (Alt.)
**	Recognition Tech Users Association	Herbert P. Schantz
		G. Edwin Hale (Alt.)
	Share Inc	Gary Ainsworth
		David Thewis (Alt.)
	U. S. Department of Defense	William Rinehuls
		C. J. Pasquariello (Alt.)
	U. S. Department of Energy	Alton Cox
		Lawrence A. Wasson (Alt.)
	Wintergreen Information Services	John Wheeler
	General Interest Group	
	American Nuclear Society	Geraldine C. Main
		Sally Hartzell (Alt.)
	Assn. of the Institute for Certification	
	of Computer Professionals	Kenneth Zemrowski
	Nat'l Institute of Standards and Technology	Robert E. Rountree
		Micharl Hogan (Alt.)
	Neville & Associates	Carlton Neville

---
* Воздержался ** Нет данных

X3J14 Состав

Во время одобрения этого проекта предложенного American National Standard, Technical Committee X3J14 по зыку программирования Forth имел следующий состав:

Elizabeth Rather, Chair
Mitch Bradley, acting Vice-Chair
Don Colburn, Secretary
John Rible, Technical Editor
Len Zettel, Vocabulary Representative
Greg Bailey, Technical Subcommittee Chair

Organization Represented	Name of Representative
ATHENA Programming, Inc	Greg Bailey
	Howe Fong (Alt.)
Bradley Forthware	Mitch Bradley
Creative Solutions, Inc	Don Colburn
Ford Motor Company	Leonard F. Zettel, Jr.
FORTH, Inc	Elizabeth Rather
	Dennis Ruffer (Alt.)
Institute for Applied Forth Research	Lawrence Forsley
	Horace Simmons (Alt.)
Johns Hopkins University, Applied Physics Lab.	John Hayes
Mephistopheles Systems	Dave Harralson
NASA/Goddard Space Flight Center	James Rash
Nomadic Software	John K. Stevenson
Unisyn, Inc	Gary Betts
	Stephen Egbert (Alt.)
Up and Running	Martin Tracy
Vesta Technology	Jack Woehr

Individual Members
Loring Craymer
John Rible
J. E. (Jet) Thomas

X3 Liasons
Clyde R. Camp
Kathleen McMillan

Следующие организации и частные лица также участвовали в этом проекте как члены Technical Committee, заместители, или наблюдатели. Technical Committee ценит и уважает их вклад:

Organizations

British Columbia Inst. of Tech.	MCI Telecommunications Corp.
Computer Cowboys	Micromotion
Computer Sciences Corp.	MicroProcessor Engineering Ltd.
Computer Strategies, Inc.	National Institute of Standards & Technology

Digalog Corp.	NCR Medical Systems Group
Embedded Sys. Programming Mag.	Performance Packages, Inc.
Forth Interest Group (FIG)	Purdue University

H.B. Pascal & Co., Inc. Robert Berkey Services

Harris Semiconductor Shaw Laboratories
IBM Corporation Social Security Administration
IEEE Software Engineering
Kelly Enterprises Texas Instruments
Laboratory Microsystems, Inc. The Dickens Company
Maxtor Corp.

Individuals

David J. Angel	Ray Duncan	Charles Moore	Dean Sanderson
Wil Baden	Douglas Fishman	Mike Nemeth	George Shaw
Robert Berkey	Tom Hand	Harry Pascal	Gerald Shifrin
Ron Braithwaite	Gregory Ilg	Stephen Pelc	Robert Smith
Jack Brown	Charles Keane	Dean Perrine	Tyler Sperry
Chris Colburn	Guy M. Kelly	David C. Petty	Tom Zimmer
Ted Dickens	Andrew Kobziar	Bill Ragsdale	
John Dorband	Martin Lascelles	James Ryland	

AMERICAN NATIONAL STANDARD
ANSI X3.215-1994

Американский Национальный стандарт
для Информационных Систем --
Языки Программирования ---

Forth
Forth
Forth

1. Введение

1.1 Цель

Цель этого Стандарта состоит в том, чтобы способствовать переносимости Forth программ для использования на большом разнообразии вычислительных систем, чтобы облегчить обмен программами, методами программирования, и идеями среди программистов Forth, и служить как основание будущего развития языка Forth.

1.2 Назначение

Этот Стандарт устанавливает интерфейс между Forth системой и Forth программой определением слов, предусмотренных Стандартной Системой.

1.2.1 Включения

Этот Стандарт определяет:
- формы, которые может иметь программа, написанная на языке Forth;
- правила для интерпретации смысла программы и ее данных.

1.2.2 Исключения

Этот Стандарт не определяет:

- механизм, которым программы преобразованы для использования на вычислительных системах;
- операции, требуемые для установки и контроля за выполнением программ на вычислительных системах;
- метод переноса программ или их входных/выходных данных на носитель или из носителя данных;
- поведение программы и Forth системы, в случаях не установленных правилами этого Стандарта;
- размер или сложность программы и ее данных, которые способны превысить вместимость какой либо вычислительной системы или возможности конкретной Forth системы;

- физические свойства записей ввода/вывода, файлов, и модулей;
- физические свойства и реализация памяти.

1.3 Организация документа

1.3.1 Наборы слов

Этот Стандарт группирует Forth слова и свойства в наборы слов под именами указывающими на некоторые общие свойства, типично их общая область функционирования. Каждый набор слов может иметь расширение, содержащее слова, которые предлагают дополнительные функциональные возможности. Эти слова - не требуются в реализации набора слов.

"Основной" набор слов, определенный в разделах с 1 по 6, содержит требуемые слова и свойства стандартной Системы. Другие наборы слов, определенные в разделах с 7 по 17, являются дополнительными, делая возможным расширение стандартной Системы дополнительными уровнями функциональности.

1.3.1.1 Текстовые разделы

В пределах каждого набора слов, раздел 1 содержит вводный и объяснительный материал, а раздел 2 представляет термины и нотацию, используемые в Стандарте. Нет никаких требований в этих разделах.

Разделы 3 и 4 содержат требования использования и документации, соответственно, для стандартных Систем и программ, в то время как раздел 5 определяет их наименование.

1.3.1.2 Разделы словаря

Раздел 6 каждого набора слов определяет требуемое поведение определений в наборе слов и наборе слов расширений.

1.3.2 Дополнения

Дополнения не содержат никакого требуемого материала.

Приложение A предоставляет некоторые разъяснения вслед за решениями комитета в создании этого Стандарта, а также примеры реализации. Та же самая нумерация раздела, что и в теле Стандарта, облегчает связь каждого раздела требований с его разделом объяснений.

Приложение B - короткая библиография Forth.

Приложение C предоставляет введение в Forth.

Приложение D обсуждает совместимость ANS Forth с более ранними Forth, подчеркивая различия с Forth 83.

Приложение E представляет некоторые методы для создания переносимых программ в ANS Forth.

Приложение F включает слова из всех наборов слов в одном списке, и служит как индекс ANS Forth слов.

1.4 Будущие направления

1.4.1 Новая технология

Этот Стандарт принимает некоторые слова и практики, которые все более и более находятся в общей практике. Также были приняты новые слова для облегчения создания переносимых программ.

1.4.2 Устаревшие функции

Этот Стандарт принимает некоторые слова и практики, которые делают некоторых ранее используемые слова устаревшими. Хотя они сохранены здесь из-за их широкого распространения, их использование в новых реализациях или новых программах не рекомендуется, потому что они могут быть изъяты из будущих редакций Стандарта.

Этот Стандарт определяет следующие слова как устаревшие:

6.2.0060	#TIB	15.6.2.1580	FORGET	6.2.2240	SPAN
6.2.0970	CONVERT	6.2.2040	QUERY	6.2.2290	TIB
6.2.1390	EXPECT				

2. Термины, нотация, и ссылки

Фраза "Смотри:" используется в этом Стандарте для направления читателя к другим разделам Стандарта, которые имеют прямое отношение к текущему разделу.

В этом Стандарте, "следует" устанавливает требование к системе или программе; наоборот, "не следует" является запрещением; "не требуется" означает "не требуется для"; "желательно" описывает рекомендации Стандарта; и "возможно", в зависимости от контекста, означает "разрешено" или "может случаться".

Повсюду в Стандарте, шрифты используются следующим способом:
- этот пропорциональный Serif шрифт используется для текста, с курсивом, используется для символов и первого появления новых терминов;
- полужирный пропорциональный sans-serif шрифт используется для заголовков;
- полужирный Serif шрифт фиксированной ширины используется для текста языка-Forth.

2.1 Определения терминов

Термины, определенные в этом разделе используются вообще повсюду в этом

Стандарте. Дополнительные термины специфичные для определенных наборов слов, определены в этих наборах слов. Другие термины, определенные при их первом появлении, обозначены курсивом. Термины, не определенные в этом Стандарте, рассматриваются согласно Словарю для Информационных Систем, ANSI X3.172-1990.

адресуемый элемент: В зависимости от контекста:
1) элементы, на которые разделено адресное пространство Forth с целью размещения объектов данных типа символов и переменных;
2) физические запоминающие элементы памяти, соответствующие этим элементам;
3) содержание такого запоминающего элемента памяти;
4) или элементы, в которых непосредственно указана длина области памяти.

выровненный адрес: Адрес местоположения в памяти, к которому можно обращаться как к символу, ячейке, паре ячеек, или целым числам две-ячейки.

неопределенная ситуация: Обстоятельство, для которого этот Стандарт не предписывает определенного поведения для Forth систем и программ.

Неопределенные ситуации включают такие вещи как отсутствие необходимого разделителя во время синтаксического анализа, попытку доступа к несуществующему файлу, или попытку использования несуществующего слова.
Неопределенная ситуация также существует, когда Стандартное слово получает значения, которые являются неподходящими или выходящими из диапазона.

ячейка: Первичный элемент информации в архитектуре Forth системы.

пара ячеек: Две ячейки, которые рассматриваются как один элемент.

символ: В зависимости от контекста:
1) элемент памяти, способный хранить символ; или
2) элемент набора символов.

символьно-выровненный адрес: Адрес местоположения в памяти, к которому можно обратиться как к символу.

символьная строка: Область данных, которая связана с непрерывной последовательностью символьно-выровненных адресов. Символьные строки обычно содержат текст. Если иначе не обозначено, термин "строка" обозначает "символьную строку".

область кода: Логическая область словаря, в котором реализована семантика слова.

компилировать: Преобразовать исходный текст в определения словаря.

семантика компиляции: Поведение Forth определения, когда его имя встречает текстовый интерпретатор в состоянии компиляции.

строка со счетчиком: Структура данных, состоящая из одного символа, содержащего длину следующей за ним нулевой или более непрерывной последовательности символьных данных. Обычно, строки со счетчиком содержат текст.

кросс компилятор: Система, которая компилирует программу для последующего выполнения в среде, которая возможно, будет физически и логически отличной от среды компилирования. В кросс компиляторе, термин "host" применяется к среде компиляции, а термин "target" применяется к среде времени-выполнения.

текущее определение: Определение, чья компиляция была начата, но еще не закончена.

поле данных: Область данных связанная со словом, определенным через, CREATE.

область данных: Логическая область словаря, к которой можно обращаться.

указатель области данных: Адрес местоположения следующей доступной области данных, то есть, значение, возвращенное HERE.

стек данных: Стек, который можно использовать для передачи параметров между определениями. Когда не возможна путаница, стек данных упомянут как "стек". В отличие от стека возвратов.

тип данных: Идентификатор для набора значений, который может иметь объект данных.

определяющее слово: Forth слово, которое создает новое определение при выполнении.

определение: Forth выполняемая процедура, откомпилированная в словарь.

словарь: Расширяемая структура, которая содержит определения и связанные области данных.

отобразить: Послать один или более символов на устройство вывода пользователя.

зависимости от окружения: Неявные программные предположения об опциях реализации Forth системы или используемого оборудования. Например, программы, которые принимают размер ячейки больший, чем 16 бит, считаются зависимыми от окружения.

семантика выполнения: Поведение Forth определения, когда оно выполняется.

маркер выполнения: Значение, которое идентифицирует семантику выполнения определения.

искать: Просмотреть словарь для имени определения, соответствующего данной строке.

слово немедленного исполнения: Forth слово, чья семантика компиляции должна выполнить его семантику выполнения.

определяемое реализацией: Обозначает системное поведение или особенности, которые должны быть обеспечены и документированы системой, но которые более детально не определены этим Стандартом.

зависящее от реализации: Обозначает системное поведение или особенности, которые должны быть обеспечены системой, но которые более детально не определены этим Стандартом.

входной буфер: Область памяти, содержащей последовательность символов из входного источника, которая настоящее время доступна для программы.

входной источник: Устройство, файл, блок, или другой источник, который поставляет символы для пополнения входного буфера.

спецификация входного источника: Набор информации, описывающей подробное состояние входного источника, входного буфера, и области анализа. Этой информации достаточно, для корректного сохранения и восстановления для обеспечения операций вложенного синтаксического анализа в том же или другом входном источнике.

семантика интерпретации: Поведение Forth определения, когда его имя встречает

текстовый интерпретатор в состоянии интерпретации.

событие клавиатуры: Значение, полученное системой, обозначающее пользовательское действие на пользовательском входном устройстве. Термин "клавиатура" в этом документе не исключает другие типы пользовательских входных устройств.

строка: Последовательность символов, с последующим фактическим или подразумеваемым признаком конца строки.

область имен: Логическая область словаря, в которой сохранены имена определений.

число: В этом Стандарте, "число" используемое без других оговорок, означает "целое число". Аналогично "двойное число" означает "целое число две-ячейки".

анализ: Выбрать и исключить символьную строку из области анализа, используя заданный набор символов разделителей, называемых разделителями.

область анализа: Часть входного буфера, которая еще не анализировалась, и таким образом доступна системе для последующей обработки текстовым интерпретатором или другими операциями синтаксического анализа.

вывод отображаемого числа: Формат числового вывода, в который преобразовано число, используя Forth слова, который напоминает символическое "изображение" желаемого вывода.

программа: Законченная спецификация выполнения, для выполнения определенной функции (прикладной задачи) выраженная в форме исходного текста Forth.

принять: Получить символы из пользовательского входного устройства.

стек возвратов: Стек, который можно использовать для вложенного выполнения программ, выполнения do-loop, временной памяти, и других целей.

стандартное слово: Именованная Forth процедура, формально определенная в этом Стандарте.

пользовательское входное устройство: Устройство ввода данных, в настоящее время выбранное как источник получаемых данных, обычно клавиатура.

пользовательское выходное устройство: Устройство вывода, в настоящее время выбранное как адресат выводимых данных.

переменная: Именованная область данных расположенная и доступная по ее адресу памяти.

слово: В зависимости от контекста:
    1) имя Forth определения; или
    2) анализируемая непрерывная последовательность символов, которая могла быть именем Forth определения.

список слов: Список связанных имен Forth определений, который может быть исследован во время поиска в словаре.

набор слов: Набор определений Forth, сгруппированных вместе в этом Стандарте под именем, указывающим на некоторые общие свойства, типично их общая область функционирования.

2.2 Нотация

2.2.1 Числовая Нотация

Если иначе не определено, все ссылки к числам относятся к знаковым целым числам одна-ячейка. Диапазон значений величин показывается как {от... до}. Допустимый диапазон для содержимого адреса показывается в двойных фигурных скобках, особенно для содержимого переменных, например, BASE {{2... 36}}.

2.2.2 Стековая нотация

Входные и выходные параметры стековых определений описаны, с использованием следующей нотации:

```
( stack-id  before -- after )
```

Где stack-id определяет, какой стек описывается, before представляет типы данных параметров стека перед выполнением определения и after представляет их после выполнения. Символы, используемые в before и after показаны в таблице 3.1.

Stack-id стека потока-управления - "C:", stack-id стека данных - "S:", и stack-id стека возвратов - "R:". В случае отсутствия противоречий, stack-id стека данных, может быть опущен.

Когда есть альтернативные варианты представления after, они описаны как "after1 | after2". Вершина стека - справа. Показываются только те элементы, которые требуются для выполнения или предоставляются в результате выполнения определения.

2.2.3 Нотации анализа текста

Если, кроме использования параметров стека, определение анализирует текст, то этот текст обозначенный сокращением из таблицы 2.1, показывается окруженным двойными кавычками и помещается между before параметрами и "-" разделителем в первом описанном стеке, Например:

```
(S: before "аббревиатура анализируемого текста" - after).
```

Таблица 2.1 - сокращения анализатора текста

Сокращение	Описание
<char>	символ ограничитель, отмечающий конец анализируемой строки
<chars>	ноль или более последовательно расположенных символов char
<space>	символ ограничитель - пробел
<spaces>	ноль или более последовательно расположенных символов space
<quote>	символ ограничитель - двойная кавычка
<paren>	символ ограничитель - правая круглая скобка
<eol>	подразумеваемый ограничитель, отмечающий конец строки
ccc	анализируемая последовательность произвольных символов, исключая символ ограничитель
name	обозначение ограниченное пробелом, эквивалент ccc<space> или ccc<eol>

2.2.4 Словарные нотации

Словарные входы для каждого набора слов представлены стандартными ASCII последовательностями. Каждый вход словаря определяет ANS Forth слово и состоит из двух частей: индексной строки и семантического описания определения.

2.2.4.1 Индексная строка словаря

Индексная строка - однострочный вход содержащий, слева направо:
- Номер раздела, последние четыре цифры которого определяют уникальный последовательный номер для всех слов, включенных в этот Стандарт;
- ИМЯ-ОПРЕДЕЛЕНИЯ в верхнем регистре, фиксированной ширины, символы жирного начертания;
- Произношение на естественном языке в кавычках, если оно отличается от Английского языка;
- Обозначение набора слов из таблицы 2.2. Обозначение для расширения набора слов включает "EXT".

Таблица 2.2 - Обозначения наборов слов

Набор слов	Обозначение
Базовый набор слов	CORE
Блочный набор слов	BLOCK
Набор слов двойных чисел	DOUBLE
Набор слов исключений	EXCEPTION
Сервисный набор слов	FACILITY
Набор слов доступа к файлу	FILE
Набор слов для плавающей точки	FLOATING
Locals набор слов	LOCALS
Набор слов распределения памяти	MEMORY
Набор слов утилит	TOOLS
Набор слов порядка поиска	SEARCH
Набор слов обработки строк	STRING

2.2.4.2 Семантическое описание словаря

Первый параграф семантического описания содержит стековую нотацию для каждого стека, задействованного выполняемым словом. Остальные параграфы содержат текстовое описание семантики. Смотри 3.4.3 Семантика.

2.3 Ссылки

Следующие национальные и международные стандарты упомянуты в этом Стандарте:

ANSI X3.172-1990, Dictionary for information systems (2.1 Определения терминов);

ANSI X3.4-1974, American Standard Code for Information Interchange (ASCII) (3.1.2.1 Графические символы);

ISO 646-1983, ISO 7-bit coded characterset for information interchange, International Reference Version (IRV) (3.1.2.1 Графические символы);

ANSI/IEEE 754-1985, Floating-point standard (12.2.1 Определения терминов).

3. Условия применения

Система должна предоставлять все слова, определенные в 6.1 Основные слова. Она также может предоставлять любые слова, определенные в дополнительных и расширенных наборах слов. Никакое стандартное слово предоставленное системой не должно изменять состояние системы способом, изменяющим эффект выполнения любого другого стандартного слова, иным чем предусмотрено в этом Стандарте. Система может содержать не стандартные расширения предусмотренного здесь, при условии, что они совместимы с требованиями этого Стандарта.

Реализация слов и методов могущих использоваться в системе вне области рассмотрения этого Стандарта.

Система не требует предоставления всех слов в выполняемой форме. Реализация может предоставлять определения, включая определения Основного набора слов, только в исходной форме. Если так, то механизм для добавления определений в словарь - определяемый реализацией.

Программа, которая требует, чтобы система предоставила слова или методы, не определенные в этом Стандарте является зависимой от окружения.

3.1 Типы данных

Тип данных идентифицирует набор допустимых значений для объекта данных. Это - не свойство конкретного местоположения в памяти или позиции на стеке. Перемещение объекта данных не будет затрагивать его тип.

Система не требует никакой проверки типа данных. Неопределенная ситуация существует, если встречается неправильно определенный тип объекта данных.

Таблица 3.1 суммирует типы данных, используемые везде в этом Стандарте. Множественные варианты одного и того же типа данных, в обозначении определения обозначены последовательными цифровыми сносками, для их различения.

3.1.1 Отношения типов данных

Некоторые из типов данных - подтипы других типов данных. Тип данных i - подтип типа j, если и только если члены i - подмножество членов j. Следующий список представляет отношения подтипов, используя фразу "i = > j" чтобы обозначить, что "i является подтипом j". Отношения подтипа транзитивны; если i = > j и j = > k , тогда i = > k:

```
+n => u => x;
+n => n => x;
char => +n;
a-addr => c-addr => addr => u;
flag => x;
```



```
xt => x;
+d => d => xd;
+d => ud => xd.
```

Любое определение Forth, которое принимает параметр типа i должно также принять параметр являющийся подтипом i.

3.1.2 Символьные типы

Символы должны быть, по крайней мере, шириной в один адресуемый элемент, содержать по крайней мере восемь бит, и иметь размер меньше чем или равный размеру ячейки.

Символы, предоставленные системой должны включать графические символы {32..126}, которые представляют графические формы, показанные в таблице 3.2.

3.1.2.1 Графические символы

Графический символ является обычным отображаемым символом (например, A, #, &, 6). Эти значения и графические формы, показанные в таблице 3.2, приняты непосредственно от ANS X3.4-1974 (ASCII) и ISO 646-1983, International Reference Version (IRV). Графические формы символов вне hex диапазона {20.. 7E} - зависящие от реализации. Программы, использующие графический символ hex 24 (знак валюты) имеют зависимость от окружения.

Графическое представление символов не ограничено специфическим типом или стилем шрифта. Графика здесь это пример.

3.1.2.2 Управляющие символы

Все неграфические символы, включенные в определенный-реализацией набор символов определены в этом Стандарт как управляющие символы. В частности символы {0.. 31}, которые могли быть включены в определенный-реализацией набор символов - управляющие символы.

Программы, которые требуют возможность посылать, или принимать управляющие символы имеют зависимость от окружения.

Таблица 3.1 - Типы данных

Символ	Тип данных	Размер на стеке
flag	флаг	1 ячейка
true	флаг истина	1 ячейка
false	флаг ложь	1 ячейка
char	символ	1 ячейка
n	число со знаком	1 ячейка
+n	не отрицательное число	1 ячейка
u	без знаковое число	1 ячейка

n|u (1)	число	1 ячейка
x	любая ячейка	1 ячейка
xt	идентификатор исполнения	1 ячейка
addr	адрес	1 ячейка
a-addr	выровненный адрес	1 ячейка
c-addr	символьно-выровненный адрес	1 ячейка
d	число со знаком две-ячейки	2 ячейки
+d	не отрицат. число две-ячейки	2 ячейки
ud	беззнаковое число две-ячейки	2 ячейки
d|ud (2)	число две-ячейки	2 ячейки
xd	любая пара ячеек	2 ячейки
colon-sys	компиляция определения	определяемое реализацией
do-sys	do-loop структуры	определяемое реализацией
case-sys	CASE структуры	определяемое реализацией

of-sys	OF структуры	определяемое реализацией
orig	источник потока-управления	определяемое реализацией
dest	назначение потока-управления	определяемое реализацией
loop-sys	loop-control параметры	определяемое реализацией
nest-sys	вызовы определения	определяемое реализацией
i*x, j*x, k*x (3)	любой тип данных	0 или более ячеек

(1) Может быть число со знаком или без знака в зависимости от контекста.
(2) Может быть две-ячейки число со знаком или две-ячейки число без знака в зависимости от контекста.
(3) Может быть неопределенное число входов стека неопределенного типа. Для примеров использования, см. 6.1.1370 EXECUTE, 6.1.2050 QUIT.

Таблица 3.2 - Стандартные графические символы

Hex	IRV	ASCI	Hex	IRV	ASCI	Hex	IRV	ASCI	Hex	IRV	ASCI	Hex	IRV	ASCI	Hex	IRV	ASCI
20			30	0	0	40	@	@	50	P	P	60	`	`	70	p	p
21	!	!	31	1	1	41	A	A	51	Q	Q	61	a	a	71	q	q
22	"	"	32	2	2	42	B	B	52	R	R	62	b	b	72	r	r
23	#	#	33	3	3	43	C	C	53	S	S	63	c	c	73	s	s
24	-	$	34	4	4	44	D	D	54	T	T	64	d	d	74	t	t
25	%	%	35	5	5	45	E	E	55	U	U	65	e	e	75	u	u
26	&	&	36	6	6	46	F	F	56	V	V	66	f	f	76	v	v
27	'	'	37	7	7	47	G	G	57	W	W	67	g	g	77	w	w
28	(	(	38	8	8	48	H	H	58	X	X	68	h	h	78	x	x
29	)	)	39	9	9	49	I	I	59	Y	Y	69	i	i	79	y	y
2A	*	*	3A	:	:	4A	J	J	5A	Z	Z	6A	j	j	7A	z	z
2B	+	+	3B	;	;	4B	K	K	5B	[	[	6B	k	k	7B	{	{
2C	,	,	3C	<	<	4C	L	L	5C	\	\	6C	l	l	7C	|	|
2D	-	-	3D	=	=	4D	M	M	5D	]	]	6D	m	m	7D	}	}
2E	.	.	3E	>	>	4E	N	N	5E	^	^	6E	n	n	7E	~	~
2F	/	/	3F	?	?	4F	O	O	5F	_	_	6F	o	o			

Определенный-реализацией фиксированный размер ячейки определен адресуемыми элементами и соответствующим количеством бит. См. E.2 Аппаратные особенности.

Ячейки должны быть, по крайней мере, размером в один адресуемый элемент, и содержать, по крайней мере, 16-бит. Размер ячейки должен быть кратным целым размера символа. Элементы стека данных, элементы стека возвратов, адреса, идентификаторы исполнения, флаги, и целые числа - размером в одну ячейку.

3.1.3.1 Флаги

Флаги могут иметь одно из двух логических значений: true или false. Программы, которые используют флаги как арифметические операнды, имеют зависимость от окружения.

True флаг, возвращенный стандартным словом должен иметь значение одна-ячейка со всеми установленными битами. A false флаг, возвращенный стандартным словом должен иметь значение одна-ячейка со всеми сброшенными битами.

3.1.3.2 Целые числа

Определенный реализацией диапазон целых чисел со знаком должен включать {-32767.. +32767}.

Определенный реализацией диапазон неотрицательных целых чисел должен включать {0.. 32767}.

Определенный реализацией диапазон целых чисел без знака должен включать {0.. 65535}.

3.1.3.3 Адреса

Адрес идентифицирует местоположение в области данных с размером одного адресуемого элемента, который программа может выбирать из памяти или заносить в память, кроме ограничений, установленных в этом Стандарте. Размер адресуемого элемента определен в битах. Каждое отличное значение адреса идентифицирует точно один такой запоминающий элемент. См. 3.3.3 Область данных.

Набор символьно-выровненных адресов, адреса которыми можно обращаться к символу - определенное-реализацией подмножество всех адресов. Добавление размера символа к символьно-выровненному адресу должно произвести другой символьно-выровненный адрес.

Набор выровненных адресов - определенное-реализацией подмножество символьно-выровненных адресов. Добавление размера ячейки к выровненному адресу должно произвести другой выровненный адрес.

3.1.3.4 Строки со счетчиком

Строка со счетчиком в памяти идентифицирована адресом (c-addr)символа её длины.

Символ длины строки со счетчиком должен содержать число символов данных двоичного представления, между нулем и определенной-реализацией максимальной длиной для строки со счетчиком. Максимальная длина строки со счетчиком должна быть, по крайней мере, 255.

3.1.3.5 Идентификаторы исполнения

Различные определения могут иметь один и тот же идентификатор исполнения, если определения эквивалентны.

3.1.4 Типы пара-ячеек

Пара-ячеек в памяти состоит из последовательности двух непрерывных ячеек. Ячейка с меньшим адресом - первая ячейка, и её адрес используется, для идентификации пары ячеек. Если иначе не определено, пара ячеек на стеке состоит из первой ячейки непосредственно выше второй ячейки.

3.1.4.1 Целые числа две-ячейки

На стеке, ячейка содержащая самую старшую часть целого числа две-ячейки должна быть выше ячейки содержащей самую младшую часть.

Определенный-реализацией диапазон целых чисел две-ячейки со знаком должен включать {-2147483647..+2147483647}.

Определенный-реализацией диапазон не отрицательных целых две-ячейки чисел должен включать {0..2147483647}.

Определенный-реализацией диапазон без знаковых целых чисел две-ячейки должен включать {0..4294967295}. Размещение нулевого целого числа одна-ячейка на стеке выше целого числа одна-ячейка без знака производит целое число две-ячейки без знака с тем же самым значением. См. 3.2.1.1 Внутреннее представление числа.

3.1.4.2 Символьные строки

Строка определенная как пара ячеек (c-addr u) представляет её стартовый адрес и длину в символах.

3.1.5 Системные типы

Системные типы данных определяют допустимые комбинации слов в течение компиляции и выполнения.

3.1.5.1 Системные типы компиляции

Эти типы данных обозначают нуль или большее количество элементов на стеке потока-управления (см. 3.2.3.2). Возможное присутствие таких элементов на стеке данных означает, что любые элементы уже там находящиеся должны быть недоступны программе, пока не использованы элементы стека потока-управления.

Зависящие-от-реализации данные, сгенерированные после начала компиляции определения и использованные при его завершении обозначены символом colon-sys повсюду этом Стандарте.

Зависящие-от-реализации данные, сгенерированные после начала компиляции do-loop структуры типа DO ... LOOP и использованные при её завершении обозначены символом do-sys повсюду этом Стандарте.

Зависящие-от-реализации данные, сгенерированные после начала компиляции CASE ... ENDCASE структуры и использованные при её завершении обозначены символом case-sys повсюду этом Стандарте.

Зависящие-от-реализации данные, сгенерированные после начала компиляции OF ... ENDOF структуры и использованные при её завершении обозначены символом of-sys повсюду этом Стандарте.

Зависящие-от-реализации данные, сгенерированные и использованные при выполнении другого стандартного слова потока-управления обозначены символами orig и dest повсюду в этом Стандарте.

3.1.5.2 Системные типы выполнения

Эти типы данных обозначают нуль или большее количество элементов на стеке возвратов. Их возможное присутствие означает, что любые элементы на стеке возвратов уже должны быть недоступны программе, пока элементы системных типов выполнения не использованы.

Зависящие-от-реализации данные, сгенерированные после начала выполнения определения и использованные после выхода из него обозначены символом nest-sys повсюду в этом Стандарте.

Зависящие-от-реализации параметры loop-control используемые для управления выполнением do-loops обозначены символом loop-sys повсюду в этом Стандарте. Loop-control параметры должны быть доступны внутри do-loop для слов, которые используют или изменяют эти параметры, слова типа I, J, LEAVE и UNLOOP.

3.2 Среда выполнения

3.2.1 Числа

3.2.1.1 Внутреннее представление числа

Этот Стандарт допускает представления чисел и арифметические операции с обратным кодом (дополнение до единицы), с дополнительным кодом (дополнение до двух), или величин со знаком. Арифметический нуль представлен как значение одна-ячейка со всеми сброшенными битами.

Представление числа как скомпилированного литерала или в памяти - зависящее от реализации.

3.2.1.2 Преобразование цифр

Числа должны быть представлены внешне с использованием символов из стандартного набора символов.

Преобразование между внутренними и внешними формами цифры должно производиться следующим образом:

Значение в BASE - основание системы счисления для преобразования чисел. Цифра имеет значение в пределах от нуля до на единицу меньшего содержимого BASE. Цифра с нулевым значением соответствует символу "0". Это представление цифр продолжается по набору символов до десятичного значения девять соответствующему символу "9". Для цифр начинающихся с десятичного значения десять используются графические символы начинающиеся с символа "A".

Это соответствие продолжается вплоть до, и включая цифру с десятичным значением тридцать пять, которая представляется символом "Z". Преобразование цифр вне этого диапазона - определенное реализацией.

3.2.1.3 Изображение чисел в свободном поле

Изображение чисел в свободном поле использует символы, описанные в Преобразовании цифр, без предшествующих нулей, в поле точно по размеру преобразованной строки плюс конечный пробел. Если число нулевое, наименьшая значащая цифра не рассматривается как начальный ноль. Если число отрицательное, отображается ведущий знак "минус".

Изображение чисел может использовать буфер выходной строки отображаемого числа, для хранения частично преобразованных строк (см. 3.3.3.6 Другие временные области).

3.2.2 Арифметика

3.2.2.1 Целочисленное деление

Деление производит частное q и остаток r, деля операнд a на операнд b. Операции деления возвращают q, r, или оба. Тождество b * q + r = a должно сохраняться для всех a и b.

Когда делятся целые числа без знака, и остаток - не нулевой, q - самое большее целое число меньше чем истинное частное.

Когда делятся целые числа со знаком, и остаток - не нулевой, и a и b имеют одинаковый знак, q - самое большее целое число меньше чем истинное частное. Если только один операнд отрицательный, округлено ли q к отрицательной бесконечности (минимальное деление) или округлено к нулю (симметричное деление) - определенное реализацией.

Минимальное деление - целочисленное деление, в котором остаток имеет знак делителя или нулевой, и частное округлено к его арифметическому минимуму. Симметричное деление - целочисленное деление, в котором остаток имеет знак делимого или нуль и частное - математическое частное "округленное к нулю" или "обрезанное". Примеры каждого показаны в таблицах 3.3 и 3.4.

В случаях, где операнды отличаются по знаку и в вопросе направления округления, программа должна или включать код, генерирующий желательную форму деления, не полагаясь на результат по умолчанию определенный реализацией, или иметь зависимость от окружения желательного направления округления.

Таблица 3.3 - Пример минимального деления

Делимое	Делитель	Остаток	Частное
10	7	3	1
-10	7	4	-2
10	-7	-4	-2
-10	-7	-3	1

Таблица 3.4 - Пример симметричного деления

Делимое	Делитель	Остаток	Частное
10	7	3	1
-10	7	-3	-1
10	-7	3	-1
-10	-7	-3	1

3.2.2.2 Другие целочисленные операции

Во всех целочисленных арифметических операциях переполнение и антипереполнение должны игнорироваться. Возвращаемое значение, когда происходит переполнение или антипереполнение - определенное реализацией.

3.2.3 Стеки

3.2.3.1 Стек Данных

Объекты на стеке данных должны быть шириной в одну ячейку.

3.2.3.2 Стек потока-управления

Стек потока-управления - это список типа "последним пришел", "первым вышел",

чьи элементы определяют допустимые соответствия слов потока-управления и ограничения, наложенные на использование стека данных в течение компиляции управляющих структур.

Элементы стека потока-управления - системные типы данных компиляции.

Стек потока-управления, может, но не обязательно, физически существовать в реализации. Если он существует, то может быть, но не обязательно, реализован с использованием стека данных. Формат стека потока-управления - определяется реализацией. Так как стек потока-управления может быть реализован с использованием стека данных, элементы, помещенные на стек данных недоступны для программ после помещения элементов на стек потока-управления, и остаются недоступным до удаления элементов стека потока-управления.

3.2.3.3 Стек возвратов

Элементы на стеке возвратов должны состоять из одной или более ячеек. Система может использовать стек возвратов зависящим от реализации способом в течение компиляции определений, в течение выполнения do-loops, и для сохранения вложенной информации во время выполнения.

Программа может использовать стек возвратов как временную память в течение выполнения определения, подчиняясь следующим ограничениям:

- Программа не должна обращаться к значениям на стеке возвратов (используя R@, R>, 2R@ или 2R>) для элементов, не размещенных туда с помощью >R или 2>R;
- Программа не должна обращаться изнутри do-loop к значениям, помещенным на стек возвратов прежде, чем был открыт цикл;

- Все значения, помещенные на стек возвратов в пределах do-loop должны быть удалены прежде, чем будут выполнены I, J, LOOP, +LOOP, UNLOOP, или LEAVE;
- Все значения, помещенные на стек возвратов в пределах определения должны быть удалены перед тем, как определение будет закончено или прежде, чем будет выполнен EXIT.

3.2.4 Терминал Оператора

См. 1.2.2 Исключения.

3.2.4.1 Пользовательское устройство ввода

Метод выбора пользовательского устройства ввода данных - определенный реализацией.

Метод индикации конца входной строки текста - определенный реализацией.

3.2.4.2 Пользовательское устройство вывода

Метод выбора пользовательского устройства вывода - определенный реализацией.

3.2.5 Запоминающее устройство большой емкости

Система не предусматривает какие либо стандартные слова для доступа к запоминающему устройству большой емкости. Если система предоставляет какое либо стандартное слово для доступа к запоминающему устройству большой емкости, она должна также дополнить Блочный набор слов.

3.2.6 Запросы к окружению

Пространства имен для ENVIRONMENT? и определений непересекающиеся. Названия определений являющиеся такими же как ENVIRONMENT? строки не должны мешать операции ENVIRONMENT?.
Таблица 3.5 содержит допустимые входные строки и соответствующие возвращаемые значения для запросов к программной среде с помощью ENVIRONMENT?.

Таблица 3.5 - Строки запросов к окружению

Строка	Тип	Константа	Значение
/COUNTED-STRING	n	yes	максимальный размер строки со счетчиком, в символах
/HOLD	n	yes	размер буфера выходной строки отображаемого числа, в символах
/PAD	n	yes	размер рабочей области, указываемой PAD, в символах
ADDRESS-UNIT-BITS	n	yes	размер одного адресуемого элемента, в битах
CORE	flag	no	true если присутствует полный основной набор слов (то есть, не подмножество как определено в 5.1.1)
CORE-EXT	flag	no	true если присутствует расширенный набор слов
FLOORED	flag	yes	true если минимальное деление по умолчанию
MAX-CHAR	u	yes	максимальное значение какого либо символа в наборе символов определенном реализацией

MAX-D	d	yes	наибольшее используемое знаковое двойное число
MAX-N	n	yes	наибольшее используемое знаковое число
MAX-U	u	yes	наибольшее используемое без знаковое число
MAX-UD	ud	yes	наибольшее используемое без знаковое двойное число
RETURN-STACK-CELLS	n	yes	максимальный размер стека возвратов, в ячейках
STACK-CELLS	n	yes	максимальный размер стека данных, в ячейках

Если запрос к окружению (с помощью ENVIRONMENT?) возвращает false (то есть, неизвестное) в ответ на строку, последующие запросы, использующие ту же самую строку могут возвратить true. Если запрос возвращает true (то есть, известный) в ответ на строку, последующие запросы с той же самой строкой должны также возвращать true. Если запрос, обозначенный как константа в вышеупомянутой таблице, возвращает true и значение в ответ на строку, последующие запросы с той же самой строкой должны возвратить true и то же самое значение.

3.3 Словарь Forth

Слова Forth организованы в структуру называемую словарем. В то время как форма этой структуры не определена Стандартом, она может быть описана как состоящая из тех логических частей: области имени, области кода, и области данных. Логическое разделение этих частей не требует их физического разделения.

Программа не должна брать или сохранять данные в место вне области данных. Неоднозначное условие существует, если программа адресует область имени или область кода.

3.3.1 Область имени

Отношения между областью имени и областью данных - зависящие от реализации.

Структура списка слов - зависящая от реализации. Когда в списке слов существуют дубликаты имен, в процессе поиска имени должен быть найден последний определенный дубликат.

3.3.1.2 Имена определений

Имена определений должны содержать {1.. 31} символов. Система может позволять или запрещать создание имен определений, содержащих не стандартные символы.

Программы, которые используют строчные буквы для стандартных имен определений или зависят от регистро-чувствительных свойств системы - имеют зависимость от окружения.

Программа не должна создавать имена определений, содержащие неграфические символы.

3.3.2 Область кода

Отношения между областью кода и областью данных - зависящие от реализации.

3.3.3 Область данных

Область данных - единственная логическая область словаря, для которой предусмотрены стандартные слова, для того чтобы распределять и обращаться к областям памяти. Этими областями являются: непрерывные области, переменные, области текстовых констант, буферы ввода, и другие временные области, каждая из которых описана в следующих разделах. Программа может читать из или записывать в эти области, если иначе не определено.

3.3.3.1 Выравнивание адреса

Большинство адресов, используемых в ANS Forth это выровненные адреса (обозначены - a-addr) или символьно-выровненные (обозначены c-addr). ALIGNED, CHAR+, и арифметические операции могут изменять состояние выравнивания адреса на стеке. CHAR+ применяемое к выровненным адресам возвращает символьно-выровненные адреса, которые могут использоваться только, для обращения к символам. Применение CHAR+ к символьно-выровненному адресу производит следующий символьно-выровненный адрес. Добавление или вычитание произвольного числа к адресу может производить не выровненный адрес, который не может быть использован для выборки или сохранения чего-нибудь. Единственный путь найти следующий выровненный адреса - это ALIGNED. Неопределенная ситуация существует когда @, !, , (запятая), +!, 2@, or 2! используется с адресом, который не выровнен, или когда C@, C!, или C, используется с адресом, который не символьно-выровненный.

Определения 6.1.1000 CREATE и 6.1.2410 VARIABLE требуют, чтобы определения созданные ими возвращали выровненные адреса.

После того, как определения откомпилированы, или выполнено слово ALIGN, указатель области данных гарантировано будет выровненным.

3.3.3.2 Непрерывные области

Система гарантирует, что зона области данных распределенная с использованием ALLOT, , (запятая) , C, (c-запятая) , и ALIGN, будет неразрывна с предыдущей зоной, распределенной одним из вышеупомянутых слов, если не применяются ограничения из следующих параграфов. Указатель области данных HERE всегда идентифицирует начало следующей зоны области данных, которая будет распределена. Как только последовательные распределения сделаны, увеличивается указатель области данных. Программа может исполнять адресную арифметику в пределах непрерывно распределенной области. Последняя зона области данных распределенная с использованием вышеупомянутых операторов может быть освобождена, распределением соответствующей зоны отрицательного размера, используя ALLOT, с соблюдением ограничений следующих параграфов.

CREATE устанавливает начало непрерывной зоны области данных, чей стартовый адрес возвращен созданным CREATE определением. Эта зона заканчивается, компиляцией следующего определения.

Так как реализация не накладывает ограничений на распределение области данных для использования кодом, вышеупомянутые операторы не обязаны производить непрерывные зоны области данных, если между распределениями были добавлены или удаленны из словаря определения. Неопределенная ситуация существует если освобожденная память содержит определения.

3.3.3.3 Переменные

Область, распределенная для переменной может состоять из нескольких несмежных участков с областями впоследствии распределенными с помощью , (запятая) или ALLOT. Например, в:

```
VARIABLE X  1 CELLS ALLOT
```

зона X и распределенная ALLOT зона могут быть несмежными.

Некоторые поставляемые-системой переменные, типа STATE, ограничены доступом только для чтения.

3.3.3.4 Области текстовых констант

Области текстовых констант, заданные строками, откомпилированными с помощью S" и C", могут быть только-для-чтения.

Программа не должна сохранять в областях текстовых констант созданных с помощью S" и C", ни в другие только-для-чтения системные переменные или только-для-чтения временные области. Неопределенная ситуация существует когда программа пытается сохранять в области только-для-чтения.

3.3.3.5 Входные буферы

Адрес, длина, и содержание входного буфера могут быть временными. Программа не должна записывать во входной буфер. В отсутствии каких либо дополнительных наборов слов предусматривающих альтернативные входные источники, входной буфер - это либо входной буфер терминала, используемый QUIT для хранения одной строки из пользовательского устройства ввода данных, либо буфер определяемый EVALUATE. Во всех случаях SOURCE возвращает начальный адрес и длину в символах текущего входного буфера.

Минимальный размер входного буфера терминала должен быть 80 символов.

Адрес и длина возвращенная SOURCE, строки возвращенной PARSE, и непосредственно вычисленный адрес входного буфера имеют силу только пока текстовый интерпретатор не делает ввод-вывод для наполнения входного буфера, или не изменен входной источник.

Программа может изменять размер области анализа, изменяя содержимое >IN в пределах ограничений, наложенных этим Стандартом. Например, если содержимое >IN сохранено перед операцией анализа и восстановлено впоследствии, текст, который анализировался, будет доступен снова для последующих операций синтаксического анализа. Степень допустимого повторного позиционирования с использованием этого метода зависит от входного источника (см. 7.3.3 Области Блочного буфера и 11.3.4 Входной источник).

Программа может непосредственно исследовать входной буфер, используя его адрес и длину возвращенную SOURCE; начало области анализа в пределах входного буфера индексировано числом в >IN. Значения имеют силу на ограниченный срок. Неопределенная ситуация существует если программа изменяет содержимое входного буфера.

3.3.3.6 Другие временные области

Зоны области данных, идентифицированные PAD, WORD, и #> (буфер выходной строки отображаемого числа) могут быть временными. Их адреса и содержание могут стать недействительными после того, как:

- создано определение через определяющее слово;

- определения скомпилированы с помощью : или :NONAME;
- область данных выделена, с помощью ALLOT, , (запятая), C, (с-запятая), или ALIGN.

Предыдущее содержание областей, идентифицированных WORD и #> , может быть недействительно после каждого использования этих слов. Кроме того, области, возвращенные WORD и #> , могут накладываться в памяти. Следовательно, использование одного из этих слов может разрушать область, возвращенную ранее другим словом. Другие слова, которые создают выходную строку отображаемого числа (<#, #, #S, и HOLD) могут также изменять содержание этих областей. Слова, которые выводят числа, могут быть реализованы с использованием слов вывода отображаемых чисел. Следовательно . (точка), .R, .S, ?, D., D.R, U., и U.R могут также разрушать области.

Размер рабочей области, чей адрес возвращен PAD, должен быть, по крайней мере, 84 символа. Содержание области адресованной PAD предназначено, для того чтобы быть под полным контролем пользователя: никакие слова, определенные в этом Стандарте не помещают что-либо в эту область, также при изменении распределения области данных как описано в 3.3.3.2 непрерывная область может изменять адрес возвращаемый PAD. Нестандартные слова, предусмотренные реализацией, могут использовать PAD, но такое использование должно быть документировано.

Размер области, идентифицированной WORD должен быть, по крайней мере, 33 символа.

Размер буфера строки отображаемого числового вывода должен быть, по крайней мере, (2*n) + 2 символов, где n - число бит в ячейке. Программы, которые рассматривают её фиксированной областью с не изменяющимися параметрами доступа - имеют зависимость от окружения.

3.4

После запуска система должна иметь возможность интерпретировать, как описано в 6.1.2050 QUIT, Forth исходный текст полученный в интерактивном режиме из пользовательского устройства ввода данных.

Такие интерактивные системы обычно предоставляют "приглашение", указывающее, что они принимают пользовательский запрос и действуют в соответствии с ним. Определенное реализацией приглашение Forth должно содержать слово "OK" в некоторой комбинации верхних или строчных букв.

Текстовая интерпретация (см. 6.1.1360 EVALUATE и 6.1.2050 QUIT) должна повторять последовательные шаги до опустения области анализа, или возникновения неопределенной ситуации:

а) Пропустить ведущие пробелы и выделить имя (см. 3.4.1);
б) Просмотреть область имен словаря (см. 3.4.2). Если имя определения, соответствующее строке найдено:
   1) при интерпретации, исполнить семантику интерпретации определения (см. 3.4.3.2), и продолжить с а);
   2) при компиляции, исполнить семантику компиляции определения (см. 3.4.3.3), и продолжить с а).
в) Если имя определения, соответствующее строке не найдено, пытаться преобразовать строку в число (см. 3.4.1.3). Если успешно:

   1) при интерпретации, положить число на стек данных, и продолжить с а);
   2) при компиляции, скомпилировать код, который при выполнении положит число на стек (см. 6.1.1780 LITERAL), и продолжить с а);
г) Если неудачно, существует неопределенная ситуация (см. 3.4.4).

3.4.1 Синтаксический анализ

Если иначе не отмечено, число анализируемых символов может быть от нуля до определенной реализацией максимальной длины строки со счетчиком.

Если область анализа пуста, то есть когда число в >IN равно длине входного буфера, или не содержит других символов кроме разделителей, то выбранная строка пуста. Иначе, выбранная строка начинается со следующего символа в области анализа, который является символом, индексированным содержимым >IN. Неопределенная ситуация существует если число в >IN больше чем размер входного буфера.

Если символы-разделители присутствуют в области анализа после начала выбранной строки, строка продолжается вплоть до символа как раз перед первым таким разделителем, и число в >IN изменяется, так чтобы указывать непосредственно после этого разделителя, таким способом удаляются анализируемые символы и разделитель из области анализа. Иначе, строка продолжается вплоть до последнего символа в области анализа, и число в >IN изменяется на длину входного буфера, таким способом освобождая область анализа.

Синтаксический анализ может изменять содержимое >IN, но не должен затрагивать содержимое буфера ввода. В частности, если значение в >IN сохранено перед стартом анализа, возврат >IN к этому значению немедленно после анализа, должен восстановить область анализа без потери данных.

3.4.1.1 Разделители

Если разделитель - пробел (hex 20 (BL)) управляющие символы могут быть интерпретированы как разделители. Набор условий, если таковые вообще имеются,

согласно которым разделитель "пробел" соответствует управляющим символам - определенное реализацией.

Для пропуска ведущих разделителей необходимо пропустить ноль или более смежных разделителей в области анализа перед синтаксическим анализом.

3.4.1.2 Синтаксис

Forth имеет простой, операторно-упорядоченный синтаксис. Фраза A B C возвращает значения, как будто A было выполнено первым, затем B и наконец C. Слова, которые вызывают отклонения от этого линейного потока-управления, называются словами потока-управления. Структуры потока-управления - комбинации слов потока-управления, чьи стековые эффекты являются совместимой формой. Примеры типичного использования даны для каждого слова потока-управления в Приложении A.

Синтаксис Forth расширяем; например, в терминах существующих могут быть определены новые слова потока-управления.

Этот Стандарт не требует синтаксиса или средств проверки структуры программы.

3.4.1.3 Преобразование входных чисел текстового интерпретатора

При преобразовании входных чисел, текстовый интерпретатор должен распознавать положительные и отрицательные числа; отрицательные числа, представленные отдельным знаком "минус" (символом "-") предшествующим цифрам. Значение в BASE - основание системы счисления для преобразования чисел.

3.4.2 Поиск имен определений

Строка соответствует имени определения, если каждый символ в строке соответствует соответствующему символу в строке, используемой как имя определения, когда определение было создано. Регистровая чувствительность (действительно ли символы верхнего регистра соответствуют символам нижнего регистра) - определенная реализацией. Система может быть либо чувствительной к регистру и трактовать символы верхнего - и нижнего регистра как различные и не соответствующие, либо нечувствительной к регистру, игнорируя различия в регистре при поиске.

Соответствие символов верхнего - и нижнего регистра с алфавитными символами в расширенном наборе символов типа интернациональных символов с диакритическим знаком - определенное реализацией.

Система должна быть способна к нахождению имен определений, определенных в этом Стандарте, когда они записаны символами в верхнем регистре.

3.4.3 Семантика

Семантика определения Forth реализована машинным кодом или последовательностью идентификаторов исполнения или другими представлениями. Они в значительной степени определены стековой нотацией в словарных входах, которые показывают какие значения должны быть использованы и произведены. Проза в каждом входе словаря далее определяет поведение определения.

Каждое определение Forth может иметь несколько поведений, описанных в следующих разделах. Термины "семантика инициации" и "семантика времени-выполнения" относятся к фрагментам определения, и имеют значение только в пределах

индивидуальных входов словаря, где они появляются.

3.4.3.1 Семантика выполнения

Семантика выполнения каждого определения Forth определена в секции "Выполнение:" его входа словаря. Когда определение имеет только одно указанное поведение, метка опущена.

Выполнение может происходить неявно, когда выполняется определение, в которое оно было скомпилировано, или явно, когда его идентификатор исполнения передается в EXECUTE. Семантика выполнения синтаксически правильного определения при условиях отличных от тех, что определены в этом Стандарте - зависящая от реализации.

Входы словаря для определяющих слов включают семантику выполнения для нового определения в секции "имя Выполнение:".

3.4.3.2 Семантика интерпретации

Если иначе не определено в секции "Интерпретация:" входа словаря, семантика интерпретации определения Forth - это его семантика выполнения.

Система должна быть способна выполнить в состоянии интерпретации все определения из основного набора слов и любые определения, включенные в дополнительный набор слов или слова расширенного набора, чья семантика интерпретации определена в этом Стандарте.

Система должна быть способна выполнить в состоянии интерпретации любые новые определения созданные в соответствии с 3. Условия применения.

3.4.3.3 Семантика компиляции

Если иначе не определено в секции "Компиляция:" входа словаря, семантика компиляции определения Forth должна добавить его семантику выполнения в конец семантики выполнения текущего определения.

3.4.4 Возможное действие на неопределенную ситуацию

Когда существует неопределенная ситуация, система может предпринимать одно или более из следующих действий:

- игнорировать, и продолжить;
- вывести сообщение;
- выполнить специфическое слово;
- установить состояние интерпретации и начать текстовую интерпретацию;
- предпринимать другие действия определенные реализацией;
- предпринимать зависящие от реализации действия.

Ответ на специфическую неопределенную ситуацию не обязан быть один и тот же при всех обстоятельствах.

3.4.5 Компиляция

Программа не должна пытаться выполнять вложенную компиляцию определений.

В течение компиляции текущего определения, программа не должна выполнять

никакое определяющее слово, :NONAME, или любое определение, которое распределяет область данных словаря. Компиляция текущего определения может быть приостановлена, с помощью [(левая скобка) и продолжена, с помощью] (правая скобка). Если компиляция текущего определения приостановлена, программа не должна выполнять никакое определяющее слово, :NONAME, или любое определение, которое распределяет область данных словаря.

4. Документационные требования

Когда невозможно или неосуществимо для системы или программы определить ее детали поведения, допустимо установить, что поведение - точно неопределяемое и объяснить обстоятельства и причины, почему это - так.

4.1 Системная документация

4.1.1 Опции определяемые реализацией

В следующем списке пункты определенные реализацией представляют характеристики и варианты, оставленные на усмотрение разработчика, при условии, что выполнены требования этого Стандарта. Система должна документировать значения, или поведения, для каждого пункта.

- требования к выравниванию адреса (3.1.3.3 Адреса);
- поведение 6.1.1320 EMIT для неграфических символов;
- символьное редактирование 6.1.0695 ACCEPT и 6.2.1390 EXPECT;
- набор символов (3.1.2 Символьные типы, 6.1.1320 EMIT, 6.1.1750 KEY);
- требования символьно-выровненного адреса (3.1.3.3 Адреса);
- характеристики, соответствующие расширениям набора символов (3.4.2 Поиск имен определений);
- условия, при которых управляющие символы соответствуют пробельному разделителю (3.4.1.1 Разделители);
- формат стека потока-управления (3.2.3.2 Стек потока-управления);
- преобразование цифр больших, чем тридцать пять (3.2.1.2 Преобразование цифр);
- изображение после окончания ввода в 6.1.0695 ACCEPT и 6.2.1390 EXPECT;
- последовательность сброса исключения (как в 6.1.0680 ABORT");
- признак конца входной строки (3.2.4.1 Пользовательское устройство ввода данных);
- максимальный размер строки со счетчиком, в символах (3.1.3.4 Строки со счетчиком, 6.1.2450 WORD);
- максимальный размер анализируемой строки (3.4.1 Синтаксический анализ);
- максимальный размер имени определения, в символах (3.3.1.2 Имена определений);
- максимальная длина строки для 6.1.1345 ENVIRONMENT?, в символах;
- метод выбора 3.2.4.1 Пользовательское устройство ввода;
- метод выбора 3.2.4.2 Пользовательское устройство вывода;
- методы компиляции словаря (3.3 Словарь Forth);
- число бит в одном адресуемом элементе (3.1.3.3 Адреса);
- представление числа и арифметика (3.2.1.1 Внутреннее представление числа);
- диапазоны для n, +n, u, d, +d, и ud (3.1.3 Типы одна-ячейка, 3.1.4 Типы пара-ячеек);
- зоны области данных только-для-чтения (3.3.3 Область данных);
- размер буфера в 6.1.2450 WORD (3.3.3.6 Другие временные области);
- размер одной ячейки в адресуемых элементах (3.1.3 Типы одна-ячейка);
- размер одного символа в адресуемых элементах (3.1.2 Символьные типы);
- размер входного буфера клавиатуры терминала (3.3.3.5 Входные буферы);
- размер буфера строки вывода отображаемого числа (3.3.3.6 Другие временные области);
- размер рабочей области, чей адрес возвращен 6.2.2000 PAD (3.3.3.6 Другие

временные области);
- системные регистро-чувствительные характеристики (3.4.2 Поиск имен определений);
- системное приглашение (3.4 Интерпретатор текста Forth, 6.1.2050 QUIT);
- тип округления деления (3.2.2.1 Целочисленное деление, 6.1.0100 */, 6.1.0110 */MOD, 6.1.0230 /, 6.1.0240 /MOD, 6.1.1890 MOD);

- значения 6.1.2250 STATE, если true;
- значения, возвращаемые после арифметического переполнения (3.2.2.2 Другие целочисленные операции);
- может ли текущее определение быть найдено после 6.1.1250 DOES> (6.1.0450 :).

4.1.2 Неоднозначные условия

Система должна документировать системное действие, предпринимаемое на каждое общее или специфическое неоднозначное условие, идентифицированное в этом Стандарте. См. 3.4.4 Возможное действие на неопределенную ситуацию.

Следующие общие неоднозначные условия могут происходить из-за комбинации факторов:

- имя не определено ни допустимым именем определения, ни допустимым числом в течение интерпретации текста (3.4 Интерпретатор текста Forth);
- имя определения превысило максимальную допустимую длину (3.3.1.2 Имена определений);
- адресование области, не перечисленной в 3.3.3 Области данных;
- тип параметра, несовместимый с указанным входным параметром, например, передача флага слову ожидающему n (3.1 Типы данных);
- попытка получить идентификатор исполнения (например, с помощью 6.1.0070 ', 6.1.1550 FIND, и т.д.) определения с неопределенной семантикой интерпретации;
- деление на ноль (6.1.0100 */, 6.1.0110 */MOD, 6.1.0230 /, 6.1.0240 /MOD, 6.1.1561 FM/MOD, 6.1.1890 MOD, 6.1.2214 SM/REM, 6.1.2370 UM/MOD, 8.6.1.1820 M*/);
- недостаточный размер стека данных или размер стека возвратов (переполнение стека);
- недостаточно места для параметров loop-control;
- недостаточно места в словаре;
- интерпретация слова с неопределенной семантикой интерпретации;
- изменение содержания входного буфера или строкового литерала (3.3.3.4 Области текстовых констант, 3.3.3.5 Входные буферы);
- переполнение строки вывода отображаемого числа;
- переполнение анализируемой строки;
- производимый результат вне диапазона, например умножение (с использованием *) приводит к значению слишком большому, чтобы быть представленным целым числом одна-ячейка (6.1.0090 *, 6.1.0100 */, 6.1.0110 */MOD, 6.1.0570 >NUMBER, 6.1.1561 FM/MOD, 6.1.2214 SM/REM, 6.1.2370 UM/MOD, 6.2.0970 CONVERT, 8.6.1.1820 M*/);
- чтение из пустого стека данных или стека возвратов (выход за нижнюю границу стека);
- неожиданный конец входного буфера, приводящий к попытке использовать строку нулевой длины как имя;

Следующие специфические неоднозначные условия отмечены в словарных входах важных слов:

- >IN больше чем размер входного буфера (3.4.1 Синтаксический анализ);
- 6.1.2120 RECURSE появляется после 6.1.1250 DOES>;
- параметр источника ввода, отличный от текущего входного источника для 6.2.2148 RESTORE-INPUT;
- область данных, содержащая определения освобождена (3.3.3.2 Непрерывные

области);
- чтение-запись области данных с неправильным выравниванием (3.3.3.1 Выравнивание адреса);
- указатель области данных, не должным образом выровненный (6.1.0150 ,, 6.1.0860 C,);
- меньше чем u+2 элементов в стеке (6.2.2030 PICK, 6.2.2150 ROLL);
- параметры loop-control не доступны (6.1.0140 +LOOP, 6.1.1680 I, 6.1.1730 J, 6.1.1760 LEAVE, 6.1.1800 LOOP, 6.1.2380 UNLOOP);

- самое последнее определение не имеет имени (6.1.1710 IMMEDIATE);
- имя, не определенное 6.2.2405 VALUE, для использования с 6.2.2295 TO;
- имя, не найдено (6.1.0070 ', 6.1.2033 POSTPONE, 6.1.2510 ['], 6.2.2530 [COMPILE]);
- параметры не одного типа (6.1.1240 DO, 6.2.0620 ?DO, 6.2.2440 WITHIN);
- 6.1.2033 POSTPONE, или 6.2.2530 [COMPILE] применено к 6.2.2295 TO;
- строка длиннее, чем строка со счетчиком возвращенная 6.1.2450 WORD;
- u больше или равно числу битов в ячейке (6.1.1805 LSHIFT, 6.1.2162 RSHIFT);
- слово, не определенное с помощью 6.1.1000 CREATE (6.1.0550 >BODY, 6.1.1250 DOES>);
- слова, ненадлежащим образом используемые вне 6.1.0490 <# и 6.1.0040 #> (6.1.0030 #, 6.1.0050 #S, 6.1.1670 HOLD, 6.1.2210 SIGN).

4.1.3 Другая системная документация

Система должна предоставить следующую информацию:

- список нестандартных слов, использующих 6.2.2000 PAD (3.3.3.6 Другие временные области);
- доступные терминальные средства оператора;
- доступная область данных программы, в адресуемых элементах;
- доступный размер стека возвратов, в ячейках;
- доступный размер стека, в ячейках;
- требуемое системное пространство словаря, в адресуемых элементах.

4.2 Программная документация

4.2.1 Зависимости окружения

Программа должна документировать следующие зависимости окружения, где они применяются, и должна документировать другие известные зависимости окружения:

- рассмотрение буфера строки вывода отображаемого числа фиксированной областью с неизменными параметрами доступа (3.3.3.6 Другие временные области);
- зависимости присутствия или отсутствия неграфических символов в полученной строке (6.1.0695 ACCEPT, 6.2.1390 EXPECT);
- доверие к особенностям направления округления (3.2.2.1 Целочисленное деление);
- требования к особенностям представления чисел и арифметики (3.2.1.1 Внутреннее представление числа);
- требование к нестандартным словам или методам (3. Условия применения);
- требование к способности посылать или получать управляющие символы (3.1.2.2 Управляющие символы, 6.1.1750 KEY);
- использование управляющих символов, для выполнения специфических функций (6.1.1320 EMIT, 6.1.2310 TYPE);
- использование флагов как арифметических операндов (3.1.3.1 Флаги);
- использование строчных букв для стандартных имен определений или зависимость от чувствительности к регистру системы (3.3.1.2 Имена определений);
- использование графического символа со значением hex 24 (3.1.2.1 Графические

символы).

4.2.2 Другая программная документация

Программа должна также документировать:

- минимальные требования терминальных средств оператора;
- существует ли Стандартная система после загрузки программы.

5. Соглашение и наименование

5.1 ANS Forth системы

5.1.1 Системное соглашение

Система, которая соблюдает все системные требования разделов 3. Условия применения и 4.1 Системная документация и их подразделов - Стандартная система. Иначе Стандартная система, которая предоставляет только часть основных слов - подмножество Стандартной системы. Иначе Стандартная система (подмножество), которая не в состоянии соблюдать один или большее количество минимальных значений или диапазонов, указанных в 3. Условия применения и их подразделы - имеет ограничения окружающей среды.

5.1.2 Системное наименование

Стандартная система (подмножество) должна быть именована "ANS Forth система (подмножество)". Это наименование не должно применяться к Стандартным системам или Стандартным системам подмножествам, которые имеют ограничения окружающей среды.

Фраза "с ограничениями окружающей среды" должна быть добавлена в конец наименования Стандартной системы (подмножества), которая имеет ограничения окружающей среды.

Фраза "Предоставляет имена из расширений основного набора слов" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет части расширений основного набора слов.

Фраза "Предоставляет расширения основного набора слов" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет все расширения основного набора слов.

5.2 ANS Forth программы

5.2.1 Программное соглашение

Программа, которая соблюдает все программные требования разделов 3. Условия применения и 4.2 Программная документация и их подразделы - Стандартная программа.

5.2.2

Программное наименование

Стандартная программа должна быть именована "ANS Forth программа". Это наименование не должно применяться к Стандартным программам, которые требуют, чтобы система предоставляла стандартные слова вне Основного набора слов или она имеет зависимость от окружения.

Фраза "с Зависимостями от окружения" должна быть добавлена наименования Стандартной программы, которая имеет зависимость от окружения.

Фраза "Требуются имена из расширений основного набора слов" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила часть расширений основного набора.

Фраза "Требуются расширения основного набора слов" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила все расширения основного набора.

6. Словарь

6.1 Основные слова

6.1.0010 ! "store " CORE
(x a-addr --)
Сохраняет x в a-addr .

См.: 3.3.3.1 Выравнивание адреса.

6.1.0030 # "number-sign" CORE
(ud1 -- ud2)
Делит ud1 числом в BASE, дает частное ud2 и остаток n. (n - младший разряд ud1.) Преобразовывает n к внешней форме и добавляет результирующий символ к началу выходной строки отображаемого числа. Неопределенная ситуация существует если # выполняется вне <# #> ограничивающих преобразование числа.
См.: 6.1.0040 #>, 6.1.0050 #S, 6.1.0490 <#.

6.1.0040 #> "number-sign-greater" CORE
(xd -- c-addr u)
Удаляет xd. Делает выходную строку отображаемого числа доступной как символьную строку. C-addr и u определяют результирующую символьную строку. Программа может заменять символы в пределах строки.

См.: 6.1.0030 #, 6.1.0050 #S, 6.1.0490 <#.

6.1.0050 #S "number-sign-s" CORE
(ud1 -- ud2)
Преобразует одну цифру ud1 согласно правилу для #. Продолжает преобразование до нулевого частного. ud2 нулевое. Неопределенная ситуация существует, если #S выполняется вне <# #> ограничивающих преобразование числа.

См.: 6.1.0030 #, 6.1.0040 #>, 6.1.0490 <#.

6.1.0070 ' "tick" CORE
("<spaces>name" -- xt)
Пропускает ведущие разделители пробелы. Выделяет name ограниченное пробелом. Ищет name, и возвращает xt, идентификатор исполнения для

name. Неопределенная ситуация существует, если name не найдено.

При интерпретации, ' xyz EXECUTE, эквивалентно xyz.

См.: 3.4 Интерпретатор текста Forth, 3.4.1 Синтаксический анализ, A.6.1.2033 POSTPONE, A.6.1.2510 ['], D.6.7 Immediate-ности.

6.1.0080 ("paren" CORE
Компиляция: Исполняет семантику выполнения, данную ниже.

Выполнение: ("ccc<paren>" --)
Выделяет ccc, ограниченное) (правая круглая скобка). (- слово немедленного исполнения.

Число символов в ccc может быть нулевое для числа символов в области анализа.

См.: 3.4.1 Синтаксический анализ, 11.6.1.0080 (.

6.1.0090 * "star" CORE
(n1|u1 n2|u2 -- n3|u3)
Умножает n1|u1 на n2|u2, возвращает n3|u3.

6.1.0100 */ "star-slash" CORE
(n1 n2 n3 -- n4)
Умножает n1 на n2, с промежуточным результатом двойная-ячейка d. Делит d на n3, выдает частное одна-ячейка n4. Неопределенная ситуация существует, если n3 нулевое или если частное n4 находится вне диапазона числа со знаком. Если d и n3 отличаются по знаку, возвращенный результат, определенный реализацией будет такой же самый, как и возвращаемый фразой >R M* R> FM/MOD SWAP DROP или фразой >R M* R> SM/REM SWAP DROP.

См.: 3.2.2.1 Целочисленное деление.

6.1.0110 */MOD "star-slash-mod" CORE
(n1 n2 n3 -- n4 n5)
Умножает n1 на n2, с промежуточным результатом двойная-ячейка d. Делит d на n3, выдает остаток одна-ячейка n4 и частное одна-ячейка n5. Неопределенная ситуация существует, если n3 нулевое или если частное n5 находится вне диапазона числа одна-ячейка со знаком. Если d и n3 отличаются по знаку, возвращенный результат, определенный реализацией будет тот же самый, как и возвращаемый фразой >R M* R> FM/MOD или фразой >R M* R> SM/REM.

См.: 3.2.2.1 Целочисленное деление.

6.1.0120 + "plus" CORE
(n1|u1 n2|u2 -- n3|u3)
Прибавляет n2|u2 к n1|u1, возвращая сумму n3|u3.

См.: 3.3.3.1 Выравнивание адреса.

6.1.0130 +! "plus-store" CORE
(n|u a-addr --)
Прибавляет n|u к числу одна-ячейка в a-addr.

См.: 3.3.3.1 Выравнивание адреса.

6.1.0140 +LOOP "plus-loop" CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: do-sys --)
Добавляет семантику времени-выполнения, данную ниже к текущему определению. Разрешает назначения для всех неразрешенных ссылок LEAVE между местоположением, данным do-sys и следующим местоположением для перехода к выполнению слова после +LOOP.

Время-выполнения: (n --) (R: loop-sys1 -- | loop-sys2)
Неопределенная ситуация существует, если параметры управления циклом недоступны. Прибавляет n к индексу цикла. Если индекс цикла не пересекал границу между пределом цикла минус один и пределом цикла, продолжает выполнение с начала цикла. Иначе, удаляет текущие параметры управления цикла и продолжает выполнение сразу после цикла.

См.: 6.1.1240 DO, 6.1.1680 I, 6.1.1760 LEAVE.

6.1.0150 , "comma" CORE
(x --)
Резервирует одну ячейку области данных, и сохраняет x в ячейке. Если указатель области данных выровнен перед выполнением ",", то он останется выровненным после окончания выполнения ",".
Неопределенная ситуация существует если указатель области данных не выровнен перед выполнением ",".

См.: 3.3.3 Область данных, 3.3.3.1 Выравнивание адреса.

6.1.0160 - "minus" CORE
(n1|u1 n2|u2 -- n3|u3)
Вычитает n2|u2 из n1|u1, и возвращает разницу n3|u3.

См.: 3.3.3.1 Выравнивание адреса.

6.1.0180 . "dot" CORE
(n --)
Отображает n в формате свободного поля.

См.: 3.2.1.2 Преобразование цифр, 3.2.1.3 Изображение чисел в свободном поле.

6.1.0190 ." "dot-quote" CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: ("ccc<quote>" --)
Выделяет ccc, ограниченную " (двойная кавычка). Добавляет семантику времени-выполнения данную ниже к текущему определению.

Время-выполнения: (--)
Отображает ccc.

См.: 3.4.1 Синтаксический анализ, 6.2.0200 .(.

6.1.0230 / "slash" CORE

(n1 n2 -- n3)
Делит n1 на n2, возвращает частное одна-ячейка n3. Неопределенная ситуация существует, если n2 нулевое. Если n1 и n2 отличаются по знаку, возвращенный результат, определенный реализацией будет тот

же самый, как и возвращаемый фразой >R S>D R> FM/MOD SWAP DROP или фразой >R S>D R> SM/REM SWAP DROP.

См.: 3.2.2.1 Целочисленное деление.

6.1.0240 /MOD "slash-mod" CORE
(n1 n2 -- n3 n4)
Делит n1 на n2, возвращает остаток одна-ячейка n3 и частное одна-ячейка n4. Неопределенная ситуация существует, если n2 нулевое. Если n1 и n2 отличаются по знаку, возвращенный результат, определенный реализацией будет тот же самый, как и возвращаемый фразой >R S>D R> FM/MOD или фразой >R S>D R> SM/REM.

См.: 3.2.2.1 Целочисленное деление.

6.1.0250 0< "zero-less" CORE
(n -- flag)
flag - true, если и только если n - меньше нуля.

6.1.0270 0= "zero-equals" CORE
(x -- flag)
flag - true, если и только если x - равно нулю.

6.1.0290 1+ "one-plus" CORE
(n1|u1 -- n2|u2)
Прибавляет один(1) к n1|u1 возвращает сумму n2|u2.

6.1.0300 1- "one-minus" CORE
(n1|u1 -- n2|u2)
Вычитает один (1) из n1|u1, возвращает разницу n2|u2.

6.1.0310 2! "two-store" CORE
(x1 x2 a-addr --)
Сохраняет пару-ячеек x1 x2 в a-addr, x2 в a-addr и x1 в следующей последовательной ячейке. Это эквивалентно последовательности SWAP OVER ! CELL+ !.

См.: 3.3.3.1 Выравнивание адреса.

6.1.0320 2* "two-star" CORE
(x1 -- x2)
x2 - результат смещения x1 на один бит к старшему двоичному разряду, заполнение освобожденного младшего бита нулем.

6.1.0330 2/ "two-slash" CORE
(x1 -- x2)
x2 - результат смещения x1 на один бит к младшему двоичному разряду, оставляет старший бит неизменным.

6.1.0350 2@ "two-fetch" CORE
(a-addr -- x1 x2)
Извлекает пару-ячеек x1 x2 сохраненную в a-addr. x2 сохранено в a-addr и x1 в следующей последовательной ячейке. Это эквивалентно последовательности DUP CELL+ @ SWAP @.

См.: 3.3.3.1 Выравнивание адреса, 6.1.0310 2!.

6.1.0370 2DROP "two-drop" CORE
(x1 x2 --)
Удаляет пару-ячеек x1 x2 из стека.

6.1.0380 2DUP "two-dupe" CORE
(x1 x2 -- x1 x2 x1 x2)
Дублирует пару-ячеек x1 x2.

6.1.0400 2OVER "two-over" CORE
(x1 x2 x3 x4 -- x1 x2 x3 x4 x1 x2)
Копирует пару-ячеек x1 x2 на вершину стека.

6.1.0430 2SWAP "two-swap" CORE
(x1 x2 x3 x4 -- x3 x4 x1 x2)
Меняет местами две верхних пары-ячеек.

6.1.0450 : "colon" CORE
(C: "<spaces>name" -- colon-sys)
Пропускает ведущие разделители пробелы. Выделяет name, ограниченное пробелом. Создает определение для name, называемое "определение через двоеточие". Вводит состояние компиляции и начинает текущее определение, создавая colon-sys. Добавляет семантику инициирования, данную ниже к текущему определению.

Семантика выполнения name будет определяться скомпилированными в тело определения словами. Текущее определение не должно быть находимым в словаре, пока оно не закончено (или до выполнения DOES> в некоторых системах).

Инициирование: (i*x -- i*x) (R: -- nest-sys)
Сохраняет зависящую-от-реализации информацию nest-sys о вызове определения. Состояние стека i*x, представляет параметры для name.

name Выполнение: (i*x -- j*x)
Выполняет name определения. Состояние стека i*x, и j*x представляет параметры и результаты из name, соответственно.

См.: 3.4 Интерпретатор текста Forth, 3.4.1 Синтаксический анализ, 3.4.5 Компиляция, 6.1.1250 DOES>, 6.1.2500 [, 6.1.2540], 15.6.2.0470 ;CODE.

6.1.0460 ; "semicolon" CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: colon-sys --)
Добавляет семантику времени-выполнения ниже к текущему определению. Заканчивает текущее определение, позволяя ему быть найденным в словаре, и вводит состояние интерпретации, потребляя colon-sys. Если указатель области данных не выровненный, резервирует достаточно области данных для его выравнивания.

Время-выполнения: (--) (R: nest-sys --)
Возвращение к вызывающему определению, определенному nest-sys.

См.: 3.4 Интерпретатор текста Forth, 3.4.5 Компиляция.

6.1.0480 < "less-than" CORE
(n1 n2 -- flag)
flag - true, если и только если n1 - меньше чем n2.

См.: 6.1.2340 U<.

6.1.0490 <# "less-number-sign" CORE
(--)
Инициализирует процесс выходного преобразования отображаемого

числа.

См.: 6.1.0030 #, 6.1.0040 #>, 6.1.0050 #S.

6.1.0530 = "equals" CORE
(x1 x2 -- flag)
flag - true, если и только если x1 - побитно равно x2.

6.1.0540 > "greater-than" CORE
(n1 n2 -- flag)
flag - true, если и только если n1 - больше чем n2.

См.: 6.2.2350 U>.

6.1.0550 >BODY "to-body" CORE
(xt -- a-addr)
a-addr - адрес области данных соответствующей xt. Неопределенная ситуация существует, если xt не для слова определенного через CREATE.

См.: 3.3.3 Область данных.

6.1.0560 >IN "to-in" CORE
(-- a-addr)
a-addr - адрес ячейки, содержащей смещение в символах от начала входного буфера до начала области анализа.

6.1.0570 >NUMBER "to-number" CORE
(ud1 c-addr1 u1 -- ud2 c-addr2 u2)
ud2 - без знаковый результат преобразования символов из строки указанной c-addr1 u1 в цифры, используя число в BASE, и добавления каждого в ud1 после умножения ud1 на число из BASE.

Преобразование продолжается слева направо до встречи не преобразуемого символа, включая "+" или "-", или пока строка полностью не будет преобразована. c-addr2 - местоположение первого не преобразованного символа или первого символа после конца строки, если строка была полностью преобразована. u2 - число не преобразованных символов в строке. Неопределенная ситуация существует если ud2 переполняется в процессе преобразования.

См.: 3.2.1.2 Преобразование цифр.

6.1.0580 >R "to-r" CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Выполнение: (x --) (R: -- x)
Перемещает x на стек возвратов.

См.: 3.2.3.3 Стек возвратов, 6.1.2060 R>, 6.1.2070 R@, 6.2.0340 2>R, 6.2.0410 2R>, 6.2.0415 2R@.

6.1.0630 ?DUP "question-dupe" CORE
(x -- 0 | x x)
Дублирует x, если оно ненулевое.

6.1.0650 @ "fetch" CORE
(a-addr -- x)
x - значение, сохраненное в a-addr.

См.: 3.3.3.1 Выравнивание адреса.

6.1.0670 ABORT CORE
(i*x --) (R: j*x --)
Освобождает стек данных и выполняет функцию QUIT, которая включает освобождение стека возвратов, без отображения сообщения.

См.: 9.6.2.0670 ABORT.

6.1.0680 ABORT" "abort-quote" CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: ("ccc<quote>" --)
Выделяет ccc, ограниченную " (двойная кавычка). Добавляет семантику времени-выполнения данную ниже к текущему определению.

Время-выполнения: (i*x x1 -- | i*x) (R: j*x -- | j*x)
Удаляет x1 из стека. Если какой либо бит x1 - не нулевой, отображает ccc и выполняет определенную реализацией последовательность сброса, которая включает функцию ABORT.

См.: 3.4.1 Синтаксический анализ, 9.6.2.0680 ABORT".

6.1.0690 ABS "abs" CORE
(n -- u)
u - абсолютное значение n.

6.1.0695 ACCEPT CORE
(c-addr +n1 -- +n2)
Получает строку с максимумом +n1 символов. Неопределенная ситуация существует, если +n1 нулевое или больше чем 32767. Отображает полученные графические символы. Программа, которая зависит от присутствия или отсутствия неграфических символов в строке имеет зависимость от окружающей среды. Функции редактирования, если таковые вообще имеются, которые система исполняет при создании строки - определенное реализацией.

Ввод заканчивается, когда получен определенный реализацией признак конца строки. Когда ввод заканчивается, ничего не добавляется в конец строки, и отображение обеспечивается определенным реализацией способом.

+n2 - длина строки, сохраненной в c-addr.

6.1.0705 ALIGN CORE
(--)
Если указатель области данных не выровнен, резервирует достаточно пространства, для его выравнивания.

См.: 3.3.3 Область данных, 3.3.3.1 Выравнивание адреса.

6.1.0706 ALIGNED CORE
(addr -- a-addr)
a-addr - первый выровненный адрес большее чем или равный addr.

См.: 3.3.3.1 Выравнивание адреса.

6.1.0710 ALLOT CORE
(n --)
Если n больше нуля, резервирует n адресуемых элементов области данных. Если n меньше нуля, освобождает | n | адресуемых элементов области данных. Если n нуль, оставляет указатель области данных

неизменным.

Если указатель области данных выровнен, и n - кратно размеру ячейки перед выполнением ALLOT, он останется выровненным, и после завершения выполнения ALLOT.

Если указатель области данных выровнен на символ, и n - кратно размеру символа перед выполнением ALLOT, он останется выровненным на символ, и после завершения выполнения ALLOT.

См.: 3.3.3 Область данных.

6.1.0720 AND CORE
(x1 x2 -- x3)
x3 поразрядное логическое "и" x1 и x2.

6.1.0750 BASE CORE
(-- a-addr)
a-addr - адрес ячейки, содержащей текущее основание системы счисления для преобразования чисел {{2...36}}.

6.1.0760 BEGIN CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: -- dest)
Помещает адрес следующей ячейки для передачи управления, dest, на стек потока-управления. Добавляет семантику времени-выполнения, данную ниже к текущему определению.

Время-выполнения: (--)
Продолжает выполнение.

См.: 3.2.3.2 Стек потока-управления, 6.1.2140 REPEAT, 6.1.2390 UNTIL, 6.1.2430 WHILE.

6.1.0770 BL "b-l" CORE
(-- char)
char - символьное значение для пробела.

6.1.0850 C! "c-store" CORE
(char c-addr --)
Сохраняет char в c-addr. Когда размер символа меньше чем размер ячейки, переданы только младшие биты, соответствующие размеру символа.

См.: 3.3.3.1 Выравнивание адреса.

6.1.0860 C, "c-comma" CORE
(char --)
Резервирует пространство для одного символа в области данных и запоминает char в нем. Если указатель области данных - выровнен на символ перед выполнением C,, он останется выровненным на символ и после окончания выполнения C,. Неопределенная ситуация существует, если указатель области данных - не выровненный на символ до выполнения C,.

См.: 3.3.3 Область данных, 3.3.3.1 Выравнивание адреса.

6.1.0870 C@ "c-fetch" CORE
(c-addr -- char)
Выбирает символ, сохраненный в c-addr. Когда размер ячейки больше

чем размер символа, все неиспользованные старшие биты - нули.

См.: 3.3.3.1 Выравнивание адреса.

6.1.0880 CELL+ "cell-plus" CORE
(a-addr1 -- a-addr2)

Добавляет размер в адресуемых элементах ячеек к a-addr1, возвращает a-addr2.

См.: 3.3.3.1 Выравнивание адреса.

6.1.0890 CELLS CORE
(n1 -- n2)
n2 - размер в адресуемых элементах n1 ячеек.

6.1.0895 CHAR "char" CORE
("<spaces>name" -- char)
Пропускает ведущие разделители пробелы. Выделяет name, ограниченное пробелом. Помещает значение его первого символа на стек.

См.: 3.4.1 Синтаксический анализ, 6.1.2520 [CHAR].

6.1.0897 CHAR+ "char-plus" CORE
(c-addr1 -- c-addr2)
Добавляет размер в адресуемых элементах символа к c-addr1, возвращает c-addr2.

См.: 3.3.3.1 Выравнивание адреса.

6.1.0898 CHARS "chars" CORE
(n1 -- n2)
n2 - размер в адресуемых элементах n1 символов.

6.1.0950 CONSTANT CORE
(x "<spaces>name" --)
Пропускает ведущие разделители пробелы. Выделяет name, ограниченное пробелом. Создает определение для name с семантикой выполнения, определенной ниже.

name объявляется как "constant".

name Выполнение: (-- x)
Размещает x на стеке.

См.: 3.4.1 Синтаксический анализ.

6.1.0980 COUNT CORE
(c-addr1 -- c-addr2 u)
Возвращает спецификацию символьной строки для строки со счетчиком сохраненной в c-addr1. c-addr2 - адрес первого символа после c-addr1. u - содержимое символа в c-addr1, который является длиной в символах строки в c-addr2.

6.1.0990 CR "c-r" CORE
(--)
Заставляет последующий вывод появляться в начале следующей строки.

6.1.1000 CREATE CORE
("<spaces>name" --)

Пропускает ведущие разделители пробелы. Выделяет name, ограниченное пробелом. Создает определение для name с семантикой выполнения, определенной ниже. Если указатель области данных не выровнен, резервирует достаточно области данных для его выравнивания. Новый указатель области данных определяет поле данных name. CREATE не распределяет область данных в поле данных name.

name Выполнение: (-- a-addr)
a-addr - адрес поля данных name. Семантика выполнения name может быть расширена использованием DOES>.

См.: 3.3.3 Область данных, 6.1.1250 DOES>.

6.1.1170 DECIMAL CORE
(--)
Устанавливает основание системы счисления преобразования чисел на десять (десятичные числа).

6.1.1200 DEPTH CORE
(-- +n)
+n - число значений данных одна-ячейка, содержащихся на стеке данных перед тем, как +n было помещено на стек.

6.1.1240 DO CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: -- do-sys)
Помещает do-sys на стек потока управления. Добавляет семантику времени-выполнения данную ниже к текущему определению. Семантика не завершена, пока do-sys не разрешена потребителем типа LOOP.

Время-выполнения: (n1|u1 n2|u2 --) (R: -- loop-sys)
Устанавливает параметры управления цикла с индексом n2|u2, и границей n1|u1. Неопределенная ситуация существует, если оба n1|u1 и n2|u2- не одного типа. Что-либо уже имеющееся на стеке возвратов становится недоступным до удаления параметров управления циклом.

См.: 3.2.3.2 Стек потока управления, 6.1.0140 +LOOP, 6.1.1800 LOOP.

6.1.1250 DOES> "does" CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: colon-sys1 -- colon-sys2)
Добавляет семантику времени-выполнения ниже к текущему определению. В любом случае текущее определение представленное находимым в словаре при компиляции DOES> - определенное реализацией. Потребляет colon-sys1 и производит colon-sys2. Добавляет семантику инициирования, данную ниже к текущему определению.

Время-выполнения: (--) (R: nest-sys1 --)
Заменяет семантику выполнения самого последнего определения, упоминаемого как name, семантикой выполнения имени данной ниже. Возвращает управление на вызывающее определение, определенное nest-sys1. Неопределенная ситуация существует если name не было определено CREATE, или определенным пользователем словом которое вызывает CREATE.

Инициирование: (i*x -- i*x a-addr) (R: -- nest-sys2)

Сохраняет зависящую-от-реализации информацию nest-sys2 о вызывающем определении. Размещает адрес поля данных name на стеке. Состояние стека i*x представляет параметры name.

name Выполнение: (i*x -- j*x)
Выполняет часть определения, которая начинается с семантики инициирования добавленной изменившим name DOES>. Состояния стека i*x и j*x представляют параметры, и результаты name, соответственно.

См.: 6.1.1000 CREATE.

6.1.1260 DROP CORE
(x --)
Удаляет x из стека.

6.1.1290 DUP "dupe" CORE
(x -- x x)
Дублирует x.

6.1.1310 ELSE CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: orig1 -- orig2)
Помещает адрес ячейки новой неразрешенной ссылки вперед orig2 на стек потока управления. Добавляет семантику времени-выполнения данную ниже к текущему определению. Семантика будет не завершена, пока orig2 не разрешена (например, THEN). Разрешает ссылку вперед orig1 использованием адреса ячейки после добавленной семантики времени-выполнения.

Время-выполнения: (--)
Продолжает выполнение с адреса ячейки данного ссылкой orig2.

См.: 6.1.1700 IF, 6.1.2270 THEN.

6.1.1320 EMIT CORE
(x --)
Если x - графический символ в наборе символов определенном реализацией - показать x. Эффект EMIT для всех других значений x - определенное реализацией.

Когда просматривается символ, чьи символом-определенные биты имеют значение между hex 20 и 7E включительно, отображается соответствующий стандартный символ, определенный 3.1.2.1 Графические символы. Поскольку различные устройства вывода могут реагировать по-разному на управляющие символы, программы, использующие управляющие символы для выполнения специфических функций, имеют зависимость от окружения. Каждый EMIT, имеет дело только с одним символом.

См.: 6.1.2310 TYPE.

6.1.1345 ENVIRONMENT? "environment-query" CORE
(c-addr u -- false | i*x true)
c-addr - адрес символьной строки, и u - ее размер. u может иметь значение в диапазоне от нуля до максимума определенного реализацией, но не должен быть меньше чем 31. Символьная строка должна содержать ключевое слово из 3.2.6 Запросы к окружению или дополнительных наборов слов, которое будут проверено на соответствие атрибутам существующей среды. Если система

рассматривает атрибут как неизвестный, возвращается флаг - false; иначе, флаг - true, и возвращенное i*x имеет тип, указанный в таблице для запрашиваемого атрибута.

6.1.1360 EVALUATE CORE
(i*x c-addr u -- j*x)
Сохраняет текущую спецификацию входного источника. Записывает минус - один (-1) в SOURCE-ID, если он присутствует. Делает строку, описанную c-addr и u входным источником и входным буфером, устанавливает >IN в нуль, и интерпретирует. Когда область анализа пуста, восстанавливает предыдущую спецификацию входного источника. Изменения состояния стека определяются интерпретируемыми словами.

6.1.1370 EXECUTE CORE
(i*x xt -- j*x)
Удаляет xt из стека, и выполняет идентифицированную им семантику. Изменения состояния стека определяются выполненным словом.

См.: 6.1.0070 ', 6.1.2510 ['].

6.1.1380 EXIT CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Выполнение: (--) (R: nest-sys --)
Возвращает управление на вызывающее определение, определенное nest-sys. Перед выполнением EXIT в пределах do-loop, программа должна удалить параметры управления циклом, выполняя UNLOOP.

См.: 3.2.3.3 Стек возвратов, 6.1.2380 UNLOOP.

6.1.1540 FILL CORE
(c-addr u char --)
Если u больше чем нуль, сохраняет char в каждом из u последовательных символов в памяти, начинающейся с c-addr.

6.1.1550 FIND CORE
(c-addr -- c-addr 0 | xt 1 | xt -1)
Находит определение, по имени строки со счетчиком в c-addr. Если определение не найдено, возвращает c-addr и нуль. Если определение найдено, возвращает его идентификатор исполнения xt. Если определение немедленного исполнения, возвращает также один (1), иначе минус - один (-1). Для данной строки значения возвращенные FIND, во время компиляции могут отличаться от возвращенных при не компиляции.

См.: 3.4.2 Поиск имен определений, A.6.1.0070 ', A.6.1.2510 ['], A.6.1.2033 POSTPONE, D.6.7 Immediate-ности.

6.1.1561 FM/MOD "f-m-slash-mod" CORE
(d1 n1 -- n2 n3)
Делит d1 на n1, возвращает минимальное частное n3 и остаток n2. Входные и выходные аргументы стека знаковые. Неопределенная ситуация существует, если n1 нулевое или если частное находится вне диапазона целого числа одна-ячейка со знаком.

См.: 3.2.2.1 Целочисленное деление, 6.1.2214 SM/REM, 6.1.2370 UM/MOD.

6.1.1650 HERE CORE
(-- addr)
addr - указатель области данных.

См.: 3.3.3.2 Непрерывные области.

6.1.1670 HOLD CORE
(char --)
Добавляет char к началу выходной строки отображаемого числа. Неопределенная ситуация существует если HOLD выполняется вне <# #> ограничивающих преобразование числа.

6.1.1680 I CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Выполнение: (-- n|u) (R: loop-sys -- loop-sys)
n|u - копия текущего (самого внутреннего) индекса цикла. Неопределенная ситуация существует, если параметры управления циклом недоступны.

6.1.1700 IF CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: -- orig)
Помещает адрес ячейки новой неразрешенной ссылки вперед orig на стек потока управления. Добавляет семантику времени-выполнения, данную ниже к текущему определению. Семантика не завершена, пока orig не разрешена, например THEN или ELSE.

Время-выполнения: (x --)
Если все биты x нулевые, продолжает выполнение с адреса ячейки определенного ссылкой orig.

См.: 3.2.3.2 Стек потока-управления, 6.1.1310 ELSE, 6.1.2270 THEN.

6.1.1710 IMMEDIATE CORE
(--)
Делает самое последнее определение словом немедленного исполнения. Неопределенная ситуация существует, если самое последнее определение не имеет имени.

См.: D.6.7 Immediate-ности.

6.1.1720 INVERT CORE
(x1 -- x2)
Инвертирует все биты x1, возвращает его логическую инверсию x2.

См.: 6.1.1910 NEGATE, 6.1.0270 0=.

6.1.1730 J CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Выполнение:(-- n|u)(R: loop-sys1 loop-sys2 -- loop-sys1 loop-sys2)
n|u - копия следующего-внешнего индекса цикла. Неопределенная ситуация существует, если параметры управления цикла следующего-внешнего цикла, loop-sys1, недоступны.

6.1.1750 KEY CORE
(-- char)
Получает один символ char, член определенного реализацией набора символов. События клавиатуры, которые не соответствуют таким символам игнорируются, пока не получен допустимый символ, и впоследствии будут недоступны.

Могут быть получены все стандартные символы. Символы, полученные

KEY, не отображаются.

Любой стандартный символ, возвращенный KEY имеет числовое значение определенное в 3.1.2.1 Графические символы. Программы, которые требуют способности к получению управляющих символов, имеют зависимость от окружения.

См.: 10.6.2.1307 EKEY, 10.6.1.1755 KEY?.

6.1.1760 LEAVE CORE

Интерпретация: Семантика интерпретации для этого слова не определена.

Выполнение: (--) (R: loop-sys --)
Снимает текущие параметры управления циклом. Неопределенная ситуация существует, если они недоступны. Продолжает выполнение непосредственно вне самого внутреннего синтаксически окруженного DO ... LOOP или DO ... +LOOP.

См.: 3.2.3.3 Стек возвратов, 6.1.0140 +LOOP, 6.1.1800 LOOP.

6.1.1780 LITERAL CORE

Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (x --)
Добавляет семантику времени-выполнения, данную ниже к текущему определению.

Время-выполнения: (-- x)
Помещает x на стек.

6.1.1800 LOOP CORE

Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: do-sys --)
Добавляет семантику времени-выполнения, данную ниже к текущему определению. Разрешает назначения для всех неразрешенных ссылок LEAVE между адресом ячейки определенным do-sys и следующим за LOOP адресом ячейки для передачи управления к выполнению слова после LOOP.

Время-выполнения: (--) (R: loop-sys1 -- | loop-sys2)
Неопределенная ситуация существует, если параметры управления цикла недоступны. Добавляет один к индексу цикла. Если после этого индекс цикла равен пределу цикла, снимает параметры цикла, и продолжает выполнение непосредственно после цикла. Иначе продолжает выполнение с начала цикла.

См.: 6.1.1240 DO, 6.1.1680 I, 6.1.1760 LEAVE.

6.1.1805 LSHIFT "l-shift" CORE
(x1 u -- x2)
Выполняет логический левый сдвиг x1 на u двоичных разрядов и возвращает x2. Помещает нули в самые младшие биты, освобожденные сдвигом. Неопределенная ситуация существует, если u больше или равно числу битов в ячейке.

6.1.1810 M* "m-star" CORE
(n1 n2 -- d)
d - знаковое произведение n1 на n2.

6.1.1870 MAX CORE

(n1 n2 -- n3)
n3 большее из n1 и n2.

6.1.1880 MIN CORE
(n1 n2 -- n3)
n3 меньшее из n1 и n2.

6.1.1890 MOD CORE
(n1 n2 -- n3)
Делит n1 на n2, возвращает остаток n3 одна-ячейка. Неопределенная ситуация существует, если n2 нулевое. Если n1 и n2 отличаются по знаку, возвращенный результат определенный реализацией будет такой же самый как и возвращенный фразой >R S>D R> FM/MOD DROP или фразой >R S>D R> SM/REM DROP.

См.: 3.2.2.1 Целочисленное деление.

6.1.1900 MOVE CORE
(addr1 addr2 u --)
Если u больше чем нуль, копирует содержимое u последовательных адресуемых элементов из addr1 в u последовательных адресуемых элементов в addr2. После завершения MOVE, u последовательных адресуемых элементов в addr2 содержат тоже, что u последовательных адресуемых элементов в addr1 содержали перед перемещением.

См.: 17.6.1.0910 CMOVE, 17.6.1.0920 CMOVE>.

6.1.1910 NEGATE CORE
(n1 -- n2)
Отрицание n1, возвращает его арифметическую инверсию n2.

См.: 6.1.1720 INVERT, 6.1.0270 0=.

6.1.1980 OR CORE
(x1 x2 -- x3)
x3 - поразрядное включающее или x1 и x2.

6.1.1990 OVER CORE
(x1 x2 -- x1 x2 x1)
Помещает копию x1 на вершине стека.

6.1.2033 POSTPONE CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: ("<spaces>name" --)
Пропускает ведущие разделители пробелы. Выделяет name, ограниченное пробелом. Находит name. Добавляет семантику компиляции name в текущее определение. Неопределенная ситуация существует, если name не найдено.

См.: 3.4.1 Синтаксический анализ.

6.1.2050 QUIT CORE
(--) (R: i*x --)
Освобождает стек возвратов, сохраняет нуль в SOURCE-ID, если он присутствует, делает пользовательское устройство ввода входным источником, и вводит состояние интерпретации. Не отображает сообщение. Повторяет следующее:

- Принимает строку из входного источника во входной буфер, обнуляет >IN, и интерпретирует.

- Отображает определенную реализацией системную подсказку, если в состоянии интерпретации вся обработка была закончена, и нет неопределенной ситуации.

См.: 3.4 Интерпретатор текста Forth.

6.1.2060 R> "r-from" CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Выполнение: (-- x) (R: x --)
Перемещает x из стека возвратов на стек данных.

См.: 3.2.3.3 Стек возвратов, 6.1.0580 >R, 6.1.2070 R@, 6.2.0340 2>R, 6.2.0410 2R>, 6.2.0415 2R@.

6.1.2070 R@ "r-fetch" CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Выполнение: (-- x) (R: x -- x)
Копирует x из стека возвратов на стек данных.

См.: 3.2.3.3 Стек возвратов, 6.1.0580 >R, 6.1.2060 R>, 6.2.0340 2>R, 6.2.0410 2R>, 6.2.0415 2R@.

6.1.2120 RECURSE CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (--)
Добавляет семантику выполнения текущего определения в текущее определение. Неопределенная ситуация существует, если RECURSE появляется в определении после DOES>.

См.: 6.1.1250 DOES>, 6.1.2120 RECURSE.

6.1.2140 REPEAT CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: orig dest --)
Добавляет семантику времени-выполнения, данную ниже к текущему определению, разрешает ссылку назад dest. Разрешает ссылку вперед orig используя адрес ячейки после добавленной семантики времени-выполнения.

Время-выполнения: (--)
Продолжает выполнение с адреса ячейки, данного dest.

См.: 6.1.0760 BEGIN, 6.1.2430 WHILE.

6.1.2160 ROT "rote" CORE
(x1 x2 x3 -- x2 x3 x1)
Вращает три верхних элемента стека.

6.1.2162 RSHIFT "r-shift" CORE
(x1 u -- x2)
Выполняет правый логический сдвиг x1 на u двоичных разрядов, возвращает x2. Помещает нули в самые старшие биты, освобожденные сдвигом. Неопределенная ситуация существует, если u больше или равно числу бит в ячейке.

6.1.2165 S" "s-quote" CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: ("ccc<quote>" --)
Выделяет ccc, ограниченную " (двойная кавычка). Добавляет семантику времени-выполнения данную ниже к текущему определению.

Время-выполнения: (-- c-addr u)
Возвращает c-addr и u описывающие строку, состоящую из символов ccc. Программа не должна изменять возвращенную строку.

См.: 3.4.1 Синтаксический анализ, 6.2.0855 C", 11.6.1.2165 S".

6.1.2170 S>D "s-to-d" CORE
(n -- d)
Преобразовывает число n в число d две-ячейки с тем же самым числовым значением.

6.1.2210 SIGN CORE
(n --)
Если n - отрицательное, добавляет знак "минус" в начало выходной строки отображаемого числа. Неопределенная ситуация существует, если SIGN выполняется вне <# #> ограничивающих преобразование числа.

6.1.2214 SM/REM "s-m-slash-rem" CORE
(d1 n1 -- n2 n3)
Делит d1 на n1, возвращает симметричное частное n3 и остаток n2. Входные и выходные параметры стека знаковые. Неопределенная ситуация существует, если n1 нулевое или если частное находится вне диапазона целого числа со знаком одна-ячейка.

См.: 3.2.2.1 Целочисленное деление, 6.1.1561 FM/MOD, 6.1.2370 UM/MOD.

6.1.2216 SOURCE CORE
(-- c-addr u)
c-addr - адрес, и u - число символов во входном буфере.

6.1.2220 SPACE CORE
(--)
Отображает один пробел.

6.1.2230 SPACES CORE
(n --)
Если n больше нуля, отображает n пробелов.

6.1.2250 STATE CORE
(-- a-addr)
a-addr - адрес ячейки, содержащей флаг состояния компиляции. STATE - true когда в состояния компиляции, иначе false. Значение true в STATE ненулевое, но - иначе определенное реализацией. Только следующие стандартные слова изменяют значение в STATE: : (двоеточие), ; (Точка с запятой), ABORT, QUIT, :NONAME, [(левая скобка), и] (правая скобка).

Обратите внимание:
Программа непосредственно не должна изменять содержание STATE.

См.: 3.4 Интерпретатор текста Forth, 6.1.0450 :, 6.1.0460 ;, 6.1.0670 ABORT, 6.1.2050 QUIT, 6.1.2500 [, 6.1.2540], 6.2.0455 :NONAME,

15.6.2.2250 STATE.

6.1.2260 SWAP CORE
(x1 x2 -- x2 x1)
Переставляет два верхних элемента стека.

6.1.2270 THEN CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: orig --)
Добавляет семантику времени-выполнения, данную ниже к текущему определению. Разрешает ссылку вперед orig, используя адрес ячейки добавленной семантики времени-выполнения.

Время-выполнения: (--)
Продолжает выполнение.

См.: 6.1.1310 ELSE, 6.1.1700 IF.

6.1.2310 TYPE CORE
(c-addr u --)
Если u больше нуля, отображает символьную строку, определенную c-addr и u.

Когда просматривается символ, чьи символом-определенные биты имеют значение между hex 20 и 7E включительно, отображается соответствующий стандартный символ, определенный в 3.1.2.1 Графические символы. Поскольку различные устройства вывода могут реагировать по-разному на управляющие символы, программы, использующие управляющие символы для выполнения специфических функций, имеют зависимость от окружения.

См.: 6.1.1320 EMIT.

6.1.2320 U. "u-dot" CORE
(u --)
Отображает u в формате свободного поля.

6.1.2340 U< "u-less-than" CORE
(u1 u2 -- flag)
flag - true, если и только если u1 - меньше чем u2.

См.: 6.1.0480 <.

6.1.2360 UM* "u-m-star" CORE
(u1 u2 -- ud)
Умножает u1 на u2, возвращает результат ud без знака две-ячейки. Все значения и арифметика без знаковые.

6.1.2370 UM/MOD "u-m-slash-mod" CORE
(ud u1 -- u2 u3)
Делит ud на u1, возвращает частное u3 и остаток u2. Все значения и арифметика без знака. Неопределенная ситуация существует если u1 нулевое или если частное находится вне диапазона целого числа одна-ячейка без знака.

См.: 3.2.2.1 Целочисленное деление, 6.1.1561 FM/MOD, 6.1.2214 SM/REM.

6.1.2380 UNLOOP CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Выполнение: (--) (R: loop-sys --)
Снимает параметры управления циклом для текущего уровня вложенности. UNLOOP требуется для каждого уровня вложенности прежде, чем может быть выполнено EXIT определения. Неопределенная ситуация существует если параметры управления циклом недоступны.

См.: 3.2.3.3 Стек возвратов.

6.1.2390 UNTIL CORE

Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: dest --)
Добавляет семантику времени-выполнения, данную ниже к текущему определению, разрешая ссылку назад dest.

Время-выполнения: (x --)
Если все биты x нулевые, продолжает выполнение с адреса ячейки определенного dest.

См.: 6.1.0760 BEGIN.

6.1.2410 VARIABLE CORE

("<spaces>name" --)
Пропускает ведущие разделители пробелы. Выделяет name, ограниченное пробелом. Создает определение для name с семантикой выполнения, определенной ниже. Резервирует одну ячейку области данных в выровненном адресе.

name объявляется как "variable".

name Выполнение: (-- a-addr)
a-addr - адрес зарезервированной ячейки. Программа ответственна за инициализацию и содержание зарезервированной ячейки.

См.: 3.4.1 Синтаксический анализ.

6.1.2430 WHILE CORE

Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: dest -- orig dest)
Помещает адрес ячейки новой неразрешенной ссылки вперед orig на стек потока управления, под существующим dest. Добавляет семантику времени-выполнения, данную ниже к текущему определению. Семантика не завершена до разрешения orig и dest (например, REPEAT).

Время-выполнения: (x --)
Если все биты x нулевые, продолжает выполнение с адреса ячейки определенного ссылкой orig.

6.1.2450 WORD CORE

(char "<chars>ccc<char>" -- c-addr)
Пропускает ведущие разделители. Выделяет символы ccc ограниченные char. Неопределенная ситуация существует, если длина выделенной строки больше чем длина определенной реализацией строки со счетчиком.

c-addr - адрес временной области, содержащей выделенное слово как строку со счетчиком. Если область анализа была пуста или не содержала других символов кроме разделителей, результирующая строка имеет нулевую длину. За строкой следует не включенный в длину пробел. Программа может изменять символы в пределах строки.

Примечание: Требование, о следовании пробела за строкой устаревшее и включено как уступка существующим программам, которые используют CONVERT. Программа не должна зависеть от существования пробела.

См.: 3.3.3.6 Другие временные области, 3.4.1 Синтаксический анализ.

6.1.2490 XOR "x-or" CORE
(x1 x2 -- x3)
x3 поразрядное исключающее-или x1 и x2.

6.1.2500 ["left-bracket" CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: Исполняет семантику выполнения, данную ниже.

Выполнение: (--)
Вводит состояние интерпретации. [- слово немедленного выполнения.

См.: 3.4 Интерпретатор текста Forth, 3.4.5 Компиляция, 6.1.2540].

6.1.2510 ['] "bracket-tick" CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: ("<spaces>name" --)
Пропускает ведущие разделители пробелы. Выделяет name, ограниченное пробелом. Находит name. Добавляет семантику времени-выполнения, данную ниже к текущему определению.
Неопределенная ситуация существует, если name не найдено.

Время-выполнения: (-- xt)
Помещает идентификатор исполнения name xt на стек. Идентификатор исполнения, возвращенный скомпилированной фразой "['] X " , имеет то же значение, как и возвращенный "' X " вне состояния компиляции.

См.: 3.4.1 Синтаксический анализ, A.6.1.0070 ', A.6.1.2033 POSTPONE, D.6.7 Immediate-ности.

6.1.2520 [CHAR] "bracket-char" CORE
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: ("<spaces>name" --)
Пропускает ведущие разделители пробелы. Выделяет name, ограниченное пробелом. Добавляет семантику времени-выполнения, данную ниже к текущему определению.

Время-выполнения: (-- char)
Помещает на стек char, значение первого символа name.

См.: 3.4.1 Синтаксический анализ, 6.1.0895 CHAR.

6.1.2540] "right-bracket" CORE
(--)
Вводит состояние компиляции.

См.: 3.4 Интерпретатор текста Forth, 3.4.5 Компиляция, 6.1.2500 [.

6.2 _____

6.2.0060 #TIB "number-t-i-b" CORE EXT
(-- a-addr)
a-addr - адрес ячейки, содержащей число символов во входном буфере терминала.

Примечание: Это слово устаревшее и включено как уступка существующим реализациям.

6.2.0200 .("dot-paren" CORE EXT
Компиляция: Исполняет семантику выполнения, данную ниже.

Выполнение: ("ccc<paren>" --)
Выделяет и отображает ccc, ограниченную) (правая круглая скобка). .(является словом немедленного исполнения.

См.: 3.4.1 Синтаксический анализ, 6.1.0190 .".

6.2.0210 .R "dot-r" CORE EXT
(n1 n2 --)
Отображает n1 выровненное вправо в поле шириной n2 символов. Если число символов требуемых для отображения n1 больше чем n2, все цифры отображаются без ведущих пробелов в поле необходимой ширины.

6.2.0260 0<> "zero-not-equals" CORE EXT
(x -- flag)
flag - true, если и только если x - не равно нулю.

6.2.0280 0> "zero-greater" CORE EXT
(n -- flag)
flag - true, если и только если n - больше нуля.

6.2.0340 2>R "two-to-r" CORE EXT
Интерпретация: Семантика интерпретации для этого слова не определена.

Выполнение: (x1 x2 --) (R: -- x1 x2)
Перемещает пару-ячеек x1 x2 на стек возвратов. Семантически эквивалентно SWAP >R >R.

См.: 3.2.3.3 Стек возвратов, 6.1.0580 >R, 6.1.2060 R>, 6.1.2070 R@, 6.2.0410 2R>, 6.2.0415 2R@.

6.2.0410 2R> "two-r-from" CORE EXT
Интерпретация: Семантика интерпретации для этого слова не определена.

Выполнение: (-- x1 x2) (R: x1 x2 --)
Перемещает пару-ячеек x1 x2 со стека возвратов. Семантически эквивалентно R> R> SWAP.

См.: 3.2.3.3 Стек возвратов, 6.1.0580 >R, 6.1.2060 R>, 6.1.2070 R@, 6.2.0340 2>R, 6.2.0415 2R@.

6.2.0415 2R@ "two-r-fetch" CORE EXT
Интерпретация: Семантика интерпретации для этого слова не определена.

Выполнение: (-- x1 x2) (R: x1 x2 -- x1 x2)
Копирует пару-ячеек x1 x2 со стека возвратов. Семантически эквивалентно R> R> 2DUP >R >R SWAP.

См.: 3.2.3.3 Стек возвратов, 6.1.0580 >R, 6.1.2060 R>, 6.1.2070 R@, 6.2.0340 2>R, 6.2.0410 2R>.

6.2.0455 :NONAME "colon-no-name" CORE EXT
(C: -- colon-sys) (S: -- xt)
Создает идентификатор исполнения xt, вводит состояние компиляции и начинает текущее определение, производя colon-sys. Добавляет семантику инициирования данную ниже к текущему определению.

Семантика выполнения xt будет определена скомпилированными в тело определения словами. Это определение может быть выполнено позже, используя xt EXECUTE.

Если стек потока управления реализован с использованием стека данных, colon-sys должен быть самым верхним элементом на стеке данных.

Инициирование: (i*x -- i*x) (R: -- nest-sys)
Сохраняет зависящую-от-реализации информацию nest-sys о вызывающем определении. Состояние стека i*x представляет параметры для xt.

xt Выполнение: (i*x -- j*x)
Выполняет определение определенное xt. Состояние стека i*x и i*x представляет параметры для и результаты от xt, соответственно.

См. 3.2.3.2 Стек потока-управления.

6.2.0500 <> "not-equals" CORE EXT
(x1 x2 -- flag)
flag - true, если и только если x1 побитно не эквивалентно x2.

6.2.0620 ?DO "question-do" CORE EXT
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: -- do-sys)
Помещает do-sys на стек потока управления. Добавляет семантику времени-выполнения данную ниже к текущему определению. Семантика не завершена, пока не разрешена потребителем do-sys типа LOOP.

Время-выполнения: (n1|u1 n2|u2 --) (R: -- | loop-sys)

Если n1|u1 равно n2|u2, продолжает выполнение с адреса ячейки данного потребителем do-sys. Иначе устанавливает параметры управления цикла с индексом n2|u2, и границей n1|u1, и продолжает выполнение непосредственно после ?DO. Что - либо уже имеющееся на стеке возвратов становится недоступным до удаления параметров управления циклом. Неопределенная ситуация существует, если оба n1|u1 и n2|u2- не одного типа.

См.: 3.2.3.2 Стек потока-управления, 6.1.0140 +LOOP, 6.1.1240 DO, 6.1.1680 I, 6.1.1760 LEAVE, 6.1.1800 LOOP, 6.1.2380 UNLOOP.

6.2.0700 AGAIN CORE EXT
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: dest --)
Добавляет семантику времени-выполнения, данную ниже к текущему определению, разрешением ссылки назад dest.

Время-выполнения: (--)
Продолжает выполнение с адреса ячейки определенного dest. Если

никакое другое слово потока управления не используется, любой код программы после AGAIN не будет выполнен.

См.: 6.1.0760 BEGIN.

6.2.0855 C" "c-quote" CORE EXT
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: ("ccc<quote>" --)
Выделяет ccc, ограниченное " (двойная кавычка), и добавляет семантику времени-выполнения данную ниже к текущему определению.

Время-выполнения: (-- c-addr)
Возвращает c-addr строку со счетчиком, состоящую из символов ccc. Программа не должна изменять возвращенную строку.

См.: 3.4.1 Синтаксический анализ, 6.1.2165 S", 11.6.1.2165 S".

6.2.0873 CASE CORE EXT
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: -- case-sys)
Отмечает начало CASE ... OF ... ENDOF ... ENDCASE структуры. Добавляет семантику времени-выполнениия, данную ниже к текущему определению.

Время-выполнения: (--)
Продолжает выполнение.

См.: 6.2.1342 ENDCASE, 6.2.1343 ENDOF, 6.2.1950 OF.

6.2.0945 COMPILE, "compile-comma" CORE EXT
Интерпретация: Семантика интерпретации для этого слова не определена.

Выполнение: (xt --)
Добавляет семантику выполнения определения представленного xt к семантике выполнения текущего определения.

6.2.0970 CONVERT CORE EXT
(ud1 c-addr1 -- ud2 c-addr2)
ud2 - результат преобразования символов в пределах текста начинающегося с первого символа после c-addr1 в цифры, используя число из BASE, и добавление каждой цифры в ud1 после умножения ud1 на число из BASE. Преобразование продолжается до встречи с не преобразуемым символом. c-addr2 - адрес ячейки первого не преобразованного символа. Неопределенная ситуация существует, если ud2 переполняется.

Примечание: Это слово устаревшее и включено как уступка существующим реализациям. Его функция заменена на 6.1.0570 >NUMBER.

См.: 3.2.1.2 Преобразование цифр.

6.2.1342 ENDCASE "end-case" CORE EXT
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: case-sys --)
Отмечает конец CASE ... OF ... ENDOF ... ENDCASE структуры. Использует case-sys для разрешения всей структуры. Добавляет семантику времени-выполнения данную ниже к текущему определению.

Время-выполнения: (x --)
Снимает case селектор x, и продолжает выполнение.

См.: 6.2.0873 CASE, 6.2.1343 ENDOF, 6.2.1950 OF.

6.2.1343 ENDOF "end-of" CORE EXT
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: case-sys1 of-sys -- case-sys2)
Отмечает конец OF ... ENDOF части CASE структуры. Следующий адрес ячейки для передачи управления разрешает ссылка of-sys. Добавляет семантику времени-выполнения, данную ниже к текущему определению. Заменяет case-sys1 на case-sys2 на стеке потока управления, для разрешения ENDCASE.

Время-выполнения: (--)
Продолжает выполнение с адреса ячейки определенного потребителем case-sys2.

См.: 6.2.0873 CASE, 6.2.1342 ENDCASE, 6.2.1950 OF.

6.2.1350 ERASE CORE EXT
(addr u --)
Если u больше нуля, очистить все биты в каждом из u последовательных адресуемых элементов памяти, начинающихся с addr.

6.2.1390 EXPECT CORE EXT
(c-addr +n --)
Получает строку с максимумом в +n символов. Отображает графические символы по мере их получения. Программа, которая зависит от присутствия или отсутствия неграфических символов в строке имеет зависимость от окружения. Функции редактирования, если таковые вообще имеются, которые система исполняет, чтобы создать строку символов - определенное реализацией.

Ввод заканчивается, когда получен определенный реализацией признак конца строки или если строка длиной +n символов. Когда ввод заканчивается, ничего не добавляется в конец строки, и изображение обеспечивается определенным реализацией способом.

Сохраняет строку в c-addr и ее длинну в SPAN.

Примечание: Это слово устаревшее и включено как уступка существующим реализациям. Его функция заменена на 6.1.0695 ACCEPT.

6.2.1485 FALSE CORE EXT
(-- false)
Возвращает флаг false.

См.: 3.1.3.1 Флаги

6.2.1660 HEX CORE EXT
(--)
Устанавливает содержимое BASE в 16.

6.2.1850 MARKER CORE EXT
("<spaces>name" --)
Пропускает ведущие разделители пробелы. Выделяет name, ограниченное пробелом. Создает определение для name с семантикой выполнения, определенной ниже.

name Выполнение: (--)
Восстанавливает все распределение словаря и указатели порядка поиска на состояние, которое они имели непосредственно до определения name. Удаляет определение name и все последующие определения. Восстановление любых все еще существующих структур, которые могли бы обратиться к удаленным определениям или освобожденной области данных предусматривать не обязательно. Никакая другая контекстная информация типа основания счисления не затрагивается.

См.: 3.4.1 Синтаксический анализ, 15.6.2.1580 FORGET.

6.2.1930 NIP CORE EXT
(x1 x2 -- x2)
Удаляет первый элемент ниже вершины стека.

6.2.1950 OF CORE EXT
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: -- of-sys)
Помещает of-sys на стек потока управления. Добавляет семантику времени-выполнения данную ниже к текущему определению. Семантика не завершена, пока не разрешена of-sys потребителем типа ENDOF.

Время-выполнения: (x1 x2 -- | x1)
Если два значения на стеке - не равны, снимает верхнее значение и продолжает выполнение с адреса ячейки определенного потребителем of-sys, например, после следующего ENDOF. Иначе, снимает оба значения и продолжает выполнение в строке.

См.: 6.2.0873 CASE, 6.2.1342 ENDCASE, 6.2.1343 ENDOF.

6.2.2000 PAD CORE EXT
(-- c-addr)
c-addr - адрес временной области, которая может использоваться с целью хранения данных для промежуточной обработки.

См.: 3.3.3.6 Другие временные области.

6.2.2008 PARSE CORE EXT
(char "ccc<char>" -- c-addr u)
Выделяет ccc, ограниченную разделителем char.

c-addr - адрес (в пределах входного буфера) и u - длина выделенной строки. Если область анализа была пуста, результирующая строка имеет нулевую длину.

См.: 3.4.1 Синтаксический анализ.

6.2.2030 PICK CORE EXT
(xu ... x1 x0 u -- xu ... x1 x0 xu)
Удаляет u. Копирует xu на вершину стека. Неопределенная ситуация существует, если до выполнения PICK на стеке меньше u+2 элементов.

6.2.2040 QUERY CORE EXT
(--)
Делает пользовательское устройство ввода данных входным источником. Получает ввод в буфер ввода терминала, заменяя любое предыдущее содержимое. Делает результат, чей адрес возвращается TIB, входным буфером. Установить >IN в ноль.

Примечание: Это слово устаревшее и включено как уступка существующим реализациям.

6.2.2125 REFILL CORE EXT
(-- flag)

Пытается заполнить входной буфер из входного источника, возвращая true flag если успешно.
Если входной источник - пользовательское устройство ввода, пытается получить ввод во входной буфер терминала. Если успешно, делает результат входным буфером, устанавливает >IN в ноль, и возвращает true. Получение строки не содержащей никакие символы рассматривается успешным. Если ввод не доступен из текущего входного источника, возвращает false.

Когда входной источник - строка от EVALUATE, возвращает false и не исполняет никакое другое действие.

См.: 7.6.2.2125 REFILL, 11.6.2.2125 REFILL.

6.2.2148 RESTORE-INPUT CORE EXT
(xn ... x1 n -- flag)
Пытается восстановить спецификацию входного источника к состоянию описанному x1 ... xn. flag - true, если спецификация входного источника не может быть так восстановлена.

Неопределенная ситуация существует если входной источник, представленный параметрами - не совпадает с текущим входным источником.

См.: A.6.2.2182 SAVE-INPUT.

6.2.2150 ROLL CORE EXT
(xu xu-1 ... x0 u -- xu-1 ... x0 xu)
Удаляет u. Вращает u+1 элементов на вершине стека. Неопределенная ситуация существует, если перед выполнением ROLL на стеке имелось меньше u+2 элементов.

6.2.2182 SAVE-INPUT CORE EXT
(-- xn ... x1 n)
x1 ... xn описывают текущее состояние спецификации входного источника для более позднего использования RESTORE-INPUT.

6.2.2218 SOURCE-ID "source-i-d" CORE EXT
(-- 0 | -1)
Идентифицирует входной источник следующим образом:

SOURCE-ID	Входной источник
-1	Строка (через EVALUATE)
0	Пользовательское устройство ввода

См.: 11.6.1.2218 SOURCE-ID.

6.2.2240 SPAN CORE EXT
(-- a-addr)
a-addr - адрес ячейки, содержащей число символов сохраненное последним выполнением EXPECT.

Примечание: Это слово устаревшее и включено как уступка существующим реализациям.

6.2.2290 TIB "t-i-b" CORE EXT
(-- c-addr)
c-addr - адрес буфера ввода терминала.

Примечание: Это слово устаревшее и включено как уступка существующим реализациям.

6.2.2295 TO CORE EXT
Интерпретация: (x "<spaces>name" --)
Пропускает ведущие пробелы, и выделяет name, ограниченное пробелом. Сохраняет x в name. Неопределенная ситуация существует, если name не было определено через VALUE.

Компиляция: ("<spaces>name" --)
Пропускает ведущие пробелы, и выделяет name, ограниченное пробелом. Добавляет семантику времени-выполнения, данную ниже к текущему определению. Неопределенная ситуация существует, если name не было определено через VALUE.

Время-выполнения: (x --)
Сохраняет x в name.

Примечание: Неопределенная ситуация существует, если POSTPONE или [COMPILE], применяется к TO.

См.: 6.2.2405 VALUE, 13.6.1.2295 TO.

6.2.2298 TRUE CORE EXT
(-- true)
Возвращает флаг true, значение одна-ячейка со всеми установленными битами.

См.: 3.1.3.1 Флаги.

6.2.2300 TUCK CORE EXT
(x1 x2 -- x2 x1 x2)
Копирует верхний элемент стека ниже второго элемента стека.

6.2.2330 U.R "u-dot-r" CORE EXT
(u n --)
Отображает u выровненное вправо, в поле шириной n символов. Если число символов требуемых для отображения u больше чем n, все цифры отображаются без ведущих пробелов в поле необходимой ширины.

6.2.2350 U> "u-greater-than" CORE EXT
(u1 u2 -- flag)
flag - true, если и только если u1 больше чем u2.

См.: 6.1.0540 >.

6.2.2395 UNUSED CORE EXT
(-- u)
u - количество оставшегося пространства в области адресуемой HERE, в адресуемых элементах.

6.2.2405 VALUE CORE EXT

(x "<spaces>name" --)
Пропускает ведущие разделители пробелы. Выделяет name, ограниченное пробелом. Создает определение для name с семантикой выполнения, определенной ниже, с начальным значением равным x.

name определено как "value".

name Выполнение: (-- x)
Помещает x на стек. Значение x - то что задано, при создании name, до тех пор пока не выполнена фраза x TO name, задающая новое значение x, связанное с name.

См.: 3.4.1 Синтаксический анализ.

6.2.2440 WITHIN CORE EXT
(n1|u1 n2|u2 n3|u3 -- flag)
Выполняет сравнение проверяемого значения n1|u1 с нижним пределом n2|u2 и верхним пределом n3|u3, возвращая true, если (n2|u2 < n3|u3 и (n2|u2 <= n1|u1 и n1|u1 < n3|u3)) или (n2|u2 > n3|u3 и (n2|u2 <= n1|u1 или n1|u1 < n3|u3)) - true, иначе возвращает false. Неопределенная ситуация существует, если n1|u1, n2|u2, и n3|u3 не все одного типа.

6.2.2530 [COMPILE] "bracket-compile" CORE EXT
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: ("<spaces>name" --)
Пропускает ведущие разделители пробелы. Выделяет name, ограниченное пробелом. Находит name. Если name имеет семантику компиляции отличную от заданной по умолчанию, добавляет ее к текущему определению; иначе добавляет семантику выполнения name. Неопределенная ситуация существует, если name не найдено.

См.: 3.4.1 Синтаксический анализ.

6.2.2535 \ "backslash" CORE EXT
Компиляция: Исполняет семантику выполнения, данную ниже.

Выполнение: ("ccc<eol>"--)
Выделяет и удаляет остаток области анализа. \ слово немедленного исполнения.

См.: 7.6.2.2535 \.

7. Дополнительный Блочный набор слов

7.1 Введение

7.2 Дополнительные термины

Блок: 1024 символа данных на запоминающем устройстве, обозначенные номером блока.

Блочный буфер: зона области данных блочного размера, где блок создан временно доступным для использования. Текущий блочный буфер - блочный буфер больше всего недавно использовавшийся посредством BLOCK, BUFFER, LOAD, LIST, или THRU.

7.3

Дополнительные условия применения

7.3.1 Запросы к окружению

Добавьте таблицу 7.1 к таблице 3.5.

См.: 3.2.6 Запросы к окружению.

Таблица 7.1 - Строки запросов к окружению

Строка	Тип	Константа	Значение
BLOCK	flag	no	блочный набор слов присутствует
BLOCK-EXT	flag	no	дополнительный блочный набор слов присутствует

7.3.2 Область данных

Программа может обращаться к памяти в пределах блочного буфера.

См.: 3.3.3 Область данных.

7.3.3 Области блочного буфера

Адрес блочного буфера, возвращенного BLOCK или BUFFER временный. Вызов BLOCK или BUFFER может сделать адрес предварительно полученного блочного буфера недействительным, также как вызов любого слова типа:

- синтаксические анализаторы;
- отображающие символы на пользовательском устройстве вывода, типа TYPE или EMIT;
- управляющие пользовательским устройством вывода, типа CR или AT-XY;
- получающие или проверяющие на присутствие символов из пользовательского устройства ввода, типа ACCEPT или KEY;
- ждущие состояние или событие, типа MS или EKEY;
- управляющие блочными буферами, типа FLUSH, SAVE-BUFFERS, или EMPTY-BUFFERS;
- выполняющие любую операцию над файлом или каталогом имени файла, которые подразумевают ввод-вывод, типа REFILL или любое слово, которое возвращает ior;
- неявно выполняющие ввод-вывод, типа встроенного и не встроенного текстового интерпретатора при использовании файлов (включая не встроенный подразумеваемый THROW).

Если входной источник - блок, эти ограничения также относятся к адресу, возвращенному SOURCE.

Блочные буферы уникально назначены на блоки.

7.3.4 Синтаксический анализ

Блочный набор слов обеспечивает альтернативный входной источник для текстового интерпретатора. Когда входной источник - блок, BLK должен содержать ненулевой номер блока и входной буфер - 1024-символьный буфер, содержащий этот блок.

Блок традиционно представлен как 16 строк по 64 символа.

Программа может переключать входной источник на блок, используя LOAD или THRU. Входные источники могут быть вложены, используя LOAD, и EVALUATE в любом порядке.

Программа может переустанавливать область анализа в пределах блока, управляя >IN. Более обширное переустановление может быть выполнено, используя SAVE-INPUT и RESTORE-INPUT.

См.: 3.4.1 Синтаксический анализ.

7.3.5 Возможное действие на неопределенную ситуацию

См.: 3.4.4 Возможное действие на неопределенную ситуацию.

- Система с блочным набором слов может устанавливать состояние интерпретации и интерпретировать блок.

7.4 Дополнительные документационные требования

7.4.1 Системная документация

7.4.1.1 Опции определенные реализацией

- формат, используемый для отображения с помощью 7.6.2.1770 LIST (если реализовано);
- длина строки, поддерживаемая 7.6.2.2535 \ (если реализовано).

7.4.1.2 Неопределенные ситуации

- Верное чтение блока не было возможно;
- Исключение ввода-вывода при пересылке блока;
- Недействительный номер блока (7.6.1.0800 BLOCK, 7.6.1.0820 BUFFER, 7.6.1.1790 LOAD);
- Программа непосредственно изменяет содержимое 7.6.1.0790 BLK;
- Нет текущего блочного буфера для 7.6.1.2400 UPDATE.

7.4.1.3 Другая системная документация

- любые ограничения служб системы мультипрограммирования на использование адресов буферов;
- число блоков, доступных для исходного текста и данных.

7.4.2 Программная документация

- число блоков, требуемых программой.

7.5 Соглашение и наименование

7.5.1 ANS Forth системы

Фраза "Предоставляет блочный набор слов" должна быть добавлена к наименованию

любой Стандартной системы, которая предоставляет весь блочный набор слов.

Фраза "Предоставляет имена из расширения блочного набора слов" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет часть расширения блочного набора слов.

Фраза "Предоставляет расширение блочного набора слов" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь блочный набор слов и расширение блочного набора слов.

7.5.2 _____
ANS Forth программы

Фраза "Требуется блочный набор слов" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила блочный набор слов.

Фраза "Требуются имена из расширения блочного набора слов" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила часть расширения блочного набора слов.

Фраза "Требуется расширение блочного набора слов" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила весь блочный набор слов и расширение блочного набора слов.

7.6 _____
Словарь

7.6.1 _____
Блочные слова

7.6.1.0790 BLK "b-l-k" BLOCK
(-- a-addr)
a-addr - адрес ячейки, содержащей нуль или номер интерпретируемого блока запоминающего устройства. Если BLK содержит нуль, входной источник - не блок и может быть идентифицирован SOURCE-ID, если SOURCE-ID является доступным. Неопределенная ситуация существует если программа непосредственно изменяет содержание BLK.

См.: 7.3.3 Области блочного буфера.

7.6.1.0800 BLOCK BLOCK
(u -- a-addr)
a-addr - адрес первого символа блочного буфера назначенного на блок u запоминающего устройства. Неопределенная ситуация существует, если u не доступный номер блока.

Если блок u - уже в блочном буфере, a-addr - адрес этого блочного буфера.

Если блок u - уже не в памяти и есть свободный блочный буфер, перемещает блок u из запоминающего устройства в свободный блочный буфер. a-addr - адрес этого блочного буфера.

Если блок u - уже не в памяти и нет никаких свободных блочных буферов, освобождает блочный буфер. Если для блока в этом буфере было выполнено UPDATE, перемещает блок в запоминающее устройство, и перемещает блок u из запоминающего устройства в этот буфер. a-addr - адрес этого блочного буфера.

При завершении операции, блочный буфер указанный a-addr - текущий

буфер блоков и назначен на u.

7.6.1.0820 BUFFER BLOCK
(u -- a-addr)
a-addr - адрес первого символа блочного буфера назначенного блоку u. Содержание блока не определено. Неопределенная ситуация существует, если u - не доступный номер блока.

Если блок u - уже в блочном буфере, a-addr - адрес этого блочного буфера.

Если блок u - уже не в памяти и есть свободный буфер, a-addr - адрес этого блочного буфера.

Если блок u - уже не в памяти и нет никаких свободных блочных буферов, освобождает блочный буфер. Если для блока в этом буфере было выполнено UPDATE, перемещает блок в запоминающее устройство. a-addr - адрес этого блочного буфера.

При завершении операции, блочный буфер указанный a-addr - текущий буфер блоков и назначен на u.

См.: 7.6.1.0800 BLOCK.

7.6.1.1360 EVALUATE BLOCK
Расширяется семантика 6.1.1360 EVALUATE так, чтобы включить:

Запись нуля в BLK.

7.6.1.1559 FLUSH BLOCK
(--)
Выполняет функцию SAVE-BUFFERS, затем освобождает все блочные буферы.

7.6.1.1790 LOAD BLOCK
(i*x u -- j*x)
Сохраняет текущую спецификацию входного источника. Записывает u в BLK (таким образом делает блок u входным источником, и устанавливает входной буфер в соответствии с его содержимым), обнуляет >IN, и интерпретирует. Когда область анализа исчерпана, восстанавливает предыдущую спецификацию входного источника. Другое состояние стека из-за интерпретации слов с помощью LOAD.

Неопределенная ситуация существует, если u нулевой или - не допустимый номер блока.

См.: 3.4 Интерпретатор текста Forth.

7.6.1.2180 SAVE-BUFFERS BLOCK
(--)
Перемещает содержание каждого модифицированного блочного буфера на запоминающее устройство. Отмечает все буферы как не модифицированные.

7.6.1.2400 UPDATE BLOCK
(--)
Отмечает текущий блочный буфер как измененный. Неопределенная ситуация существует, если нет никакого текущего блочного буфера.

UPDATE не вызывает немедленно ввод-вывод.

См.: 7.6.1.0800 BLOCK, 7.6.1.0820 BUFFER, 7.6.1.1559 FLUSH, 7.6.1.2180 SAVE-BUFFERS.

7.6.2 Расширения блочных слов

7.6.2.1330 EMPTY-BUFFERS BLOCK EXT
(--)
Освобождает все блочные буферы. Не перемещает содержание никакого модифицированного блочного буфера на запоминающее устройство.

См.: 7.6.1.0800 BLOCK.

7.6.2.1770 LIST BLOCK EXT
(u --)
Отображает блок u в формате определенном реализацией. Сохраняет u в SCR.

См.: 7.6.1.0800 BLOCK.

7.6.2.2125 REFILL BLOCK EXT
(-- flag)
Расширяется семантика выполнения 6.2.2125 REFILL следующим образом:

Когда входной источник - блок, делает следующий блок входным источником и текущим входным буфером, добавляя один к значению BLK и устанавливая >IN в ноль. Возвращает true, если новое значение BLK - допустимый номер блока, иначе false.

См.: 6.2.2125 REFILL, 11.6.2.2125 REFILL.

7.6.2.2190 SCR "s-c-r" BLOCK EXT
(-- a-addr)
a-addr - адрес ячейки, содержащей номер последнего блока отображенного с помощью LIST.

7.6.2.2280 THRU BLOCK EXT
(i*x u1 u2 -- j*x)
LOAD последовательные блоки из запоминающего устройства с номерами с u1 по u2. Другое состояние стека из-за интерпретации слов с помощью LOAD.

7.6.2.2535 \ "backslash" BLOCK EXT
Компиляция: Исполняет семантику выполнения, данную ниже.

Выполнение: ("ccc<eol>"--)
Если BLK содержит нуль, выделяет, и удаляет остаток области анализа; иначе выделяет, и удаляет остаток области анализа соответствующей остатку текущей строки. \ - слово немедленного исполнения.

8. Дополнительный набор слов двойных чисел

8.1 Введение

16-бит системы Forth часто используют числа двойной длины. Однако много Forth-

ов на маленьких встроенных системах этого не имеют, и много пользователей Forth на системах с размером ячейки 32 бит или больше находят, что использование чисел двойной длины очень ограничено. Поэтому слова, которые манипулируют объектами двойной длины, были помещены в этот дополнительный набор слов.

8.2 Дополнительные термины и нотации

Нет.

8.3 Дополнительные условия применения

8.3.1 Запросы к окружению

Добавьте таблицу 8.1 к таблице 3.5.

См.: 3.2.6 Запросы к окружению.

Таблица 8.1 - Строки запросов к окружению

Строка	Тип	Константа	Значение
DOUBLE	flag	no	набор слов двойных чисел присутствует
DOUBLE-EXT	flag	no	дополнительный набор слов двойных чисел присутствует

8.3.2 Преобразование входных чисел текстового интерпретатора

Когда текстовый интерпретатор обрабатывает число, которое непосредственно сопровождается десятичной точкой и не найдено как определение name, текстовый интерпретатор должен преобразовать его к числу две-ячейки.

Например, ввод DECIMAL 1234 оставит число одна-ячейка 1234 на стеке, и ввод DECIMAL 1234. оставит число две-ячейки 1234 0 на стеке.

См.: 3.4.1.3 Преобразование входных чисел текстового интерпретатора.

8.4 Дополнительные документационные требования

8.4.1 Системная документация

8.4.1.1 Опции определенные реализацией

- нет дополнительных требований.

8.4.1.2 Неопределенные ситуации

- d вне диапазона n при 8.6.1.1140 D>S.

8.4.1.3 Другая системная документация

- нет дополнительных требований.

8.4.2

- нет дополнительных требований.

8.5 Соглашение и наименование

8.5.1 ANS Forth системы

Фраза "Предоставляет набор слов двойных чисел" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь набор слов двойных чисел.

Фраза "Предоставляет имена из расширения набора слов двойных чисел" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет часть расширения набора слов двойных чисел.

Фраза "Предоставляет расширение набора слов двойных чисел" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь набор слов двойных чисел и расширение набора слов двойных чисел.

8.5.2 ANS Forth программы

Фраза "Требуется набор слов двойных чисел" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила набор слов двойных чисел.

Фраза "Требуются имена из расширения набора слов двойных чисел" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила часть расширения набора слов двойных чисел.

Фраза "Требуется расширение набора слов двойных чисел" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила весь набор слов двойных чисел и расширение набора слов двойных чисел.

8.6 Словарь

8.6.1 Слова двойных чисел

8.6.1.0360 2CONSTANT "two-constant" DOUBLE
(x1 x2 "<spaces>name" --)
Пропускает ведущие разделители пробелы. Выделяет name, ограниченное пробелом. Создает определение для name с семантикой выполнения, определенной ниже.

name объявляется как "two-constant".

name Выполнение: (-- x1 x2)
Помещает пару-ячеек x1 x2 на стек.

См.: 3.4.1 Синтаксический анализ.

8.6.1.0390 2LITERAL "two-literal" DOUBLE
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (x1 x2 --)
Добавляет семантику времени-выполнения данную ниже к текущему

определению.

Время-выполнения: (-- x1 x2)
Помещает пару-ячеек x1 x2 на стек.

8.6.1.0440 2VARIABLE "two-variable" DOUBLE
("<spaces>name" --)
Пропускает ведущие разделители пробелы. Выделяет name, ограниченное пробелом. Создает определение для name с семантикой выполнения, определенной ниже. Резервирует две последовательных ячейки области данных.

name объявляется как "two-variable".

name Выполнение: (-- a-addr)
a-addr - адрес первой (наименьший адрес) ячейки двух последовательных ячеек в области данных, зарезервированной 2VARIABLE, когда оно определило name. Программа ответственна за инициализацию содержимого.

См.: 3.4.1 Синтаксический анализ, 6.1.2410 VARIABLE.

8.6.1.1040 D+ "d-plus" DOUBLE
(d1|ud1 d2|ud2 -- d3|ud3)
Добавляет d2|ud2 к d1|ud1, возвращает сумму d3|ud3.

8.6.1.1050 D- "d-minus" DOUBLE
(d1|ud1 d2|ud2 -- d3|ud3)
Вычитает d2|ud2 из d1|ud1, возвращает разницу d3|ud3.

8.6.1.1060 D. "d-dot" DOUBLE
(d --)
Отображает d в формате свободного поля.

8.6.1.1070 D.R "d-dot-r" DOUBLE
(d n --)
Отображает d выровненное вправо в поле шириной n символов. Если число символов требуемых для отображения d больше чем n, все цифры отображаются без ведущих пробелов в поле необходимой ширины.

8.6.1.1075 D0< "d-zero-less" DOUBLE
(d -- flag)
flag - true, если и только если d - меньше нуля.

8.6.1.1080 D0= "d-zero-equals" DOUBLE
(xd -- flag)
flag - true, если и только если xd - равно нулю.

8.6.1.1090 D2* "d-two-star" DOUBLE
(xd1 -- xd2)

xd2 - результат смещения xd1 на один бит к старшему двоичному разряду, заполнение освобожденного младшего бита нулем.

8.6.1.1100 D2/ "d-two-slash" DOUBLE
(xd1 -- xd2)
xd2 - результат смещения xd1 на один бит к младшему двоичному разряду, оставляет старший бит неизменным.

8.6.1.1110 D< "d-less-than" DOUBLE
(d1 d2 -- flag)

flag - true, если и только если d1 - меньше чем d2.

8.6.1.1120 D= "d-equals" DOUBLE
(xd1 xd2 -- flag)
flag - true, если и только если xd1 - побитно равно xd2.

8.6.1.1140 D>S "d-to-s" DOUBLE
(d -- n)
n - эквивалент d. Неопределенная ситуация существует, если d находится вне диапазона знакового числа одна-ячейка.

8.6.1.1160 DABS "d-abs" DOUBLE
(d -- ud)
ud - абсолютное значение d.

8.6.1.1210 DMAX "d-max" DOUBLE
(d1 d2 -- d3)
d3 большее из d1 и d2.

8.6.1.1220 DMIN "d-min" DOUBLE
(d1 d2 -- d3)
d3 меньшее из d1 и d2.

8.6.1.1230 DNEGATE "d-negate" DOUBLE
(d1 -- d2)
d2 - отрицание d1.

8.6.1.1820 M*/ "m-star-slash" DOUBLE
(d1 n1 +n2 -- d2)
Умножает d1 на n1, производя промежуточный результат тройную-ячейку t. Делит t на +n2 возвращая частное d2 две-ячейки. Неопределенная ситуация существует, если +n2 нулевое или отрицательное, или частное находится вне диапазона целого числа со знаком двойной точности.

8.6.1.1830 M+ "m-plus" DOUBLE
(d1|ud1 n -- d2|ud2)
Прибавляет n к d1|ud1, возвращая сумму d2|ud2.

8.6.2 Расширения слов двойных чисел

8.6.2.0420 2ROT "two-rote" DOUBLE EXT
(x1 x2 x3 x4 x5 x6 -- x3 x4 x5 x6 x1 x2)
Вращает три верхних пары-ячеек на стеке, перенося пару-ячеек x1 x2 на вершину стека.

8.6.2.1270 DU< "d-u-less" DOUBLE EXT
(ud1 ud2 -- flag)
flag - true, если и только если ud1 - меньше чем ud2.

9. Дополнительный набор слов исключений

9.1 Введение

9.2

Дополнительные термины и нотации

Нет.

9.3 Дополнительные условия применения

9.3.1 Значения THROW

Значения THROW {-255...-1} должны использоваться только как определено этим Стандартом. Значения {-4095...-256} должны использоваться только как определено системой.

Если наборы слов доступа к файлу или распределения памяти реализованы, рекомендуется чтобы ненулевые значения ior, располагались в пределах системного диапазона значений THROW, как определено выше. В среде операционной системы, это может иногда выполняться "смещением" диапазона кодов-исключений операционной системы, чтобы попасть в пределы диапазона THROW.

Программы не должны определять значения для использования с THROW в диапазоне {-4095...-1}.

9.3.2 Структура исключения

Структура исключения - зависящий-от-реализаци набор информационных записей текущего состояния выполнения необходимый для надлежащего функционирования CATCH и THROW. Она часто включает глубину стека данных и стека возвратов.

9.3.3 Стек исключения

Стек, используемый CATCH и THROW для вложенных структур исключений. Он может, но не обязательно, быть реализован с использованием стека возвратов.

9.3.4 Запросы к окружению

Добавьте таблицу 9.1 к таблице 3.5.

См.: 3.2.6 Запросы к окружению.

Таблица 9.1 - Строки запросов к окружению

Строка	Тип	Константа	Значение
EXCEPTION	flag	no	набор слов исключений присутствует
EXCEPTION-EXT	flag	no	дополнительный набор слов исключений присутствует

9.3.5 Возможное действие на неопределенную ситуацию

Код	Зарезервирован для	Код	Зарезервирован для
-1	ABORT	-2	ABORT"
-3	переполнение стека	-4	антипереполнение стека
-5	переполнение стека возвратов	-6	антипереполнение стека возвратов
-7	do-loops вложен слишком глубоко в течение выполнения		

```
                                    -8  переполнение словаря
 -9  неверный адрес памяти         -10  деление на нуль
-11  результат вне диапазона       -12  несоответствие типа параметра
-13  неопределенное слово          -14  интерпретация слова только для
                                        компиляции
-15  неверный FORGET               -16  попытка использовать строку
                                        нулевой длины как name
-17  переполнение выходной строки отображаемого числа
                                   -18  переполнение анализируемой строки
-19  определение name слишком длинное
                                   -20  запись в место только для чтения
-21  неподдерживаемая операция (например, AT-XY на слишком-примитивном
     терминале)
                                   -22  несоответствие управляющей структуры
-23  исключение выравнивания адреса
                                   -24  неверный числовой параметр
-25  несогласованность стека возвратов
                                   -26  недоступные параметры цикла
-27  неверная рекурсия             -28  пользовательское прерывание
-29  вложенность компилятора       -30  устаревшая функция
-31  >BODY используется не в CREATE определении
                                   -32  неверный параметр name (например,
                                        TO xxx)
-33  исключение блочного чтения    -34  исключение блочной записи
-35  неверный номер блока          -36  неверная позиция файла
-37  исключение ввода-вывода файла -38  несуществующий файл
-39  неожиданный конец файла       -40  неверная BASE для преобразования
                                        с плавающей запятой
-41  потеря точности               -42  деление на ноль с плавающей точкой
-43  результат с плавающей точкой вне диапазона
                                   -44  переполнение стека с плавающей точкой
-45  антипереполнение стека с плавающей точкой
                                   -46  неверный параметр с плавающей точкой
-47  удаленный список слов компиляции
                                   -48  неверный POSTPONE
-49  переполнение порядка поиска   -50  антипереполнение порядка поиска
-51  измененный список слов компиляции
                                   -52  переполнение стека потока управления
-53  исключение переполнения стека -54  антипереполнение с плавающей точкой
-55  неопознанная ошибка с плавающей точкой
                                   -56  QUIT
-57  исключение при посылке или получении символа
                                   -58  исключение [IF], [ELSE], или [THEN]
```

9.3.6 Обработка исключений

Есть несколько методов применения CATCH и THROW для использования в других процедурах. Обычные использования - выполнение определений, использование стека возвратов, использование циклов, реализация locals и использование входных источников (то есть, с LOAD, EVALUATE, или INCLUDE-FILE).

Когда THROW возвращает управление на CATCH, система должна удалить не только определения, но также и, если присутствует, locals и спецификации входного источника, чтобы возвратить систему к ее надлежащему состоянию для дальнейшего выполнения после CATCH.

9.4 Дополнительные документационные требования

9.4.1 _____
Системная документация

9.4.1.1 _____
Опции определенные реализацией

- Значения, используемые в системе 9.6.1.0875 CATCH и 9.6.1.2275 THROW (9.3.1 Значения THROW, 9.3.5 Возможное действие на неопределенную ситуацию).

9.4.1.2 _____
Неопределенные ситуации

- нет дополнительных требований.

9.4.1.3 _____
Другая системная документация

- нет дополнительных требований.

9.4.2 _____
Программная документация

- нет дополнительных требований.

9.5 _____
Соглашение и наименование

9.5.1 _____
ANS Forth системы

Фраза "Предоставляет набор слов исключений" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь набор слов исключений.

Фраза "Предоставляет имена из расширения набора слов исключений" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет часть расширения набора слов исключений.

Фраза "Предоставляет расширение набора слов исключений" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь набор слов исключений и расширение набора слов исключений.

9.5.2 _____
ANS Forth программы

Фраза "Требуется набор слов исключений" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила набор слов исключений.

Фраза "Требуются имена из расширения набора слов исключений" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила часть расширения набора слов исключений.

Фраза "Требуется расширение набора слов исключений" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила весь набор слов исключений и расширение набора слов исключений.

9.6 _____
Словарь

9.6.1 _____

Слова Исключений

9.6.1.0875 CATCH EXCEPTION
(i*x xt -- j*x 0 | i*x n)
Помещает структуру исключения на стек исключений, и затем выполняется идентификатор исполнения xt (как с EXECUTE) таким способом, что управление может быть передано в точку сразу после CATCH, если THROW выполнено в течение выполнения xt.

Если выполнение xt завершается нормально (то есть, структура исключения помещенная этим CATCH не изъята выполнением THROW) изымает структуру исключения и возвращает ноль на вершине стека данных, выше тех элементов стека которые были бы возвращены xt EXECUTE. Иначе, остаток семантики выполнения определяется THROW.

9.6.1.2275 THROW EXCEPTION
(k*x n -- k*x | i*x n)
Если какие либо биты n ненулевые, выталкивает самую верхнюю структуру исключения со стека исключений, вместе со всем содержимым стека возвратов выше этой структуры. Затем восстанавливает спецификацию входного источника в состояние перед соответствующим CATCH и корректирует состояния всех стеков, определенных этим Стандартом так, чтобы они были те же самые как состояния, сохраненные в структуре исключения (i - то же самое число, что и i во входных параметрах для соответствующего CATCH), помещает n на вершину стека данных, и передает управление на точку сразу после CATCH, которое поместило эту структуру исключения.

Если вершина стека - не нулевая и нет никакой структуры исключения на стеке исключения, поведение следующие:

Если n - минус-один (-1), исполняет функцию 6.1.0670 ABORT (версия ABORT из основного набора слов), не отображая сообщение.

Если n - минус-два, исполняет функцию 6.1.0680 ABORT" (версия ABORT" из основного набора слов), отображая символы ссс связанные с ABORT" которое генерировало THROW.

Иначе, система может отображать зависящее-от-реализации сообщение, возвращающее информацию о состоянии связанном с кодом THROW n. Впоследствии система должна выполнить функцию 6.1.0670 ABORT (версия ABORT из основного набора слов).

9.6.2 _____
Расширения слов исключений

9.6.2.0670 ABORT EXCEPTION EXT
Расширьте семантику 6.1.0670 ABORT, таким образом:
(i*x --) (R: j*x --)
Выполняет функцию -1 THROW.

См.: 6.1.0670 ABORT.

9.6.2.0680 ABORT" "abort-quote" EXCEPTION EXT
Расширьте семантику 6.1.0680 ABORT" таким образом:
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: ("ccc<quote>" --)
Выделяет ссс, ограниченную " (двойная кавычка). Добавляет семантику времени-выполнения данную ниже к текущему определению.

Время-выполнения: (i*x x1 -- | i*x) (R: j*x -- | j*x)
Удаляет x1 из стека. Если любой бит x1 - не нулевой, выполняет функцию -2 THROW, отображая ccc, если нет никакой структуры исключения на стеке исключения.

См.: 3.4.1 Синтаксический анализ, 6.1.0680 ABORT".

10. Дополнительный сервисный набор слов

10.1 Введение

10.2 Дополнительные термины и нотации

Нет.

10.3 Дополнительные условия применения

10.3.1 Символьные типы

Программы, которые используют больше чем семь бит символа 10.6.2.1305 EKEY имеют зависимость от окружения.

См.: 3.1.2 Символьные типы.

Таблица 10.1 - Строки запросов к окружению

Строка	Тип	Константа	Значение
FACILITY	flag	no	сервисный набор слов присутствует
FACILITY-EXT	flag	no	дополнительный сервисный набор слов присутствует

10.4 Дополнительные документационные требования

10.4.1 Системная документация

10.4.1.1 Опции определенные реализацией

- кодирование событий клавиатуры (10.6.2.1305 EKEY);

- длительность такта системных часов;

- повторяемость ожидаемая от выполнения 10.6.2.1905 MS.

10.4.1.2 Неопределенные ситуации

- 10.6.1.0742 AT-XY операция не может быть выполнена на пользовательском устройстве вывода.

10.4.1.3

- нет дополнительных требований.

10.4.2 Программная документация

10.4.2.1 Зависимости от окружения

- Использование больше чем семь битов символа в 10.6.2.1305 EKEY.

10.4.2.2 Другая программная документация

- нет дополнительных требований.

10.5 Соглашение и наименование

10.5.1 ANS Forth системы

Фраза "Предоставляет сервисный набор слов" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь сервисный набор слов.

Фраза "Предоставляет имена из расширения сервисного набора слов" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет часть расширения сервисного набора слов.

Фраза "Предоставляет расширение сервисного набора слов" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь сервисный набор слов и расширение сервисного набора слов.

10.5.2 ANS Forth программы

Фраза "Требуется сервисный набор слов" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила сервисный набор слов.

Фраза "Требуются имена из расширения сервисного набора слов" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила часть расширения сервисного набора слов.

Фраза "Требуется расширение сервисного набора слов" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила весь сервисный набор слов и расширение сервисного набора слов.

10.6 Словарь

10.6.1 Сервисные слова

10.6.1.0742 AT-XY "at-x-y" FACILITY
(u1 u2 --)
Исполняет зависящие-от-реализации шаги так, чтобы следующий отображаемый символ появился в столбце u1, строке u2 пользовательского устройства вывода, верхний левый угол которого - нулевой столбец и нулевая строка. Неопределенная ситуация

существует, если операция не может быть выполнена на пользовательском устройстве вывода с определенными параметрами.

10.6.1.1755 KEY? "key-question" FACILITY
(-- flag)
Если символ доступен, возвращает true. Иначе, возвращает false. Если не символьные события клавиатуры доступны перед первым допустимым символом, они будут отвергнуты и впоследствии недоступны. Символ должен быть возвращен следующим выполнением KEY.

После возвращения KEY? со значением true, последующие выполнения KEY? до выполнения KEY или EKEY также возвращают true, без отмены событий клавиатуры.

10.6.1.2005 PAGE FACILITY
(--)
Перемещение на другую страницу для вывода. Фактическая функция зависит от устройства вывода. На терминале, PAGE очищает экран и сбрасывает позицию курсора к верхнему левому углу. На принтере, PAGE исполняет прогон страницы.

10.6.2
Расширения сервисных слов

10.6.2.1305 EKEY "e-key" FACILITY EXT
(-- u)
Получает одно событие клавиатуры u. Кодирование событий клавиатуры определенное реализацией.

См.: 10.6.1.1755 KEY?, 6.1.1750 KEY.

10.6.2.1306 EKEY>CHAR "e-key-to-char" FACILITY EXT
(u -- u false | char true)
Если событие клавиатуры u соответствует символу в наборе символов определенном реализацией, возвращает этот символ и true. Иначе возвращает u и false.

10.6.2.1307 EKEY? "e-key-question" FACILITY EXT
(-- flag)
Если событие клавиатуры доступно, возвращает true. Иначе возвращает false. Событие должно быть возвращено следующим выполнением EKEY.

После возвращения EKEY? со значением true, последующее выполнение EKEY? до выполнения KEY, KEY? или EKEY также возвращает true, ссылаясь на то же самое событие.

10.6.2.1325 EMIT? "emit-question" FACILITY EXT
(-- flag)
flag - true, если пользовательское устройство вывода готово принять данные и выполнение EMIT на месте EMIT? не испытывало бы неопределенной задержки. Если состояние устройства не определено, flag - true (? false).

10.6.2.1905 MS FACILITY EXT
(u --)
Ожидание, по крайней мере, u миллисекунд.

Примечание: Фактическая длина и изменчивость периода времени зависят

от определенной реализацией разрешающей способности системных часов и других системных и компьютерных характеристик, и находится вне контекста этого Стандарта.

10.6.2.2292 TIME&DATE "time-and-date" FACILITY EXT
(-- +n1 +n2 +n3 +n4 +n5 +n6)
Возвращает текущее время и дату. +n1 - секунды {0...59}, +n2 минуты {0...59}, +n3 - часы {0...23}, +n4 - день {1...31} +n5 - месяц {1...12}, и +n6 - год (например, 1991).

11. Дополнительный набор слов доступа к файлам

11.1 Введение

Эти слова предоставляют доступ к запоминающему устройству в форме "файлов" при следующих допущениях:

- файлы предусмотрены базовой операционной системой;
- имена файлов представлены как символьные строки;
- формат имен файла определен базовой операционной системой;
- открытый файл идентифицирован идентификатором файла одна-ячейка (fileid);
- информация состояния файла (например, позиция, размер) управляется базовой операционной системой;
- к содержанию файла обращаются как к последовательности символов;
- операции чтения файла возвращают фактический размер передачи, который может отличиться от требуемого размера передачи.

11.2 Дополнительные термины

метод доступа к файлу: допустимые средства доступа к файлу, типа "чтения - записи" или "только для чтения".

позиция файла: смещение символа от начала файла.

входной файл: файл, содержащий последовательность строк, которая является входным источником.

11.3 Дополнительные условия применения

11.3.1 Типы данных

Добавьте таблицу 11.1 к таблице 3.1.

Таблица 11.1 - Типы данных

Символ	Тип данных	Размер на стеке
ior	результат ввода-вывода	1 ячейка
fam	метод доступа к файлу	1 ячейка
fileid	идентификатора файла	1 ячейка

11.3.1.1 Идентификаторы файла

Идентификаторы файла - это зависящие-от-реализации значения одна-ячейка, которые передаются к файловым операторам для обозначения конкретных файлов. Открытие файла назначает идентификатор файла, который остается действительным до закрытия файла.

11.3.1.2 Результаты ввода-вывода

Результат ввода-вывода - это числа одна-ячейка, указывающие результат операций ввода-вывода. Значение нуль указывает, что операция ввода-вывода завершилась успешно; другие значения и их смысл - определенное реализацией. Достижение конца файла должно быть сообщено как нуль.

Исключение ввода-вывода при выполнении слова доступа к файлу, которое может возвращать результат ввода-вывода не должно вызывать THROW; индикация исключений возвращается в ior.

11.3.1.3 Методы доступа к файлу

Методы доступа к файлу - это значение одна-ячейка определенное реализацией.

11.3.1.4 Имена файла

Символьная строка, содержащая имя файла. Имя файла может включать зависящий-от-реализации путь имени. Формат имен файла - определенное реализацией.

11.3.2 Блоки в файлах

Если набор слов доступа к файлу реализован, блочный набор слов должен быть реализован.

Блоки могут находиться в файлах, но не обязательно. Тогда:

- Номера блоков могут быть отображены на один или более файлов средствами определенными реализацией. Неопределенная ситуация существует, если требуемый номер блока в настоящее время не отображен;
- Блок, для которого выполнен UPDATE, и который получен из файла, должен быть возвращен назад в тот же самый файл.

11.3.3 Запросы к окружению

Добавьте таблицу 11.2 к таблице 3.5.

См.: 3.2.6 Запросы к окружению.

Таблица 11.2 - Строки запросов к окружению

Строка	Тип	Константа	Значение
FILE	flag	no	файловый набор слов представлен
FILE-EXT	flag	no	расширение файлового набора слов представлено

11.3.4 Входной источник

Набор слов доступа к файлу создает другой входной источник для текстового интерпретатора. Когда входной источник - текстовый файл, BLK должен содержать нуль, SOURCE-ID должен содержать fileid этого текстового файла, и входной буфер должны содержать одну строку текстового файла.

Ввод с INCLUDED, INCLUDE-FILE, LOAD и EVALUATE, должен иметь возможность вложения в любом порядке по крайней мере до восьми уровней.

Программа, которая использует больше чем восемь уровней вложения входного файла, имеет зависимость от окружения.

См.: 3.3.3.5 Входные буферы, 9. Дополнительный набор слов исключений.

11.3.5 Другие временные области

Список слов использующих память во временных областях расширен, для включения 11.6.1.2165 S".

См.: 3.3.3.6 Другие временные области.

11.3.6 Синтаксический анализ

При синтаксическом анализе из текстового файла, использующего разделитель пробел, управляющие символы должны быть обработаны так же как пробел.

Должны поддерживаться строки, по крайней мере, 128 символов. Программа, которая требует строки больше чем 128 символов, имеет зависимость от окружения.

Программа может переустанавливать область анализа в пределах входного буфера, управляя содержанием >IN. Более обширное переустановление может быть выполнено, с использованием SAVE-INPUT и RESTORE-INPUT.

См.: 3.4.1 Синтаксический анализ.

11.4 Дополнительные документационные требования

11.4.1 Системная документация

11.4.1.1 Опции определенные реализацией

- методы доступа к файлу, используемые в 11.6.1.0765 BIN, 11.6.1.1010 CREATE-FILE, 11.6.1.1970 OPEN-FILE, 11.6.1.2054 R/O, 11.6.1.2056 R/W, and 11.6.1.2425 W/O;
- файловые исключения;
- признак конца строки файла (11.6.1.2090 READ-LINE);
- формат имени файла (11.3.1.4 Имена файла);
- информация, возвращаемая 11.6.2.1524 FILE-STATUS;
- состояние входного файла после исключения (11.6.1.1717 INCLUDE-FILE, 11.6.1.1718 INCLUDED);
- ior значения и смысл (11.3.1.2 Результаты ввода-вывода);
- максимальная глубина вложенности файлового ввода (11.3.4 Входной источник);
- максимальный размер входной строки (11.3.6 Синтаксический анализ);
- методы отображения диапазона блоков на файлы (11.3.2 Блоки в файлах);
- предусмотренное число строковых буферов (11.6.1.2165 S");
- размер строкового буфера, используемого в 11.6.1.2165 S".

11.4.1.2 Неопределенные ситуации

- попытка позиционировать файл вне его границ (11.6.1.2142 REPOSITION-FILE);
- попытка читать из еще не записанной позиции файла (11.6.1.2080 READ-FILE, 11.6.1.2090 READ-LINE);
- fileid неверен (11.6.1.1717 INCLUDE-FILE);
- исключение ввода-вывода чтения или закрытия fileid (11.6.1.1717 INCLUDE-FILE, 11.6.1.1718 INCLUDED);

- указанный файл не может быть открыт (11.6.1.1718 INCLUDED);
- требование не существующего номера блока (11.3.2 Блоки в файлах);
- использование 11.6.1.2218 SOURCE-ID, когда 7.6.1.0790 BLK - не нулевой.

11.4.1.3 Другая системная документация

- нет дополнительных требований.

11.4.2 Программная документация

11.4.2.1 Зависимости от окружения

- требование строки больше 128 символов (11.3.6 Синтаксический анализ);
- использование больше чем восемь уровней вложения входного файла (11.3.4 Входной источник).

11.4.2.2 Другая программная документация

- нет дополнительных требований.

11.5 Соглашение и наименование

11.5.1 ANS Forth системы

Фраза "Предоставляет набор слов доступа к файлу" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь набор слов доступа к файлу.

Фраза "Предоставляет имена из расширения набора слов доступа к файлу" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет часть расширения набора слов доступа к файлу.

Фраза "Предоставляет расширение набора слов доступа к файлу" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь набор слов доступа к файлу и расширение набора слов доступа к файлу.

11.5.2 ANS Forth программы

Фраза "Требуется набор слов доступа к файлу" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила набор слов доступа к файлу.

Фраза "Требуются имена из расширения набора слов доступа к файлу" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система

предоставила часть расширения набора слов доступа к файлу.

Фраза "Требуется расширение набора слов доступа к файлу" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила весь набор слов доступа к файлу и расширение набора слов доступа к файлу.

11.6 Словарь

11.6.1 Слова доступа к файлу

11.6.1.0080 ("paren" FILE
("ccc<paren>" --)
Расширьте семантику 6.1.0080 (включив:

При синтаксическом анализе из текстового файла, если конец области анализа достигнут прежде, чем найдена правая круглая скобка, снова наполняет входной буфер из следующей строки файла, обнуляет >IN, и возобновляет синтаксический анализ, повторяет этот процесс до нахождения правой круглой скобки или достижения конца файла.

11.6.1.0765 BIN FILE
(fam1 -- fam2)
Изменяет метод доступа к файлу определенный реализацией fam1, дополнительно выбирая "binary", то есть не строко-ориентированный метод доступа к файлу, возвращая метод доступа fam2.

См.: 11.6.1.2054 R/O, 11.6.1.2056 R/W, 11.6.1.2425 W/O.

11.6.1.0900 CLOSE-FILE FILE
(fileid -- ior)
Закрывает файл, идентифицированный fileid. ior - определенный реализацией код результата ввода-вывода.

11.6.1.1010 CREATE-FILE FILE
(c-addr u fam -- fileid ior)
Создает файл именованный символьной строкой, определенной c-addr и u, и открывает его с методом доступа к файлу fam. Смысл значений fam является определенным реализацией. Если файл с тем же самым именем уже существует, пересоздает его как пустой файл.

Если файл был успешно создан и открыт, ior нулевой, fileid его идентификатор, и файл был позиционирован в начало.

Иначе, ior - код результата ввода-вывода определенный реализацией и fileid неопределенное.

11.6.1.1190 DELETE-FILE FILE
(c-addr u -- ior)
Удаляет файл, именованный символьной строкой, определенной c-addr u. ior - определенный реализацией код результата ввода-вывода.

11.6.1.1520 FILE-POSITION FILE
(fileid -- ud ior)
ud - текущая позиция для файла идентифицированного fileid. ior является кодом результата ввода-вывода определенным реализацией. ud не определен если ior является ненулевым.

11.6.1.1522 FILE-SIZE FILE

(fileid -- ud ior)

ud - размер в символах файла, идентифицированного fileid. ior является кодом результата ввода-вывода определенным реализацией. Эта операция не затрагивает значение, возвращаемое FILE-POSITION. ud не определен, если ior ненулевой.

11.6.1.1717 INCLUDE-FILE FILE
(i*x fileid -- j*x)
Удаляет fileid со стека. Сохраняет спецификацию текущего входного источника, включая текущее значение SOURCE-ID. Запоминает fileid в SOURCE-ID. Делает файл определенный fileid входным источником. Записывает нуль в BLK. Другое состояние стека по причине обработанных слов.

Повторяет до конца файла: читает строку из файла, заполняет входной буфер содержимым этой строки, обнуляет >IN, и интерпретирует.

Текстовая интерпретация начинается в позиции файла, где произошло бы следующее чтение файла.

Когда достигнут конец файла, закрывает файл и восстанавливает спецификацию входного источника к ее сохраненному значению.

Неопределенная ситуация существует, если fileid неверен, если есть исключение ввода-вывода чтения fileid, или если исключение ввода-вывода происходит при закрытии fileid. Когда неопределенная ситуация существует, состояние (открытый или закрытый) для любых файлов которые интерпретировались - определенное реализацией.

См.: 11.3.4 Входной источник.

11.6.1.1718 INCLUDED FILE
(i*x c-addr u -- j*x)
Удаляет c-addr u со стека. Сохраняет спецификацию текущего входного источника, включая текущее значение SOURCE-ID. Открывает файл определенный c-addr u, сохраняет результирующий fileid в SOURCE-ID, и делает его входным источником. Сохраняет нуль в BLK. Другое состояние стека по причине включенных слов.

Повторяет до конца файла: читает строку из файла, заполняет входной буфер содержимым этой строки, обнуляет >IN, и интерпретирует.

Текстовая интерпретация начинается в позиции файла, где произошло бы следующее чтение файла.

Когда достигнут конец файла, закрывает файл и восстанавливает спецификацию входного источника к ее сохраненному значению.

Неопределенная ситуация существует, если заданный файл не может быть открыт, если исключение ввода-вывода происходит при чтении файла, или если исключение ввода-вывода происходит при закрытии файла. Когда неопределенная ситуация существует, состояние (открытый или закрытый) любых файлов которые интерпретировались - определенное реализацией.

См.: 11.6.1.1717 INCLUDE-FILE.

11.6.1.1970 OPEN-FILE FILE
(c-addr u fam -- fileid ior)
Открывает файл заданный символьной строкой определенной c-addr u, с методом доступа к файлу, обозначенным fam. Смысл значений fam определенный реализацией.

Если файл успешно открыт, ior нулевой, fam - его идентификатор, и файл был позиционирован начало.

Иначе, ior - код результата ввода-вывода определенный реализацией и fam неопределенное.

11.6.1.2054 R/O "r-o" FILE
(-- fam)
fam - значение определенное реализацией для выбора метода доступа к файлу "только для чтения".

См.: 11.6.1.1010 CREATE-FILE, 11.6.1.1970 OPEN-FILE.

11.6.1.2056 R/W "r-w" FILE
(-- fam)
fam - значение определенное реализацией для выбора метода доступа к файлу "чтение - запись".

См.: 11.6.1.1010 CREATE-FILE, 11.6.1.1970 OPEN-FILE.

11.6.1.2080 READ-FILE FILE
(c-addr u1 fileid -- u2 ior)
Читает u1 последовательных символов в c-addr с текущей позиции файла, идентифицированного fileid.

Если u1 символы читаются без исключения, ior нулевое, и u2 равно u1.

Если конец файла достигнут, прежде прочтения u1 символов, ior нуль и u2 - число фактически прочитанных символов.

Если операция инициализирована когда значение, возвращенное FILE-POSITION равно значению, возвращенному FILE-SIZE для файла идентифицированного fileid - ior нулевое, и u2 нулевое.

Если происходит исключение, ior - код результата ввода-вывода определенный реализацией, и u2 - число символов переданное в c-addr без исключения.

Неопределенная ситуация существует, если операция инициализирована когда значение, возвращенное FILE-POSITION большее чем значение, возвращенное FILE-SIZE для файла, идентифицированного fileid, или если требуемая операция пытается читать не записанные части файла.

При завершении операции, FILE-POSITION возвращает следующую позицию файла после последнего прочтенного символа.

11.6.1.2090 READ-LINE FILE
(c-addr u1 fileid -- u2 flag ior)
Читает следующую строку из файла, определенного fileid в память с адреса c-addr. Читаются максимум u1 символов. До двух символов завершения строки определенных реализацией могут читаться в память в конец строки, но не включаются в количество u2. Буфер строки, предусмотренный c-addr должен быть, по крайней мере, длиной u1+2 символов.

Если операция успешная, flag - true, и ior нулевой. Если признак конца строки был получен прежде прочтения символов u1, тогда фактически читается u2-число символов, не включая признак конца строки(0 <= u2 <= u1). Когда u1 = u2, признак конца строки должен все же быть достигнут.

Если операция инициализирована когда значение, возвращенное FILE-POSITION равно значению, возвращенному FILE-SIZE для файла, идентифицированного fileid, flag - false, ior нулевое, и u2 нулевое. Если ior ненулевое, в течение операции произошло исключение, и ior - определенный реализацией код завершения ввода-вывода.

Неопределенная ситуация существует, если операция инициализирована когда значение, возвращенное FILE-POSITION больше чем значение, возвращенное FILE-SIZE для файла, идентифицированного fileid, или если требуемая операция пытается читать не записанные части файла.

При завершении операции, FILE-POSITION возвращает следующую позицию файла после последнего прочтенного символа.

11.6.1.2142 REPOSITION-FILE FILE
(ud fileid -- ior)
Переустанавливает позицию файла, идентифицированного fileid на ud. ior - код завершения ввода-вывода определенный реализацией. Неопределенная ситуация существует если файл позиционирован вне границ файла.

При завершении операции, FILE-POSITION возвращает значение ud.

11.6.1.2147 RESIZE-FILE FILE
(ud fileid -- ior)
Устанавливает размер файла, идентифицированного fileid на ud. ior - код завершения ввода-вывода определенный реализацией.

Если результирующий файл больше чем файл перед операцией, часть файла добавленного в результате операции могла бы быть не записана.

При завершении операции, FILE-SIZE возвращает значение ud и FILE-POSITION возвращает неопределенное значение.

См.: 11.6.1.2080 READ-FILE, 11.6.1.2090 READ-LINE.

11.6.1.2165 S" "s-quote" FILE
Расширьте семантику 6.1.2165 S" так:

Интерпретация: ("ccc<quote>" -- c-addr u)
Выделяет ccc, ограниченную " (двойная кавычка). Сохраняет результирующую строку c-addr u во временном местоположении. Максимальная длина временного буфера зависящая-от-реализаци, но должна быть не меньше чем 80 символов. Последующие использования S" могут перезаписывать временный буфер. По крайней мере, один такой буфер должен быть предусмотрен.

Компиляция: ("ccc<quote>" --)
Выделяет ccc, ограниченную " (двойная кавычка). Добавляет семантику времени-выполнения данную ниже к текущему определению.

Время-выполнения: (-- c-addr u)
Возвращает c-addr и u, которые описывают строку, состоящую из

символов ссс.

См.: 3.4.1 Синтаксический анализ, 6.2.0855 C", 6.1.2165 S", 11.3.5 Другие временные области.

11.6.1.2218 SOURCE-ID "source-i-d" FILE
(-- 0 | -1 | fileid)
Расширьте 6.2.2218 SOURCE-ID, чтобы включить ввод текстового файла следующим образом:

SOURCE-ID	Входной источник
fileid	Текстовый файл "fileid"
-1	Строка (через EVALUATE)
0	Пользовательское устройство ввода

Неопределенная ситуация существует, если SOURCE-ID используется, когда BLK содержит ненулевое значение.

11.6.1.2425 W/O "w-o" FILE
(-- fam)
fam - значение определенное реализацией для выбора метода доступа к файлу "только запись".

См.: 11.6.1.1010 CREATE-FILE, 11.6.1.1970 OPEN-FILE.

11.6.1.2480 WRITE-FILE FILE
(c-addr u fileid -- ior)
Записывает u символов из c-addr в файл, идентифицированный fileid начиная с его текущей позиции. ior - определенный реализацией код результата ввода-вывода.

При завершении операции, FILE-POSITION возвращает следующую позицию файла после последнего символа записанного в файл, и FILE-SIZE возвращает значение большее или равное значению возвращенному FILE-POSITION.

См.: 11.6.1.2080 READ-FILE, 11.6.1.2090 READ-LINE.

11.6.1.2485 WRITE-LINE FILE
(c-addr u fileid -- ior)
Записывает u символов из c-addr, сопровождаемых зависящим-от-реализации признаком конца строки в файл, идентифицированный fileid начиная с его текущей позиции. ior - определенный реализацией код результата ввода-вывода.

При завершении операции, FILE-POSITION возвращает следующую позицию файла после последнего символа записанного в файл, и FILE-SIZE возвращает значение большее или равное значению возвращенному FILE-POSITION.

См.: 11.6.1.2080 READ-FILE, 11.6.1.2090 READ-LINE.

11.6.2 Расширения слов доступа к файлу

11.6.2.1524 FILE-STATUS FILE EXT
(c-addr u -- x ior)
Возвращает состояние файла, идентифицированного символьной строкой c-addr u. Если файл существует, ior нулевой; иначе ior - код

завершения ввода-вывода определенный реализацией. x содержит определенную реализацией информацию о файле.

11.6.2.1560 FLUSH-FILE FILE EXT
(fileid -- ior)
Пытается инициировать запись на запоминающее устройство любую буферизированную информацию, записанную в файл, относящийся к fileid, и информация о размере файла будет записана в каталоге хранения при изменении. Если операция успешна, ior нулевой. Иначе, это - код завершения ввода-вывода определенный реализацией.

11.6.2.2125 REFILL FILE EXT
(-- flag)
Расширьте семантику выполнения 6.2.2125 REFILL следующим:

Когда входной источник - текстовый файл, пытается читать следующую строку из текстового входного файла. Если успешно, делает результат текущим входным буфером, обнуляет >IN, и возвращает true. Иначе возвращает false.

См.: 6.2.2125 REFILL, 7.6.2.2125 REFILL.

11.6.2.2130 RENAME-FILE FILE EXT
(c-addr1 u1 c-addr2 u2 -- ior)

Переименовывает файл, именованный символьной строкой c-addr1 u1 в имя из символьной строки c-addr2 u2. ior - код завершения ввода-вывода определенный реализацией.

12. Дополнительный набор слов для плавающей точки

12.1 Введение

12.2 Дополнительные термины и нотация

12.2.1 Определение терминов

С-плавающей-точкой-выровненный адрес: Адрес ячейки памяти, к которой можно обращаться как к числу с плавающей точкой.

Двойной-с-плавающей-точкой-выровненный адрес: Адрес ячейки памяти, к которой можно обращаться как к 64 бит IEEE числу с плавающей точкой двойной точности.

Одинарный-с-плавающей-точкой-выровненный адрес: Адрес ячейки памяти, к которой можно обращаться как к 32 бит IEEE числу с плавающей точкой одинарной точности.

Число с плавающей точкой IEEE: Число с плавающей точкой одинарной или двойной точности, как определено в ANSI/IEEE 754-1985.

12.2.2 Нотация

12.2.2.1

Числовая нотация

Следующая нотация используется для определения синтаксиса внешнего представления числа с плавающей точкой:

- Каждый компонент числа с плавающей точкой определен правилом, состоящим из имени компонента (выделенного курсивом в угловых скобках, например, <знак>), символов := и связанных лексем и метасимволов;
- Лексемы могут быть литеральными символами (полужирным шрифтом, например, E) или правильными именами в угловых скобках (например, <цифра>);
- Метасимвол * используется, чтобы определить нуль или большее количество появлений предшествующей лексемы (например, <digit>*);
- Лексемы заключенные в [и] необязательные (например, [<знак>]);
- Вертикальные линии отделяют выборы из списка лексем заключенных в фигурные скобки (например{ + | - }).

12.2.2.2 Стековая нотация

Когда стек с плавающей точкой отделен от стека данных - стековая нотация с плавающей точкой такая:

(F: before -- after)

12.3 Дополнительные условия применения

12.3.1 Типы данных

Добавьте таблицу 12.1 к таблице 3.1.

Таблица 12.1 - Типы данных

Символ	Тип данных	Размер на стеке
r	числа с плавающей точкой	определенное реализацией
f-addr	с-плавающей-точкой выровненный адрес	1 ячейка
sf-addr	одинарный-с-плавающей-точкой выровненный адрес	1 ячейка
df-addr	двойной-с-плавающей-точкой выровненный адрес	1 ячейка

12.3.1.1 Адреса

Набор с-плавающей-точкой-выровненных адресов - подмножество определенного реализацией набора выровненных адресов. Добавление размера числа с плавающей точкой к с-плавающей-точкой-выровненному адресу должно производить с-плавающей-точкой-выровненный адрес.

Набор двойных-с-плавающей-точкой-выровненных адресов - подмножество определенного реализацией набора выровненных адресов. Добавление размера 64-бит IEEE числа с плавающей точкой двойной точности к двойному-с-плавающей-точкой-выровненному адресу должно произвести двойной-с-плавающей-точкой-выровненный адрес.

Набор одинарных-с-плавающей-точкой-выровненных адресов - подмножество

определенного реализацией набора выровненных адресов. Добавление размера 32-бит IEEE числа с плавающей точкой одинарной точности к одинарному-с-плавающей-точкой-выровненному адресу должно произвести одинарный-с-плавающей-точкой-выровненный адрес.

12.3.1.2 Числа с плавающей точкой

Внутреннее представление числа с плавающей точкой включая формат и точность мантиссы и формат и диапазон экспоненты, является определенным реализацией.

Любое округление или усечение чисел с плавающей точкой - определенное реализацией.

12.3.2 Операции с плавающей точкой

"Округление к ближайшему" означает округление результата операции с плавающей точкой к представимому значению ближайшему к результату. Если два ближайших представимых значения одинаково близки к результату, должен быть представлен тот, наименьший значащий бит которого нулевой.

"Округление к отрицательной бесконечности" означает округление результата операции с плавающей точкой к представимому значению ближайшему и не большему чем результат.

12.3.3 Стек с плавающей точкой

Последний пришел, первым ушел список, который должен использоваться всеми операторами с плавающей точкой.

Размер стека с плавающей точкой - определенный реализацией. По умолчанию стек с плавающей точкой должен быть отделен от стека данных и стека возвратов. Программа может определять, располагаются ли числа с плавающей точкой на стеке данных, посылая строку "FLOATING-STACK" в ENVIRONMENT?.

Размер стека с плавающей точкой должен быть, по крайней мере 6 элементов.

Программа, которая зависит от стека с плавающей точкой, являющегося большим чем 6 элементов, имеет зависимость от окружения.

12.3.4 Запросы к окружению

Добавьте таблицу 12.2 к таблице 3.5.

См.: 3.2.6 Запросы к окружению.

Таблица 12.2 - Строки запросов к окружению

Строка	Тип	Константа	Значение
FLOATING	flag	no	набор слов с плавающей точкой представлен
FLOATING-EXT	flag	no	расширенный набор слов с плавающей точкой представлен
FLOATING-STACK	n	yes	если n = нуль, числа с плавающей точкой располагаются на стеке данных; иначе n – максимальная глубина отделенного стека с плавающей точкой

MAX-FLOAT	r	yes	самое большое пригодное для использования число с плавающей точкой

12.3.5 Выравнивание адреса

Так как адрес возвращенный словом созданным CREATE не обязательно выровнен для любой части класса данных с плавающей точкой, программа должна выровнять адрес (чтобы он был выровненным: с плавающей точкой, одинарным-с-плавающей-точкой, или двойным-с-плавающей-точкой) перед доступом по адресу к данным с плавающей точкой.

См.: 3.3.3.1 Выравнивание адреса, 12.3.1.1 Адреса.

12.3.6 Переменные

Программа может адресовать память в зонах областей данных, сделанных доступными с помощью FVARIABLE. Эти области могут быть несмежными участками с зонами последовательно распределенными с помощью , (запятой) или ALLOT.

См.: 3.3.3.3 Переменные.

12.3.7 Преобразование входных чисел текстового интерпретатора

Если набор слов с плавающей точкой присутствует в словаре, и текущее основание DECIMAL, входной алгоритм преобразования чисел должен быть расширен для распознавания чисел с плавающей точкой в такой форме:

```
Преобразуемая строка := <мантисса><экспонента>
    <мантисса> := [<знак>]<цифры>[.<цифры0>]
    <экспонента> := E[<знак>]<цифры0>
    <знак> := { + | - }
    <цифры> := <цифра><цифры0>
```



```
    <цифры0> := <цифра>*
    <цифра> := { 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 }
```

Это - примеры допустимых представлений чисел с плавающей точкой в источнике программы:

```
1E   1.E   1.E0   +1.23E-1   -1.23E+1
```

См.: 3.4.1.3 Преобразование входных чисел текстового интерпретатора, 12.6.1.0558 >FLOAT.

12.4 Дополнительные документационные требования

12.4.1 Системная документация

12.4.1.1 Опции определенные реализацией

- формат и диапазон чисел с плавающей точкой (12.3.1 Типы данных, 12.6.1.2143 REPRESENT);
- результат 12.6.1.2143 REPRESENT, когда число с плавающей точкой находится вне диапазона;

- округление или усечение чисел с плавающей точкой (12.3.1.2 Числа с плавающей точкой);
- размер стека с плавающей точкой (12.3.3 Стек с плавающей точкой);
- ширина стека с плавающей точкой (12.3.3 Стек с плавающей точкой).

12.4.1.2 Неопределенные ситуации

- DF@ или DF! используется с адресом, который - не выровнен как двойной-с-плавающей-точкой;
- F@ или F! используется с адресом, который - не выровнен как с плавающей точкой;
- результат с плавающей точкой вне диапазона (например, в 12.6.1.1430 F/);
- SF@ или SF! используется с адресом, который - не выровнен как единственный-с-плавающей-точкой;
- BASE - не десятичная (12.6.1.2143 REPRESENT, 12.6.2.1427 F., 12.6.2.1513 FE., 12.6.2.1613 FS.);
- оба параметра равняются нулю (12.6.2.1489 FATAN2);
- косинус параметра нулевой для 12.6.2.1625 FTAN;
- d точно не может быть представлено как число с плавающей точкой в 12.6.1.1130 D>F;
- деление на нуль (12.6.1.1430 F/);
- экспонента слишком большая для преобразования (12.6.2.1203 DF!, 12.6.2.1204 DF@, 12.6.2.2202 SF!, 12.6.2.2203 SF@);
- число с плавающей точкой меньше чем один (12.6.2.1477 FACOSH);
- число с плавающей точкой меньше чем или равно минус-один (12.6.2.1554 FLNP1);
- число с плавающей точкой меньше чем или равно нулю (12.6.2.1553 FLN, 12.6.2.1557 FLOG);
- число с плавающей точкой меньше чем нуль (12.6.2.1487 FASINH, 12.6.2.1618 FSQRT);
- модуль числа с плавающей точкой больше чем один (12.6.2.1476 FACOS, 12.6.2.1486 FASIN, 12.6.2.1491 FATANH);
- целая часть числа с плавающей точкой не может быть представлена как d в 12.6.1.1470 F>D;
- строка больше чем область вывода отображаемого числа (12.6.2.1427 F., 12.6.2.1513 FE., 12.6.2.1613 FS.).

12.4.1.3 Другая системная документация

- нет дополнительных требований.

12.4.2 Программная документация

12.4.2.1 Зависимости от окружения

- требование стека с плавающей точкой большего чем 6 элементов (12.3.3 Стек с плавающей точкой).

12.4.2.2 Другая программная документация

- нет дополнительных требований.

12.5.1 ANS Forth системы

Фраза "Предоставляет набор слов с плавающей точкой" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь набор слов с

плавающей точкой.

Фраза "Предоставляет имена из расширения набора слов с плавающей точкой" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет часть расширения набора слов с плавающей точкой.

Фраза "Предоставляет расширение набора слов с плавающей точкой" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь набор слов с плавающей точкой и расширение набора слов с плавающей точкой.

12.5.2 ANS Forth программы

Фраза "Требуется набор слов с плавающей точкой" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила набор слов с плавающей точкой.

Фраза "Требуются имена из расширения набора слов с плавающей точкой" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила часть расширения набора слов с плавающей точкой.

Фраза "Требуется расширение набора слов с плавающей точкой" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила весь набор слов с плавающей точкой и расширение набора слов с плавающей точкой.

12.6 Словарь

12.6.1 Слова с плавающей точкой

12.6.1.0558 **>FLOAT** "to-float" FLOATING

(c-addr u -- true | false) (F: -- r |)

или (c-addr u -- r true | false)

Пытается выполнить преобразование строки определенной c-addr и u во внутреннее представление с плавающей точкой. Если строка представляет допустимое число с плавающей точкой в синтаксисе ниже, возвращает его значение r и true. Если строка не представляет допустимое число с плавающей точкой, возвращает только false.

Строка пробелов должна быть обработана как специальный случай представления нуля.

Синтаксис преобразуемой строки := <мантисса>[<экспонента>]
<мантисса> := [<знак>]{<цифры>[.<цифры0>] | .<цифры> }
<экспонента> := <маркер><цифры0>
<маркер> := {<e-форма> | <знак-форма>}
<e-форма> := <e-символ>[<знак-форма>]
<знак-форма> := { + | - }
<e-символ>:= { D | d | E | e }

12.6.1.1130 **D>F** "d-to-f" FLOATING

(d --) (F: -- r) or (d -- r)

r - число с плавающей точкой эквивалентное d. Неопределенная ситуация существует если d точно не может быть представлено как значение с плавающей точкой.

12.6.1.1400 **F!** "f-store" FLOATING

(f-addr --) (F: r --) or (r f-addr --)

Сохраняет r в f-addr.

12.6.1.1410 F* "f-star" FLOATING
(F: r1 r2 -- r3) or (r1 r2 -- r3)
Умножает r1 на r2, возвращает r3.

12.6.1.1420 F+ "f-plus" FLOATING
(F: r1 r2 -- r3) or (r1 r2 -- r3)
Прибавляет r1 к r2, возвращает сумму r3.

12.6.1.1425 F- "f-minus" FLOATING
(F: r1 r2 -- r3) or (r1 r2 -- r3)
Вычитает r2 из r1, возвращает r3.

12.6.1.1430 F/ "f-slash" FLOATING
(F: r1 r2 -- r3) or (r1 r2 -- r3)
Делит r1 на r2, возвращает частное r3. Неопределенная ситуация существует если r2 нулевое, или частное находится вне диапазона числа с плавающей точкой.

12.6.1.1440 F0< "f-zero-less-than" FLOATING
(-- flag) (F: r --) or (r -- flag)
flag - true, если и только если r - меньше чем нуль.

12.6.1.1450 F0= "f-zero-equals" FLOATING
(-- flag) (F: r --) or (r -- flag)
flag - true, если и только если r - равно нулю.

12.6.1.1460 F< "f-less-than" FLOATING
(-- flag) (F: r1 r2 --) or (r1 r2 -- flag)
flag - true, если и только если r1 - меньше чем r2.

12.6.1.1470 F>D "f-to-d" FLOATING
(-- d) (F: r --) or (r -- d)
d - целое число со знаком две-ячейки эквивалентное целой части из r. Дробная часть r откинута. Неопределенная ситуация существует, если целая часть r точно не может быть представлена как целое число со знаком две-ячейки.

12.6.1.1472 F@ "f-fetch" FLOATING
(f-addr --) (F: -- r) or (f-addr -- r)
r - значение, сохраненное в f-addr.

12.6.1.1479 FALIGN "f-align" FLOATING
(--)
Если указатель области данных - не выровнен с плавающей точкой, резервирует достаточно пространства данных для выравнивания.

12.6.1.1483 FALIGNED "f-aligned" FLOATING
(addr -- f-addr)
f-addr - первый с-плавающей-точкой-выровненный адрес больший или равный addr.

12.6.1.1492 FCONSTANT "f-constant" FLOATING
("<spaces>name" --) (F: r --) or (r "<spaces>name" --)
Пропускает ведущие разделители пробелы. Выделяет name, ограниченное пробелом. Создает определение для name с семантикой выполнения определенной ниже.

name объявлено как "f-constant".

name Выполнение: (--) (F: -- r) or (-- r)

Помещает r на стек с плавающей точкой.

См.: 3.4.1 Синтаксический анализ.

12.6.1.1497 FDEPTH "f-depth" FLOATING
(-- +n)
+n - число значений, содержащихся на отделенном по умолчанию стеке с плавающей точкой. Если числа с плавающей точкой сохраняются на стеке данных, +n является текущим числом возможных значений с плавающей точкой, содержащихся на стеке данных.

12.6.1.1500 FDROP "f-drop" FLOATING
(F: r --) or (r --)
Удаляет r со стека с плавающей точкой.

12.6.1.1510 FDUP "f-dupe" FLOATING
(F: r -- r r) or (r -- r r)
Дублирует r.

12.6.1.1552 FLITERAL "f-literal" FLOATING
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (F: r --) or (r --)
Добавляет семантику времени-выполнения, данную ниже к текущему определению.

Время-выполнения: (F: -- r) or (-- r)
Помещает r на стек с плавающей точкой.

12.6.1.1555 FLOAT+ "float-plus" FLOATING
(f-addr1 -- f-addr2)
Добавляет размер в адресуемых элементах числа с плавающей точкой к f-addr1, возвращает f-addr2.

12.6.1.1556 FLOATS FLOATING
(n1 -- n2)
n2 - размер в адресуемых элементах n1 чисел с плавающей точкой.

12.6.1.1558 FLOOR FLOATING
(F: r1 -- r2) or (r1 -- r2)
Округление r1 к целому значению, использующему правило "округления к отрицательной бесконечности", возвращает r2.

12.6.1.1562 FMAX "f-max" FLOATING
(F: r1 r2 -- r3) or (r1 r2 -- r3)
r3 большее из r1 и r2.

12.6.1.1565 FMIN "f-min" FLOATING
(F: r1 r2 -- r3) or (r1 r2 -- r3)
r3 меньшее из r1 и r2.

12.6.1.1567 FNEGATE "f-negate" FLOATING
(F: r1 -- r2) or (r1 -- r2)
r2 - отрицание r1.

12.6.1.1600 FOVER "f-over" FLOATING
(F: r1 r2 -- r1 r2 r1) or (r1 r2 -- r1 r2 r1)
Помещает копию r1 на вершину стека с плавающей точкой.

12.6.1.1610 FROT "f-rote" FLOATING
(F: r1 r2 r3 -- r2 r3 r1) or (r1 r2 r3 -- r2 r3 r1)

Вращает три верхних элемента стека с плавающей точкой.

12.6.1.1612 FROUND "f-round" FLOATING
(F: r1 -- r2) or (r1 -- r2)
Округляет r1 к целому значению, использующему правило "округления к ближайшему", возвращает r2.

См.: 12.3.2 Операции с плавающей точкой.

12.6.1.1620 FSWAP "f-swap" FLOATING
(F: r1 r2 -- r2 r1) or (r1 r2 -- r2 r1)
Меняет два верхних элемента стека с плавающей точкой.

12.6.1.1630 FVARIABLE "f-variable" FLOATING
("<spaces>name" --)
Пропускает ведущие разделители пробелы. Выделяет name, ограниченное пробелом. Создает определение для name с семантикой выполнения определенной ниже. Резервирует 1 FLOATS адресуемый элемент области данных как с-плавающей-точкой-выровненный адрес.

name объявлено как "f-variable".

name Выполнение: (--f-addr)
f-addr - адрес области данных, зарезервированной FVARIABLE когда оно создавало name. Программа ответственна за инициализацию содержимого зарезервированного пространства.

См.: 3.4.1 Синтаксический анализ.

12.6.1.2143 REPRESENT FLOATING
(c-addr u -- n flag1 flag2) (F: r --)
или (r c-addr u -- n flag1 flag2)
В c-addr, помещает символьное-строковое внешнее представление мантиссы числа с плавающей точкой r. Возвращает n как экспоненту в десятичном основании, знак как flag1 и "допустимый результат" как flag2. Символьная строка должна состоять из u самых старших цифр мантиссы представленной как десятичная дробь с подразумеваемой десятичной точкой слева от первой цифры, и нулевой первой цифрой только если все цифры нулевые. Мантисса округлена к u цифрам в соответствии с правилом "округления к ближайшему"; n откорректирован, в случае необходимости, чтобы соответствовать округленной величине мантиссы. Если flag2 - true тогда r было в диапазоне определенных реализацией чисел с плавающей точкой. Если flag1 - true тогда r - отрицательное.

Неопределенная ситуация существует, если значение BASE - не десятичное десять.

Когда flag2 - false, n и flag1 - определенное реализацией, как содержимое c-addr. При этих обстоятельствах, строка в c-addr должна состоять из графических символов.

См.: 3.2.1.2 Преобразование цифр, 6.1.0750 BASE, 12.3.2 Операции с плавающей точкой.

12.6.2 Расширение слов с плавающей точкой

12.6.2.1203 DF! "d-f-store" FLOATING EXT
(df-addr --) (F: r --) or (r df-addr --)
Сохраняет число с плавающей точкой r в df-addr как 64-бит IEEE

число с двойной точностью. Если мантисса внутреннего представления r имеет большую точность, чем IEEE формат с двойной точностью, оно должно быть округлено, используя правило "округления к ближайшему". Неопределенная ситуация существует, если экспонента r слишком большая, для размещения в IEEE формате с двойной точностью.

См.: 12.3.1.1 Адреса, 12.3.2 Операции с плавающей точкой.

12.6.2.1204 DF@ "d-f-fetch" FLOATING EXT
(df-addr --) (F: -- r) or (df-addr -- r)
Выбирает 64-бит IEEE число с двойной точностью, сохраненное в df-addr на стек с плавающей точкой как r во внутреннем представлении. Если IEEE мантисса с двойной точностью имеет большую точность, чем внутреннее представление, оно будет округлено к внутреннему представлению, используя правило "округления к ближайшему". Неопределенная ситуация существует если экспонента IEEE представления с двойной точностью слишком большая, для размещения во внутреннем представлении.

См.: 12.3.1.1 Адреса, 12.3.2 Операции с плавающей точкой.

12.6.2.1205 DFALIGN "d-f-align" FLOATING EXT
(--)
Если указатель области данных - выровнен не на двойное-с-плавающей-точкой, резервирует достаточно пространства данных для выравнивания.

См.: 12.3.1.1 Адреса.

12.6.2.1207 DFALIGNED "d-f-aligned" FLOATING EXT
(addr -- df-addr)
df-addr - первый двойное-с-плавающей-точкой-выровненный адрес больший или равный addr.

См.: 12.3.1.1 Адреса.

12.6.2.1208 DFLOAT+ "d-float-plus" FLOATING EXT
(df-addr1 -- df-addr2)
Добавляет размер в адресуемых элементах 64-бит IEEE числа с двойной точностью к df-addr1, возвращает df-addr2.

См.: 12.3.1.1 Адреса.

12.6.2.1209 DFLOATS "d-floats" FLOATING EXT
(n1 -- n2)
n2 - размер в адресуемых элементах n1 64-бит IEEE числа с двойной точностью.

12.6.2.1415 F** "f-star-star" FLOATING EXT
(F: r1 r2 -- r3) or (r1 r2 -- r3)
Возводит r1 в степень r2, возвращает результат r3.

12.6.2.1427 F. "f-dot" FLOATING EXT
(--) (F: r --) or (r --)
Отображает с конечным пробелом верхнее число со стека с плавающей точкой, используя нотацию с фиксированной точкой:

[-] <цифры>.<цифры0>

Неопределенная ситуация существует, если значение BASE - не (десятичное) десять или если символьное строковое представление превышает размер буфера выходной строки отображаемого числа.

См.: 12. 12.6.1.0558 >FLOAT.

12.6.2.1474 FABS "f-abs" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
r2 - абсолютное значение r1.

12.6.2.1476 FACOS "f-a-cos" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
r2 - главный угол в радианах, чей косинус - r1. Неопределенная ситуация существует, если |r1| больше единицы.

12.6.2.1477 FACOSH "f-a-cosh" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
r2 - значение с плавающей точкой, чей гиперболический косинус - r1. Неопределенная ситуация существует, если r1 - меньше единицы.

12.6.2.1484 FALOG "f-a-log" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
Возводит десять в степень r1, возвращает r2.

12.6.2.1486 FASIN "f-a-sine" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
r2 - главный угол в радианах, чей синус - r1. Неопределенная ситуация существует, если |r1| больше единицы.

12.6.2.1487 FASINH "f-a-cinch" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
r2 - значение с плавающей точкой, чей гиперболический синус - r1. Неопределенная ситуация существует, если r1 - меньше единицы.

12.6.2.1488 FATAN "f-a-tan" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
r2 - главный угол в радианах, чей тангенс - r1.

12.6.2.1489 FATAN2 "f-a-tan-two" FLOATING EXT
(F: r1 r2 -- r3) or (r1 r2 -- r3)
r3 - угол в радианах, чей тангенс - r1/r2. Неопределенная ситуация существует, если r1 и r2 нулевые.

12.6.2.1491 FATANH "f-a-tan-h" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
r2 - значение с плавающей точкой, чей гиперболический тангенс - r1. Неопределенная ситуация существует, если r1 - вне диапазона с -1E0 по 1E0.

12.6.2.1493 FCOS "f-cos" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
r2 - косинус угла r1 в радианах.

12.6.2.1494 FCOSH "f-cosh" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
r2 - гиперболический косинус r1.

12.6.2.1513 FE. "f-e-dot" FLOATING EXT
(--) (F: r --) or (r --)
Отображает с конечным пробелом верхнее число со стека с плавающей

точкой используя инженерную нотацию, где мантисса больше или равна 1.0 и меньше чем 1000.0 и десятичная экспонента - множитель трех.

Неопределенная ситуация существует, если значение BASE - не (десятичное) десять или если символьное строковое представление превышает размер буфера выходной строки отображаемого числа.

См.: 6.1.0750 BASE, 12.3.2 Операции с плавающей точкой, 12.6.1.2143 REPRESENT.

12.6.2.1515 FEXP "f-e-x-p" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
Возводит e в степень r1, возвращает r2.

12.6.2.1516 FEXPM1 "f-e-x-p-m-one" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
Возводит e в степень r1, и вычитает один, возвращает r2.

12.6.2.1553 FLN "f-l-n" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
r2 - натуральный логарифм r1. Неопределенная ситуация существует если r1 - меньше или равно нулю.

12.6.2.1554 FLNP1 "f-l-n-p-one" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
r2 - натуральный логарифм величины r1 плюс один. Неопределенная ситуация существует если r1 - меньше или равно минус один.

12.6.2.1557 FLOG "f-log" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
r2 - десятичный логарифм r1. Неопределенная ситуация существует если r1 меньше или равно нулю.

12.6.2.1613 FS. "f-s-dot" FLOATING EXT
(--) (F: r --) or (r --)
Отображает, с конечным пробелом, верхнее число со стека с плавающей точкой в экспоненциальном формате:

<мантисса><экспонента>
где:
<мантисса> := [-]<цифра>.<цифры0>
<экспонента> := E[-]<цифры>

Неопределенная ситуация существует, если значение BASE - не (десятичное) десять или если символьное строковое представление превышает размер буфера выходной строки отображаемого числа.

См.: 6.1.0750 BASE, 12.3.2 Операции с плавающей точкой, 12.6.1.2143 REPRESENT.

12.6.2.1614 FSIN "f-sine" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
r2 - синус угла r1 в радианах.

12.6.2.1616 FSINCOS "f-sine-cos" FLOATING EXT
(F: r1 -- r2 r3) or (r1 -- r2 r3)
r2 - синус угла r1 в радианах. r3 - косинус угла r1 в радианах.

12.6.2.1617 FSINH "f-cinch" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
r2 - гиперболический синус r1.

12.6.2.1618 FSQRT "f-square-root" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
r2 - квадратный корень r1. Неопределенная ситуация существует, если r1 - меньше нуля.

12.6.2.1625 FTAN "f-tan" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
r2 - тангенс угла r1 в радианах. Неопределенная ситуация существует если cos(r1) нулевое.

12.6.2.1626 FTANH "f-tan-h" FLOATING EXT
(F: r1 -- r2) or (r1 -- r2)
r2 - гиперболический тангенс r1.

12.6.2.1640 F~ "f-proximate" FLOATING EXT
(-- flag) (F: r1 r2 r3 --) or (r1 r2 r3 -- flag)
Если r3 положительно, flag - true если абсолютное значение (r1 минус r2) меньше чем r3.

Если r3 нулевое, flag - true если зависящее-от-реализации кодирование r1 и r2 точно идентично (положительный и отрицательный нуль неравны если они имеют различное кодирование).

Если r3 - отрицательное, flag - true если абсолютное значение (r1 минус r2) меньше чем абсолютное значение r3 умноженное на сумму абсолютных значений r1 и r2.

12.6.2.2035 PRECISION FLOATING EXT
(-- u)
Возвращает число u значащих цифр, в настоящее время используемых в F., FE., или FS..

12.6.2.2200 SET-PRECISION FLOATING EXT
(u --)
Устанавливает число u значащих цифр, в настоящее время используемых в F., FE., или FS..

12.6.2.2202 SF! "s-f-store" FLOATING EXT
(sf-addr --) (F: r --) or (r sf-addr --)
Сохраняет число с плавающей точкой r как 32-бит IEEE с одинарной точностью число в sf-addr. Если мантисса внутреннего представления r имеет большую точность чем IEEE формат с одинарной точностью, оно будет округленно с использованием правила "округления к ближайшему". Неопределенная ситуация существует если экспонента r слишком большая, чтобы быть размещенной в формате IEEE с одинарной точностью.

См.: 12.3.1.1 Адреса, 12.3.2 Операции с плавающей точкой.

12.6.2.2203 SF@ "s-f-fetch" FLOATING EXT
(sf-addr --) (F: -- r) or (sf-addr -- r)
Выбирает 32-бит IEEE число с одинарной точностью, сохраненное в sf-addr на стек с плавающей точкой как r во внутреннем представлении. Если IEEE мантисса с одинарной точностью имеет большую точность чем внутреннее представление, оно будет округлено к внутреннему представлению с использованием правила "округления к ближайшему". Неопределенная ситуация существует если экспонента IEEE представления с одинарной точностью слишком большая, чтобы быть размещенной во внутреннем представлении.

См.: 12.3.1.1 Адреса, 12.3.2 Операции с плавающей точкой.

12.6.2.2204 SFALIGN "s-f-align" FLOATING EXT
(--)
Если указатель области данных - выровнен не на одинарное-с-плавающей-точкой, резервирует достаточно пространства данных для выравнивания.

См.: 12.3.1.1 Адреса.

12.6.2.2206 SFALIGNED "s-f-aligned" FLOATING EXT
(addr -- sf-addr)
sf-addr - первый одинарный-с-плавающей-точкой-выровненный адрес больший или равный addr.

См.: 12.3.1.1 Адреса.

12.6.2.2207 SFLOAT+ "s-float-plus" FLOATING EXT
(sf-addr1 -- sf-addr2)
Прибавляет размер в адресуемых элементах 32-бит IEEE числа с одинарной точностью к sf-addr1, возвращает sf-addr2.

См.: 12.3.1.1 Адреса.

12.6.2.2208 SFLOATS "s-floats" FLOATING EXT
(n1 -- n2)
n2 - размер в адресуемых элементах n1 32-бит IEEE чисел с одинарной точностью.

См.: 12.3.1.1 Адреса.

13. Дополнительный набор слов Locals

13.1 Введение

См.: Приложение A.13 Locals набор слов.

13.2 Дополнительные термины и нотации

Нет.

13.3 Дополнительные условия применения

13.3.1 Locals

local - объект данных, чья семантика выполнения должна возвратить его значение, чей контекст должен быть ограничен определением в котором он объявлен, и чье использование в определении не должно препятствовать повторной входимости или рекурсии.

13.3.2 Запросы к окружению

Добавьте таблицу 13.1 к таблице 3.5.

См.: 3.2.6 Запросы к окружению.

Таблица 13.1 - Строки запросов к окружению

Строка	Тип	Константа	Значение
#LOCALS	n	yes	максимальное число local переменных в определении
LOCALS	flag	no	locals набор слов присутствует
LOCALS-EXT	flag	no	дополнительный locals набор слов присутствует

13.3.3 Обработка locals

Чтобы поддерживать набор слов locals, система должна предоставить механизм для получения сообщений определенных с помощью(LOCAL) и реагировать как описано здесь.

В течение компиляции определения после : (двоеточие), :NONAME, или DOES>, программа может начинать посылать системе сообщения local идентификатора. Процесс должен начаться, когда послано первое сообщение. Процесс должен закончиться, когда послано "последнее local" сообщение. Система должна отслеживать имена, порядок, и число идентификаторов, содержащихся в законченной последовательности.

13.3.3.1 Семантика компиляции

Система при получении последовательности сообщения local-идентификатора должна выполнять следующие действия во время компиляции:

a) Создать временные входы словаря для каждого из идентификаторов, переданного в (LOCAL) так, что каждый идентификатор будет вести себя как local. Эти временные входы словаря должны исчезнуть в конце определения обозначенного ; (точка с запятой), ;CODE, или DOES>. Система не обязана поддерживать эти идентификаторы таким же образом как она делает другие входы словаря, чтобы они могли быть найдены нормальным процессом поиска в словаре. Кроме того, если набор слов Search-Order присутствует, идентификаторы local должны всегда искаться перед любым из списков слов в любом определимом порядке поиска, и ни одно из слов Search-Order не должно изменять locals' привилегированную позицию в порядке поиска. Local идентификаторы могут находиться на запоминающем устройстве.
b) Для каждого идентификатора переданного в (LOCAL), система должна генерировать соответствующую кодовую последовательность, которая делает следующее во время выполнения:
   1) Распределяет ресурс памяти, способный содержать значение local. Память должна быть распределена способом, который не препятствует повторной входимости или рекурсии в определении использующем local.
   2) Инициализирует значение, используя верхний элемент на стеке данных. Если объявлен больше чем один local, верхний элемент на стеке должен быть перемещен в local идентифицированный первым, следующий элемент должен быть перемещен во второй, и так далее.

   Ресурс памяти может быть стеком возвратов или может быть реализован другим способом, таким как регистры. Ресурс памяти не должен быть стеком данных. Использование locals не должно ограничивать использование стека данных прежде или после точки объявления.
c) Принимается, что любой из законных методов завершения выполнения

определения, конкретно: ; (точка с запятой), ;CODE, DOES> или EXIT, освободит ресурс памяти распределенный для locals, если таковые вообще имеются, объявленный в этом определении. ABORT должно освободить все ресурсы памяти local, и CATCH / THROW (если реализовано) должны освободить такие ресурсы для всех определений, чье выполнение заканчивается.

d) Отдельные наборы locals могут быть объявлены в определяющих словах перед DOES> для использования определяющим словом, и после DOES> для использования определяемым словом.

Система, реализующая набор слов Locals должна поддерживать объявление по крайней мере восьми locals в определении.

13.3.3.2 Ограничения Синтаксиса

Слова немедленного исполнения в программе могут использовать (LOCAL), чтобы осуществить синтаксисы для объявлений local со следующими ограничениями:

a) Программа не должна компилировать никакой выполнимый код в текущее определение между временем выполнения (LOCAL), чтобы идентифицировать первый local для этого определения и временем посылки единственного требуемого "последнего local" сообщения;
b) Позиция в источнике программы в которой послана последовательность (LOCAL) сообщений, упомянутая здесь как точка в которой объявлен locals, не должна лежать в пределах области любой управляющей структуры;
c) Locals не должны быть объявлены пока не были удалены значения, предварительно помещенные на стек возвратов в пределах определения;
d) После того, как locals определения были объявлены, программа может помещать данные на стек возвратов. Однако если это сделано, locals не будут доступны, пока эти значения не будут удалены со стека возвратов;
e) Слова, которые возвращают идентификаторы исполнения, типа ' (tick), ['], или FIND, не должны использоваться с local именами;
f) Программа, которая объявляет больше чем восемь locals в отдельном определении, имеет зависимость от окружения;

g) Locals могут быть доступны или модифицированы в пределах управляющих структур, включая do-loops;
h) На local имена нельзя ссылаться посредством POSTPONE и [COMPILE].

См.: 3.4 Интерпретатор текста Forth.

13.4 Дополнительные документационные требования

13.4.1 Системная документация

13.4.1.1 Опции определенные реализацией

- максимальное число locals в определении (13.3.3 Обработка locals, 13.6.2.1795 LOCALS|).

13.4.1.2 Неопределенные ситуации

- выполнение именованного local, в состоянии интерпретации (13.6.1.0086 (LOCAL));
- имя, не определенное с помощью VALUE или LOCAL (13.6.1.2295 TO).

13.4.1.3

- нет дополнительных требований.

13.4.2 _____
Программная документация

13.4.2.1 _____
Зависимости от окружения

- объявление больше чем восемь locals в отдельном определении (13.3.3 Обработка locals).

13.4.2.2 _____
Другая программная документация

- нет дополнительных требований.

13.5 _____
Соглашение и наименование

13.5.1 _____
ANS Forth системы

Фраза "Предоставляет Locals набор слов" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь Locals набор слов.

Фраза "Предоставляет имена из расширения Locals набора слов" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет часть расширения Locals набора слов.

Фраза "Предоставляет расширение Locals набора слов" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь Locals набор слов и расширение Locals набора слов.

13.5.2 _____
ANS Forth программы

Фраза "Требуется Locals набор слов" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила Locals набор слов.

Фраза "Требуются имена из расширения Locals набора слов" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила часть расширения Locals набора слов.

Фраза "Требуется расширение Locals набора слов" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила весь Locals набор слов и расширение Locals набора слов.

13.6 _____
Словарь

13.6.1 _____
Locals слова

13.6.1.0086 (LOCAL) "paren-local-paren" LOCAL
Интерпретация: Семантика интерпретации для этого слова не определена.

Выполнение: (c-addr u --)
Когда выполнено в течение компиляции, (LOCAL) посылает сообщение

системе, которое имеет одно из двух значений. Если u ненулевое, сообщение идентифицирует новый local, чье имя определения дается строкой символов идентифицированной c-addr u. Если u нулевое, сообщение - "последний local" и c-addr не имеет никакого значения.

Результат выполнения (LOCAL) в течение компиляции определения - создание набора именованных local идентификаторов, каждый из которых - имя определения которое имеет только семантику выполнения в пределах этого источника определений.

local Выполнение: (-- x)
Помещает значение local, x на стек. Значение local инициализировано как описано в 13.3.3 Обработка locals, и может быть изменено с помощью TO предшествующим имени local. Неопределенная ситуация существует когда local выполнено в состоянии интерпретации.

Примечание: Это слово не имеет специальной семантики компиляции в обычном смысле, потому что оно предоставляет доступ к системной возможности использования другими пользователем-определенными словами, которые его включают. Однако, средство locals в целом и последовательность посылаемых сообщений, определяет специфические правила использования с семантическими значениями, которые описаны подробно в разделе 13.3.3 Обработка locals.

Примечание: Это слово не предназначено для прямого использования в определении, для объявления этих locals определений. Оно вместо этого используется системными или пользовательскими компилирующими словами. Эти компилирующие слова в свою очередь определяют их собственный синтаксис, и могут использоваться непосредственно в определениях для объявления locals. В этом контексте, синтаксис для (LOCAL) определен в терминах последовательности сообщений времени-компиляции и описан подробно в разделе 13.3.3 Обработка locals.

Примечание: Набор слов Locals изменяет синтаксис и семантику 6.2.2295 TO как определено в расширении основного набора слов.

См.: 3.4 Интерпретатор текста Forth.

13.6.1.2295 TO LOCAL
Расширьте семантику 6.2.2295 TO таким образом:

Интерпретация: (x "<spaces>name" --)
Пропускает ведущие пробелы, и выделяет name, ограниченное пробелом. Сохраняет x в name. Неопределенная ситуация существует, если name не было определено с помощью VALUE.

Компиляция: ("<spaces>name" --)
Пропускает ведущие пробелы, и выделяет name, ограниченное пробелом. Добавляет семантику времени-выполнения, данную ниже к текущему определению. Неопределенная ситуация существует, если name не было определено с помощью VALUE или (LOCAL).

Время-выполнения: (x --)
Сохраняет x в name.

Примечание: Неопределенная ситуация существует, если POSTPONE или [COMPILE], применяется к TO.

См.: 3.4.1 Синтаксический анализ, 6.2.2295 TO, 6.2.2405 VALUE, 13.6.1.0086 (LOCAL).

13.6.2 Расширения Locals слов

13.6.2.1795 LOCALS| "locals-bar" LOCAL EXT

Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: ("<spaces>name1" "<spaces>name2" ... "<spaces>namen" "|" --)

Создает до восьми идентификаторов local, периодически пропуская ведущие пробелы, выделя name, и выполняя 13.6.1.0086 (LOCAL). Список locals, который будет определен, заканчивается |. Добавляет семантику времени-выполнения данную ниже к текущему определению.

Время-выполнения: (xn ... x2 x1 --)

Инициализирует до восьми идентификаторов local как описано в 13.6.1.0086 (LOCAL), каждый из которых берет как его начальное значение верхний элемент стека, удаляя его со стека. Идентификатор name1 инициализирован x1, идентификатор name2 - x2, и т.д. При вызове, каждый local будет возвращать его значение. Значение local может быть изменено, использованием 13.6.1.2295 TO.

14. Дополнительный набор слов распределения памяти

14.1 Введение

14.2 Дополнительные термины и нотация

Нет.

14.3 Дополнительные условия применения

14.3.1 Тип данных результата ввода-вывода

Результат ввода - числа одна-ячейка, указывающие результат операций ввода-вывода. Значение нуль указывает, что операция ввода-вывода завершилась успешно; другие значения и их смысл - определенное реализацией.

Добавьте таблицу 14.1 к таблице 3.1.

Таблица 14.1 - Типы данных

Символ	Тип данных	Размер на стеке
ior	результат ввода-вывода	1 ячейка

14.3.2 Запросы к окружению

Добавьте таблицу 14.2 к таблице 3.5.

См.: 3.2.6 Запросы к окружению.

Таблица 14.2 - Строки запросов к окружению

Строка	Тип	Константа	Значение
MEMORY-ALLOC	flag	no	набор слов распределения памяти присутствует
MEMORY-ALLOC-EXT	flag	no	дополнительный набор слов распределения памяти присутствует

14.3.3 _____
Распределенные области

Программа может адресовать память в зонах области данных, сделанных доступными с помощью ALLOCATE или RESIZE и еще не освобожденными с помощью FREE.

См.: 3.3.3 Области данных.

14.4 _____
Дополнительные документационные требования

14.4.1 _____
Системная документация

14.4.1.1 _____
Опции определенные реализацией

- смысл и значение ior (14.3.1 Тип данных результата ввода-вывода, 14.6.1.0707 ALLOCATE, 14.6.1.1605 FREE, 14.6.1.2145 RESIZE).

14.4.1.2 _____
Неопределенные ситуации

- нет дополнительных требований.

14.4.1.3 _____
Другая системная документация

- нет дополнительных требований.

14.4.2 _____
Программная документация

- нет дополнительных требований.

14.5 _____
Соглашение и наименование

14.5.1 _____
ANS Forth системы

Фраза "Предоставляет набор слов распределения памяти" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь набор слов распределения памяти.

Фраза "Предоставляет имена из расширения набора слов распределения памяти" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет часть расширения набора слов распределения памяти.

Фраза "Предоставляет расширение набора слов распределения памяти" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь набор слов распределения памяти и расширение набора слов распределения памяти.

14.5.2 ANS Forth программы

Фраза "Требуется набор слов распределения памяти" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила набор слов распределения памяти.

Фраза "Требуются имена из расширения набора слов распределения памяти" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила часть расширения набора слов распределения памяти.

Фраза "Требуется расширение набора слов распределения памяти" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила весь набор слов распределения памяти и расширение набора слов распределения памяти.

14.6 Словарь

14.6.1 Слова распределения памяти

14.6.1.0707 ALLOCATE MEMORY
(u -- a-addr ior)

Распределяет u адресуемых элементов непрерывной области данных. Указатель области данных не затронут этой операцией. Начальное содержимое распределенного пространства не определено.

Если распределение успешное, a-addr - выровненный стартовый адрес распределенного пространства и ior нулевое.

Если операция ошибочна, a-addr не представляет допустимый адрес и ior - код завершения ввода-вывода определенный реализацией.

См.: 6.1.1650 HERE, 14.6.1.1605 FREE, 14.6.1.2145 RESIZE.

14.6.1.1605 FREE MEMORY
(a-addr -- ior)
Возвращает непрерывную зону области данных, обозначенную a-addr в систему для последующего распределения. a-addr должен указать зону области данных, которая была предварительно получена с помощью ALLOCATE или RESIZE. Указатель области данных не затронут этой операцией.

Если операция успешна, ior нулевое. Если операция ошибочна, ior - код завершения ввода-вывода определенный реализацией.

См.: 6.1.1650 HERE, 14.6.1.0707 ALLOCATE, 14.6.1.2145 RESIZE.

14.6.1.2145 RESIZE MEMORY
(a-addr1 u -- a-addr2 ior)
Изменяет распределение непрерывной области данных, начинающейся с адреса a-addr1, предварительно распределенной ALLOCATE или RESIZE, до u адресуемых элементов. u может быть или больше или меньше чем текущий размер зоны. Указатель области данных не затронут этой операцией.

Если операция успешна, a-addr2 - выровненный стартовый адрес u адресуемых элементов распределенной памяти и ior нулевое. a-addr2 может быть, но не обязательно, такой же как a-addr1. Если они - не

равны, значения содержащиеся в области a-addr1 скопированы в a-addr2, до минимального размера любой из этих двух областей. Если они равны, значения содержащиеся в области сохраняются по минимуму u или первоначального размера. Если a-addr2 - не такой же как a-addr1, область памяти a-addr1 возвращается системе согласно операции FREE.

Если операция ошибочна, a-addr2 равен a-addr1, область памяти a-addr1 не затронута, и ior - код результата ввода-вывода определенный реализацией.

См.: 6.1.1650 HERE, 14.6.1.0707 ALLOCATE, 14.6.1.1605 FREE.

14.6.2 Расширения слов распределения памяти

Нет.

15. Дополнительный набор слов утилит

15.1 Введение

Этот дополнительный набор слов содержит слова, наиболее часто используемые в течение разработки приложения.

15.2 Дополнительные термины и нотация

Нет.

15.3 Дополнительные условия применения

15.3.1 Запросы к окружению

Добавьте таблицу 15.1 к таблице 3.5.

См.: 3.2.6 Запросы к окружению.

Таблица 15.1 - Строки запросов к окружению

Строка	Тип	Константа	Значение
TOOLS	flag	no	набор слов утилит присутствует
TOOLS-EXT	flag	no	дополнительный набор слов утилит присутствует

15.3.2 Словарь Forth

Программа, использующая слова CODE или ;CODE, связанные с кодом ассемблера имеет зависимость от окружения на этой специфической системе команд и нотации ассемблера.

Программы, использующие слова EDITOR или ASSEMBLER требуют набора слов Порядка поиска или эквивалентную определенную реализацией возможность.

См.: 3.3 Словарь Forth.

15.4 Дополнительные документационные требования

15.4.1 Системная документация

15.4.1.1 Опции определенные реализацией

- завершающая последовательность для входного потока 15.6.2.0470 ;CODE и 15.6.2.0930 CODE;
- способ обработки входного потока 15.6.2.0470 ;CODE и 15.6.2.0930 CODE;
- возможность порядка поиска для 15.6.2.1300 EDITOR и 15.6.2.0740 ASSEMBLER (15.3.3 Словарь Forth);
- источник и формат отображения с помощью 15.6.1.2194 SEE.

15.4.1.2 Неопределенные ситуации

- удаление списка слов компиляции (15.6.2.1580 FORGET);
- количество элементов на стеке потока управления меньше чем u+1 (15.6.2.1015 CSPICK, 15.6.2.1020 CSROLL);
- имя не может быть найдено (15.6.2.1580 FORGET);
- имя не определено через 6.1.1000 CREATE (15.6.2.0470 ;CODE);
- 6.1.2033 POSTPONE применено к 15.6.2.2532 [IF];
- достижение конца входного источника перед согласованием 15.6.2.2531 [ELSE] или 15.6.2.2533 [THEN] (15.6.2.2532 [IF]);
- удаление необходимого определения (15.6.2.1580 FORGET).

15.4.1.3 Другая системная документация

- нет дополнительных требований.

15.4.2 Программная документация

15.4.2.1 Зависимости от окружения

- использование слов 15.6.2.0470 ;CODE или 15.6.2.0930 CODE.

15.4.2.2 Другая программная документация

- нет дополнительных требований.

15.5 Соглашение и наименование

15.5.1 ANS Forth системы

Фраза "Предоставляет набор слов утилит" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь набор слов утилит.

Фраза "Предоставляет имена из расширения набора слов утилит" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет часть расширения набора слов утилит.

Фраза "Предоставляет расширение набора слов утилит" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь набор слов утилит и расширение набора слов утилит.

15.5.2 _____
ANS Forth программы

Фраза "Требуется набор слов утилит" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила набор слов утилит.

Фраза "Требуются имена из расширения набора слов утилит" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила часть расширения набора слов утилит.

Фраза "Требуется расширение набора слов утилит" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила весь набор слов утилит и расширение набора слов утилит.

15.6 _____
Словарь

15.6.1 _____
Слова Утилит

15.6.1.0220 .S "dot-s" TOOLS
(--)
Копирует и отображает значения находящиеся на стеке данных. Формат отображения зависящий-от-реализаци.

.S может быть реализовано с использованием слов вывода отображаемых чисел. Следовательно, его использование может разрушать временную область, идентифицированную #>.

См.: 3.3.3.6 Другие временные области.

15.6.1.0600 ? "question" TOOLS
(a-addr --)
Отображает значение, сохраненное в a-addr.

? может быть реализовано, с использованием слов вывода отображаемых чисел. Следовательно, его использование может разрушать временную область, идентифицированную #>.

См.: 3.3.3.6 Другие временные области.

15.6.1.1280 DUMP TOOLS
(addr u --)
Отображает содержание u последовательных адресов, начинающихся с addr. Формат отображения зависящий от реализации.

DUMP может быть реализовано, с использованием слов вывода отображаемых чисел. Следовательно, его использование может разрушать временную область, идентифицированную #>.

См.: 3.3.3.6 Другие Временные Области.

15.6.1.2194 SEE TOOLS
("<spaces>name" --)
Отображает удобочитаемое представление определения именованного

слова. Источник представления (декомпиляция объектного кода, блок источника, и т.д.) и специфическая форма отображения - определенное реализацией.

SEE может быть реализовано, с использованием слов вывода отображаемых чисел. Следовательно, его использование может разрушать временную область, идентифицированную #>.

См.: 3.3.3.6 Другие временные области.

15.6.1.2465 WORDS TOOLS
(--)
Перечисляет имена определений в первом списке слов порядка поиска. Формат отображения зависящий-от-реализаци.

WORDS может быть реализовано, с использованием слов вывода отображаемых чисел. Следовательно, его использование может разрушать временную область, идентифицированную #>.

См.: 3.3.3.6 Другие Временные Области.

15.6.2 _____
Расширения слов Утилит

15.6.2.0470 ;CODE "semicolon-code" TOOLS EXT
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: colon-sys --)
Добавляет семантику времени-выполнения ниже к текущему определению. Заканчивает текущее определение, позволяя ему быть найденным в словаре, и вводит состояние интерпретации, потребляя colon-sys.

Последующие символы в области анализа типично представляют исходный текст на языке программирования, обычно некоторой формы ассемблера. Эти символы, обработанные определенным реализацией способом, производят соответствующий машинный код. Процесс продолжается, пополнением входного буфера по мере необходимости, до обработки определенной реализацией завершающей последовательности.

Время-выполнения: (--) (R: nest-sys --)
Заменяет семантику выполнения самого последнего определения на семантику выполнения name, данную ниже. Возвращает управление на вызывающее определение, определенное nest-sys. Неопределенная ситуация существует если самое последнее определение не было определено с помощью CREATE или определенного пользователем слова, которое вызывает CREATE.

name Выполнение: (i*x -- j*x)
Исполняет машинную последовательность кода, которая была сгенерирована после ;CODE.

См.: 6.1.1250 DOES>.

15.6.2.0702 AHEAD TOOLS EXT
Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (C: -- orig)
Помещает адрес ячейки новой неразрешенной ссылки вперед orig на стек потока управления. Добавляет семантику времени-выполнения, данную ниже к текущему определению. Семантика не завершена, пока orig не разрешена (например, с помощью THEN).

Время-выполнения: (--)
Продолжает выполнение с адреса ячейки, определенного разрешением orig.

15.6.2.0740 ASSEMBLER TOOLS EXT
(--)
Заменяет первый список слов в порядке поиска на список слов ASSEMBLER.

См.: 16. Дополнительный набор слов порядка поиска.

15.6.2.0830 BYE TOOLS EXT
(--)
Возвращает управление в базовую операционную систему, если существует.

15.6.2.0930 CODE TOOLS EXT
("<spaces>name" --)
Пропускает ведущие разделители пробелы. Выделяет name, ограниченное пробелом. Создает определение для name, обозначенное "code definition", с семантикой выполнения определенной ниже.

Последующие символы в области анализа типично представляют исходный текст на языке программирования, обычно некоторой формы ассемблера. Эти символы, обработанные определенным реализацией способом, производят соответствующий машинный код. Процесс продолжается, пополнением входного буфера по мере необходимости, до обработки определенной реализацией завершающей последовательности.

name Выполнение: (i*x -- j*x)
Исполняет машинную последовательность кода, которая была сгенерирована после CODE.

См.: 3.4.1 Синтаксический анализ.

15.6.2.1015 CS-PICK "c-s-pick" TOOLS EXT
Интерпретация: Семантика интерпретации для этого слова не определена.

Выполнение:(C: destu ... orig0|dest0 -- destu ... orig0|dest0 destu)
(S: u --)
Удаляет u. Копирует destu на вершину стека потока управления. Неопределенная ситуация существует, если есть меньше чем u+1 элементов, каждый из которых должен быть orig или dest, на стеке потока управления перед выполнением CS-PICK.

Если стек потока управления реализован с использованием стека данных, u должен быть самым верхним элементом на стеке данных.

15.6.2.1020 CS-ROLL "c-s-roll" TOOLS EXT
Интерпретация: Семантика интерпретации для этого слова не определена.

Выполнение: (C: origu|destu origu-1|destu-1 ... orig0|dest0 --
origu-1|destu-1 ... orig0|dest0 origu|destu)
(S: u --)
Удаляет u. Вращает u+1 элементов на вершине стека потока управления так, чтобы origu|destu находились на вершине стека потока управления. Неопределенная ситуация существует, если есть меньше чем u+1 элементов, каждый из которых должен быть orig или dest, на стеке потока управления перед выполнением CS-ROLL.

Если стек потока управления реализован с использованием стека данных, u должен быть самым верхним элементом на стеке данных.

15.6.2.1300 EDITOR TOOLS EXT
(--)
Заменяет первый список слов в порядке поиска на список слов EDITOR.

См.: 16. Дополнительный набор слов порядка поиска.

15.6.2.1580 FORGET TOOLS EXT
("<spaces>name" --)
Пропускает ведущие разделители пробелы. Выделяет name, ограниченное пробелом. Находит name, затем удаляет name из словаря наряду со всеми словами, добавленными в словарь после name. Неопределенная ситуация существует, если name не может быть найдено.

Если набор слов Порядка поиска присутствует, FORGET исследует список слов компиляции. Неопределенная ситуация существует если список слов компиляции удален.

Неопределенная ситуация существует, если FORGET, удаляет слово, требуемое для правильного выполнения.

Примечание: Это слово устаревшее и включено как уступка существующим реализациям.

См.: 3.4.1 Синтаксический анализ.

15.6.2.2250 STATE TOOLS EXT
(-- a-addr)
Расширьте семантику 6.1.2250 STATE, чтобы позволить ;CODE, изменить значение в STATE. Программа непосредственно не должна изменять содержимое STATE.

См.: 3.4 Интерпретатор текста Forth, 6.1.0450 :, 6.1.0460 ;, 6.1.0670 ABORT, 6.1.2050 QUIT, 6.1.2250 STATE, 6.1.2500 [, 6.1.2540], 6.2.0455 :NONAME, 15.6.2.0470 ;CODE.

15.6.2.2531 [ELSE] "bracket-else" TOOLS EXT
Компиляция: Исполняет семантику выполнения, данную ниже.

Выполнение: ("<spaces>name ... " --)
Пропускает ведущие пробелы, выделяет и снимает ограниченные пробелом слова из области анализа, включая вложенные ссылки [IF] ... [THEN] и [IF] ... [ELSE] ... [THEN], пока слово [THEN] не будет проанализировано и снято. Если область анализа становится исчерпанной, она пополняется как с помощью REFILL. [ELSE] - слово немедленного исполнения.

См.: 3.4.1 Синтаксический анализ.

15.6.2.2532 [IF] "bracket-if" TOOLS EXT

Компиляция: Исполняет семантику выполнения, данную ниже.

Выполнение: (flag | flag "<spaces>name ... " --)
Если flag - true, не делает ничего. Иначе, пропускает ведущие пробелы, выделяет и снимает ограниченные пробелом слова из области

анализа, включая вложенные ссылки [IF] ... [THEN] и [IF] ... [ELSE] ... [THEN], пока слово [ELSE] или слово [THEN] не будет проанализировано и снято. Если область анализа становится исчерпанной, она пополняется как с помощью REFILL. [IF] - слово немедленного исполнения.

Неопределенная ситуация существует, если [IF] - следствие выполнения POSTPONE, или если достигнут конец входного буфера и не может быть пополнен перед анализом ограничителей [ELSE] или [THEN].

15.6.2.2533 [THEN] "bracket-then" TOOLS EXT

Компиляция: Исполняет семантику выполнения, данную ниже.

Выполнение: (--)
Не делает ничего. [THEN] - слово немедленного исполнения.

16. Дополнительный набор слов порядка поиска

16.1 Введение

16.2 Дополнительные термины и нотация

Список слов компиляции: Список слов, в который помещаются новые имена определений.

Порядок поиска: Список списков слов, определяющих порядок, в котором будет просмотрен словарь.

16.3 Дополнительные условия применения

16.3.1 Типы данных

Идентификаторы списка слов - зависящие-от-реализации одна-ячейка значения, которые идентифицируют списки слов.

Добавьте таблицу 16.1 к таблице 3.1.

Таблица 16.1 - Типы данных

Символ	Тип данных	Размер на стеке
wid	идентификатор списка слов	1 ячейка

См.: 3.1 Типы данных, 3.4.2 Поиск имен определений, 3.4 Интерпретатор текста Forth.

16.3.2 Запросы к окружению

Добавьте таблицу 16.2 к таблице 3.5.

См.: 3.2.6 Запросы к окружению.

Таблица 16.2 - Строки запросов к окружению

Строка	Тип	Константа	Значение
SEARCH-ORDER	flag	no	набор слов порядка поиска присутствует
SEARCH-ORDER-EXT	flag	no	дополнительный набор слов порядка поиска присутствует
WORDLISTS	n	yes	максимальное число списков слов используемых в порядке поиска

16.3.3 Поиск имен определений

При просмотре списка слов для имени определения, система должна просматривать каждый список слов от его последнего определения до первого. Просмотр может охватывать только отдельный список слов, как с SEARCH-WORDLIST, или все списки слов в порядке поиска, как в текстовом интерпретаторе и FIND.

Изменение порядка поиска должно затрагивать только последующий поиск имен определений в словаре.

Система с набором слов Порядка поиска должна позволять, по крайней мере, восемь списков слов в порядке поиска.

Неопределенная ситуация существует, если программа изменяет список слов компиляции в течение компиляции определения или перед модификацией поведения последнего откомпилированного определения с помощью ;CODE, DOES>, или IMMEDIATE.

Программа, которая требует больше чем восемь списков слов в порядке поиска, имеет зависимость от окружения.

См.: 3.4.2 Поиск имен определений

16.3.4 Непрерывные области

Зоны области данных созданной операциями, описанными в 3.3.3.2 Непрерывные области, могут быть состоящими из нескольких несмежных участков, если WORDLIST выполнимо в пределах распределений.

16.4 Дополнительные документационные требования

16.4.1 Системная документация

16.4.1.1 Опции определенные реализацией

- максимальное число списков слов в порядке поиска (16.3.3 Поиск имен определений, 16.6.1.2197 SET-ORDER);
- минимальный порядок поиска (16.6.1.2197 SET-ORDER, 16.6.2.1965 ONLY).

16.4.1.2 Неопределенные ситуации

- изменение списка слов компиляции (16.3.3 Поиск имен определений);
- пустой порядок поиска (16.6.2.2037 PREVIOUS);
- слишком много списков слов в порядке поиска (16.6.2.0715 ALSO).

16.4.1.3 _____
Другая системная документация

- нет дополнительных требований.

16.4.2 _____
Программная документация

16.4.2.1 _____
Зависимости от окружения

- требование больше чем восьми списков слов в порядке поиска (16.3.3 Поиск имен определений).

16.4.2.2 _____
Другая программная документация

- нет дополнительных требований.

16.5 _____
Соглашение и наименование

16.5.1 _____
ANS Forth системы

Фраза "Предоставляет набор слов порядка поиска" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь набор слов порядка поиска.

Фраза "Предоставляет имена из расширения набора слов порядка поиска" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет часть расширения набора слов порядка поиска.

Фраза "Предоставляет расширение набора слов порядка поиска" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь набор слов порядка поиска и расширение набора слов порядка поиска.

16.5.2 _____
ANS Forth программы

Фраза "Требуется набор слов порядка поиска" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила набор слов порядка поиска.

Фраза "Требуются имена из расширения набора слов порядка поиска" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила часть расширения набора слов порядка поиска.

Фраза "Требуется расширение набора слов порядка поиска" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила весь набор слов порядка поиска и расширение набора слов порядка поиска.

16.6 _____
Словарь

16.6.1 _____
Слова порядка поиска

16.6.1.1180 DEFINITIONS SEARCH
(--)

Делает список слов компиляции таким же, как первый список слов в порядке поиска. Определяет, что имена последующих определений будут помещены в список слов компиляции. Последующие изменения в порядке поиска не будут затрагивать список слов компиляции.

См.: 16.3.3 Поиск имен определений.

16.6.1.1550 FIND SEARCH
Расширьте семантику 6.1.1550 FIND, таким образом:
(c-addr -- c-addr 0 | xt 1 | xt -1)
Ищет определение именованное в строке со счетчиком в c-addr. Если определение не найдено после поиска всех списков слов в порядке поиска, возвращает c-addr и нуль. Если определение найдено, возвращает xt.

Если определение немедленного исполнения, также возвращает один (1); иначе также возвращает минус-один (-1). Для данной строки, значения возвращенные FIND во время компиляции, могут отличаться от возвращенных при не компиляции.

См.: 3.4.2 Поиск имен определений, 6.1.0070 ', 6.1.1550 FIND, 6.1.2033 POSTPONE, 6.1.2510 ['], D.6.7 Immediate-ности.

16.6.1.1595 FORTH-WORDLIST SEARCH
(-- wid)
Возвращает wid, идентификатор списка слов, который включает все стандартные слова, предоставленные реализацией. Этот список слов - первоначально список слов компиляции и - часть начального порядка поиска.

16.6.1.1643 GET-CURRENT SEARCH
(-- wid)
Возвращает wid, идентификатор списка слов компиляции.

16.6.1.1647 GET-ORDER SEARCH
(-- widn ... wid1 n)
Возвращает число списков слов n в порядке поиска и идентификаторы списков слов widn ... wid1 идентифицирующие эти списки слов. wid1 идентифицирует список слов, который просматривается первым, и widn список слов, который просматривается последним. Порядок поиска не затронут.

16.6.1.2192 SEARCH-WORDLIST SEARCH
(c-addr u wid -- 0 | xt 1 | xt -1)
Ищет определение, идентифицированное строкой c-addr u в списке слов идентифицированном wid. Если определение не найдено, возвращает нуль. Если определение найдено, возвращает его идентификатор исполнения xt и один (1) если определение немедленного исполнения, и минус-один (-1) иначе.

16.6.1.2195 SET-CURRENT SEARCH
(wid --)
Устанавливает список слов компиляции на список слов, идентифицированный wid.

16.6.1.2197 SET-ORDER SEARCH
(widn ... wid1 n --)
Устанавливает порядок поиска на список слов, идентифицированный widn ... wid1. Впоследствии, список слов wid1 будет просматриваться первым, и список слов widn просматриваться последним. Если n нулевой, порядок поиска пуст. Если n - минус один, установит

порядок поиска на определенный реализацией минимальный порядок поиска. Минимальный порядок поиска должен включать слова FORTH-WORDLIST и SET-ORDER. Система должна позволять быть n по крайней мере восьми.

16.6.1.2460 WORDLIST SEARCH
(-- wid)
Создает новый пустой список слов, возвращая его идентификатор списка слов wid. Новый список слов может быть возвращен из пула предварительно размещенных списков слов или может быть динамически распределен в области данных. Система должна позволять создание, по крайней мере, 8 новых списков слов в дополнение к любым предоставленным как часть системы.

16.6.2 Расширения слов порядка поиска

16.6.2.0715 ALSO SEARCH EXT
(--)
Преобразует порядок поиска, состоящий из widn, ... wid2, wid1 (где wid1 просматривается первым) в widn, ... wid2, wid1, wid1. Неопределенная ситуация существует, если есть слишком много списков слов в порядке поиска.

16.6.2.1590 FORTH SEARCH EXT
(--)
Преобразует порядок поиска, состоящий из widn, ... wid2, wid1 (где wid1 просматривается первым) в widn, ... wid2, widFORTH-WORDLIST.

16.6.2.1965 ONLY SEARCH EXT
(--)
Устанавливает порядок поиска на определенный реализацией минимальный порядок поиска. Минимальный порядок поиска должен включать слова FORTH-WORDLIST и SET-ORDER.

16.6.2.1985 ORDER SEARCH EXT
(--)
Отображает списки слов в порядке поиска в их последовательности порядка поиска, от первого просматриваемого до последнего. Также отображает список слов, в который будут помещены новые определения. Формат отображения - зависящий от реализации.

ORDER может быть реализован с использованием слов вывода отображаемых чисел. Следовательно, их использование может разрушать временную область, идентифицированную #>.

См.: 3.3.3.6 Другие временные области.

16.6.2.2037 PREVIOUS SEARCH EXT
(--)
Преобразует порядок поиска, состоящий из widn, ... wid2, wid1 (где wid1 просматривается первым) в widn, ... wid2. Неопределенная ситуация существует, если порядок поиска был пуст прежде, чем было выполнено PREVIOUS.

17. Дополнительный строковый набор слов

17.1 Введение

17.2 Дополнительные термины и нотация

Нет.

17.3 Дополнительные условия применения

Добавьте таблицу 17.1 к таблице 3.5.

См.: 3.2.6 Запросы к окружению.

Таблица 17.1 - Строки запросов к окружению

Строка	Тип	Константа	Значение
STRING	flag	no	строковый набор слов присутствует
STRING-EXT	flag	no	дополнительный строковый набор слов присутствует

17.4 Дополнительные документационные требования

Нет.

17.5 Соглашение и наименование

17.5.1 ANS Forth программы

Фраза "Предоставляет строковый набор слов" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь строковый набор слов.

Фраза "Предоставляет имена из расширения строкового набора слов" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет часть расширения строкового набора слов.

Фраза "Предоставляет расширение строкового набора слов" должна быть добавлена к наименованию любой Стандартной системы, которая предоставляет весь строковый набор слов и расширение строкового набора слов.

17.5.2 ANS Forth программы

Фраза "Требуется строковый набор слов" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила строковый набор слов.

Фраза "Требуются имена из расширения строкового набора слов" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила часть расширения строкового набора слов.

Фраза "Требуется расширение строкового набора слов" должна быть добавлена к наименованию Стандартной программы, которая требует, чтобы система предоставила весь строковый набор слов и расширение строкового набора слов.

17.6 Словарь

17.6.1 Строковые слова

17.6.1.0170 -TRAILING "dash-trailing" STRING
(c-addr u1 -- c-addr u2)
Если u1 больше чем нуль, u2 равно u1 минус число пробелов в конце символьной строки, определенной c-addr u1. Если u1 нуль или вся строка состоит из пробелов, u2 нулевой.

17.6.1.0245 /STRING "slash-string" STRING
(c-addr1 u1 n -- c-addr2 u2)
Корректирует символьную строку c-addr1, n символами. Результирующая символьная строка, определенная c-addr2 u2, начинается с c-addr1 плюс n символов, и длиной u1 минус n символов.

17.6.1.0780 BLANK STRING
(c-addr u --)
Если u больше чем нуль, сохраняет символьное значение пробела в u последовательных символьных позициях, начинающихся с c-addr.

17.6.1.0910 CMOVE "c-move" STRING
(c-addr1 c-addr2 u --)
Если u больше чем нуль, копирует u последовательных символов из области данных начинающейся с c-addr1 в область, начинающуюся с c-addr2, выполняя посимвольно с наименьших адресов к наибольшим.

Отличие с: 17.6.1.0920 CMOVE>.

17.6.1.0920 CMOVE> "c-move-up" STRING
(c-addr1 c-addr2 u --)
Если u больше чем нуль, копирует u последовательных символов из области данных начинающейся с c-addr1 в область, начинающуюся с c-addr2, выполняя посимвольно с наибольших адресов к наименьшим.

Отличие с: 17.6.1.0910 CMOVE.

17.6.1.0935 COMPARE STRING
(c-addr1 u1 c-addr2 u2 -- n)
Сравнивает строку, определенную c-addr1 u1 со строкой, определенной c-addr2 u2. Строки сравниваются, начиная с заданных адресов, посимвольно, до длины более короткой строки или до нахождения различия. Если две строки идентичны - n нулевое. Если две строки идентичны до длины более короткой строки, n равно минус-один (-1) если u1 - меньше чем u2, иначе один (1). Если две строки - не идентичны до длины более короткой строки, n равно минус-один (-1) если первый несоответствующий символ в строке определенной c-addr1 u1 имеет меньшее числовое значение чем соответствующий символ в строке, определенной c-addr2 u2, иначе один (1).

17.6.1.2191 SEARCH STRING
(c-addr1 u1 c-addr2 u2 -- c-addr3 u3 flag)
Ищет строку, определенную c-addr1 u1 в строке, определенной c-addr2 u2. Если flag - true, соответствие было найдено с c-addr3 и с u3 оставшимися символами. Если flag - false не было никакого соответствия и c-addr3 равен c-addr1, и u3 равно u1.

17.6.1.2212 SLITERAL STRING

Интерпретация: Семантика интерпретации для этого слова не определена.

Компиляция: (c-addr1 u --)
Добавляет семантику времени-выполнения, данную ниже к текущему определению.

Время-выполнения: (-- c-addr2 u)
Возвращает c-addr2 u описание строки, состоящей из символов определенных c-addr1 u в течение компиляции. Программа не должна изменять возвращенную строку.

17.6.2 Расширения строковых слов

Нет

A. Разъяснения (информационное приложение)

A.1 Введение

A.1.1 Цель

A.1.2 Назначение

Этот Стандарт более обширен, чем предыдущие промышленные стандарты для языка Forth. Несколько факторов сделали это необходимым:

- желание разрешить конфликты между предыдущими стандартами;
- потребность устранить семантические двусмысленности и другие несоответствия;
- требование стандартизировать обычную практику, где возможно разрешение расхождений способом, который минимизирует стоимость соглашения;
- желание стандартизировать общие системные методы, включая соответствующие аппаратным средствам.

Результат усилий удовлетворения всех этих требований - Стандарт устроенный так, чтобы требуемый набор слов остался маленьким. Таким образом, ANS Forth может быть предоставлен для ресурсо-ограниченных встраиваемых систем. Слова вне требуемого набора слов организованы во множество дополнительных наборов слов и их расширений, допуская реализацию адаптированных систем, которые являются Стандартными.

При оценке относительных достоинств, члены X3J14 Технического Комитета руководствовались следующими целями (перечисленными в алфавитном порядке):

Последовательность
Стандарт предоставляет функционально законченный набор слов с минимальным функциональным перекрытием.

Стоимость соглашения
Эта цель включает такие проблемы как обычная практика, сколько существующего кода было бы сломано предложенным изменением, и количество усилий требуемых для приведения существующих приложений и систем в соответствие со Стандартом.

Эффективность
Скорость выполнения, компактность памяти.

Переносимость
Слова, выбранные для включения должны быть свободны от зависящих от системы функций.

Удобочитаемость
Имена определений Forth должны ясно очерчивать их поведение. Это поведение должно иметь очевидную простоту, которая обеспечивает быстрое понимание. Forth должен быть легко познаваем, и обеспечивать легко поддерживаемый код.

Полезность
Считается, имеет достаточно существенные функциональные возможности и частоту использования, чтобы считать подходящим для включения.

A.1.3 Организация Документа

A.1.3.1 Наборы слов

С начала, X3J14 Технический Комитет столкнулся не только с противоречивыми идеями относительно того каким является "реальный" Forth, но также и противоречивыми потребностями различных групп в пределах Forth сообщества. В одной крайности были те, кто требовали "пустого" Forth. В другой крайности были те, кто хотели "жирного" Forth. Многие были где-то по середине. Все были убеждены в справедливости их собственной позиции и не правильности, по крайней мере, одной из этих двух крайностей. Состав комитета отразил этот полный диапазон интересов.

Подход, который мы приняли, состоит в определении Основного набора слов устанавливающего самый нижний предел требуемой системной функциональности, и предоставлении портфеля дополнительных наборов слов для специальных целей. Этот простой подход соответствует фундаментальной природе Forth как расширяемого языка, и таким образом достигает своего рода мета-расширяемости.

В этом ключе, высокоуровневый компромисс, независимо от фактического состава индивидуального набора слов, на долгий срок установлена твердая и реальная структура. Могут соглашаться или не соглашаться, что должен быть набор слов Locals, или что слово COMPILE, принадлежит расширению основного набора слов. Но, по крайней мере, есть механизм посредством которого такие вещи могут быть включены логичным и организованным способом.

Важно несколько следствий этой схемы дополнительных наборов слов.

Первое, ANS Forth системы могут по-прежнему быть реализованы на большем диапазоне аппаратных средств, чем может быть заявлено почти любым другим отдельным языком. Так как требуется только Основной набор слов, даже очень ограниченные аппаратные средства будут способны вмещать ANS Forth реализацию.

Второе, ожидается большая степень переносимости приложений, и программистов. Дополнительные наборы слов стандартизируют различные функции (например, с плавающей точкой), это было широко реализовано прежде, но не с однородными именами определений и методологий, ни с равными уровнями полноты. С такими словами, теперь стандартизированными в дополнительные наборы слов, связь между программистами - устно, через журнал или газетные статьи, и т.д. - прыгнут к новому уровню возможностей, и совместноиспользуемость кода и приложений должна повыситься впечатляюще.

Третье, ANS Forth системы могут быть разработаны, чтобы предложить пользователю возможность выборочно, даже динамически, включать или исключать один или большее количество дополнительных наборов слов или их частей. Также, более дешевые изделия могут быть предложены пользователю, который нуждается в

Основном наборе слов и не более. Таким образом, пользователю будет доступна фактически неограниченная гибкость.

Но эти преимущества имеют цену. Бремя находится на пользователе, чтобы решить какие свойства желательны, и соответственно выбирать предложенные изделия, особенно когда важна переносимость приложения. Мы не ожидаем от большинства разработчиков попыток предоставлять все наборы слов, а скорее выбирать наиболее ценные для их рыночного предназначения.

Основное требование - это, если разработчик утверждает что имеет частичный дополнительный набор слов, должна быть доступна полная требуемая часть этого набора слов. Если разработчик желает предложить только часть дополнительного набора слов, приемлемо говорить, например "Эта система предлагает части [названного] набора слов", особенно если выбранные или исключенные слова перечислены ясно.

Каждый дополнительный набор слов будет вероятно апеллировать к специфической клиентуре. Например, ученые, выполняющие сложные математические анализы могут возлагать большее значение на набор слов с плавающей точкой, чем программисты разрабатывающие простые внедряемые контроллеры. Как в случае основных расширений мы ожидаем, что разработчики предложат те наборы слов, которые по их мнению будут оценены их пользователями.

Дополнительные наборы слов можно предлагать в исходной форме или иначе факторизованными так, чтобы пользователь мог выборочно их загружать.

Расширения к дополнительным наборам слов включают слова, которые считаются менее обязательными для выполнения основной деятельности поддерживаемой набором слов, хотя, несомненно, для этого уместны. Как в случае основных расширений, разработчики могут выборочно добавлять описанные подмножества расширения набора слов, предусматривая наименование не вводящее в заблуждение пользователя относительно мысли, что все слова присутствуют.

A.2 Термины и нотация

A.2.1 Определения терминов

неопределенная ситуация
Реакция Стандартной системы на неопределенную ситуацию оставлена на усмотрение разработчика. Стандартная система не нуждается в явном обнаружении или сообщении о случаях неопределенных ситуаций.

кросс компилятор
Кросс компиляторы могут использоваться для подготовки программы к выполнению во внедренной системе, или могут использоваться для генерации ядра Forth для той же самой или другой среды времени-выполнения.

поле данных
В более ранних стандартах, поля данных были известны как "поля параметров".

В Forth системах с подпрограммным шитым кодом, все - объектный код. Нет традиционных полей кода или данных. Только слово, определенное с помощью CREATE или словом, которое вызывает CREATE, имеет поле данных. Только поле данных, определенное через CREATE, может обрабатываться переносимо.

набор слов
Этот Стандарт признает, что некоторые функции, хотя и полезны в некоторых прикладных областях, не достаточно общие, чтобы оправдывать требование их во всех системах Forth. Но они являются полезными для группирования слов Forth по

родственным функциям. Эти вопросы имеют дело с использованием концепции наборов слов.

"Основной" набор слов содержит неотъемлемое тело слов в Forth системе. Только этот - "требуемый" набор слов. Другие наборы слов определенные в этом Стандарте – это необязательные добавления делающие возможным предоставление Стандартной системы с приспособленными уровнями функциональности.

A.2.2 _____
Нотация

A.2.2.2 _____
Стековая нотация

Использование -sys, orig, и dest типов данных в диаграммах стекового эффекта передает две части информации. Первое, это предупреждает читателя, что много реализаций используют стек данных не установленным способом для их целей, так что элементы ниже стеков потока управления или данных являются недоступными. Второе, в случаях, где используются orig и dest, документированы явные правила спаривания при допущении, что все системы осуществят такую модель, чтобы ее результат был эквивалентен применению некоторого стека, и фактически многие реализации используют стек данных для этой цели. Однако ничто в этом Стандарте не требует, чтобы реализации фактически использовали стек данных (или любой другой) для этой цели, при условии, что поддержано подразумеваемое поведение модели.

A.3 _____
Условия применения

Системы Forth необычно просты для развития, по сравнению с компиляторами для более традиционных языков типа C. В дополнение к системам Forth, поддерживаемым производителями, общедоступные реализации и руководства по реализации были широко доступны почти двадцать лет, и большое количество людей разработали их собственные Forth системы. В результате, сформировалось разнообразие подходов реализации, каждый оптимизированный для специфической платформы или конкретного рынка.

X3J14 Технический Комитет пытался приспособить это разнообразие, сдерживая разработчиков так мало насколько это возможно, совместимо с целью определения стандартного интерфейса между основной Системой Forth и прикладной программой, разрабатываемой на нем.

Так же мы не будем ручаться, в этом разделе сообщать Вам, как реализовать Forth систему, а скорее предусмотреть некоторое руководство относительно того, какие минимальные требования имеются для системы, которая может должным образом претендовать на соответствие этому Стандарту.

A.3.1 Типы данных

Большинство компьютеров имеет дело с произвольными наборами двоичных разрядов. Нет никакого способа установить контроль, содержит ли ячейка адрес или целое число без знака. Единственное значение, которым данная величина обладает - это значение, назначенное приложением.

Когда данные используются, значение результата зависит от значения, назначенного для входных величин. Некоторые комбинации входных значений производят бессмысленный результат: например, какое значение может быть назначено для арифметической суммы ASCII представления символа "A" и TRUE флага? Ответ может быть "никакое значение"; или альтернативно, эта операция могла бы быть первым шагом в создании контрольной суммы. Контекст - определяющий.

Порядок ограничения значений, которые программа может назначить различным комбинациям наборов двоичных разрядов - иногда называется типизация данных. Много машинных языков устанавливают явную типизацию данных, и имеются компиляторы, которые предотвращают неточные операции.

Forth редко явно налагает ограничения типа данных. Однако типы данных неявно существуют, и требуется дисциплина, особенно если переносимость программ - цель. В Forth, возложено на программиста (скорее чем на компилятор) определение того, что данные точно типизированы.

Этот раздел пытается предлагать руководство относительно фактической типизации данных в Forth.

A.3.1.2 Символьные типы

Forth не принуждает к применению типа данных посредством глубины стека, для правильной идентификации и надлежащей манипуляции символьным типом данных. Символы не обязательно занимают значимыми данными всю ширину их единственного элемента стека. В то время как различие между знаковым и без знаковым символом полностью отсутствует в формальной спецификации Forth, практическая тенденция состоит в том, чтобы рассматривать символы как короткие положительные целые числа, когда входят в силу математические операции.

а) Стандартный набор символов

1) Элемент памяти для символьного типа данных (C@, C!, FILL, и т.д.) должен быть способен содержать числа без знака от 0 до 255.

2) Реализация не требует ограничения символьной памяти этим диапазоном, но стандартная программа без зависимостей от окружения не должна быть способна сохранять числа вне этого диапазона в "char" адресе ячейки.

3) Разрешенные представления чисел - дополнительный код (дополнение до двух), обратный код (дополнение до единицы), и знаковая величина. Заметьте, что все эти системы счисления сходятся в представлении положительных чисел.

4) Так как "char" может хранить маленькие положительные числа и так как символьный тип данных - это под диапазон типа данных целого числа без знака, C! должен хранить n младших бит ячейки (8 <= n <= биты/ячейка). Учитывая диапазон допустимых представлений числа и их известное кодирование, "TRUE xx C! xx C@" должно оставить элемент стека с некоторым набором битов, который будет принят словом IF как ненулевой.

5) Для целей ввода (KEY, ACCEPT, и т.д.) и вывода (EMIT, TYPE, и т.д.), кодирование между числами и удобочитаемыми символами - это ISO646/IRV (ASCII) в пределах диапазона от 32 до 126 (пробел в ~). EBCDIC - исключен (большинство компьютерных систем "EBCDIC" также поддерживает ASCII). Вне этого диапазона, это - на уровне реализации. Очевидный выбор реализации это использовать управляющие символы ASCII для диапазона от 0 до 31, по крайней мере, для "отображаемых" символов в этом диапазоне (TAB, RETURN, LINEFEED, FORMFEED). Однако это - не столь же ясно, как это может казаться, из-за различий между операционными системами в трактовке этих символов. Например, в некоторых системах TAB по 4 символьным границам, в других по 8 символьным границам, и в иных по предварительно устанавливаемым табуляторам. Некоторые системы исполняют автоматический перевод строки после перевода каретки, другие исполняют автоматический перевод каретки после перевода строки, и другие не делают ни чего.

Коды от 128 до 255 могут, в конечном счете, быть стандартизированы формально

или неформально для использования как интернациональных символов, типа символов с диакритическими знаками, находимых во многих Европейских языках. Одно такое кодирование - 8-bit ISO Latin-1 набор символов. Компьютерный рынок в целом, в конечном счете, решит, какой набор кодирования этих символов будет преобладать. Для реализаций Forth выполняющихся под операционной системой (большинство их выполняется на стандартных платформах этих дней), большинство разработчиков Forth будет вероятно делать выбор не зависимо от того что делает система, без выполнения какого либо перекодирования в пределах самой системы Forth.

6) Стандартная программа может зависеть от способности получать любой символ в диапазоне 32 ... 126 посредством KEY, и подобно отображать тот же самый набор символов с помощью EMIT. Если программа должна мочь получать или отображать любой специфический символ вне этого диапазона, она может объявить зависимость от окружения на способность получать или отображать этот символ.

7) Стандартная программа не может использовать управляющие символы в именах определений. Однако Стандартная система не требует устанавливать это запрещение. Таким образом, существующие системы, которые в настоящее время позволяют управляющие символы в именах слов из BLOCK источника, могут продолжать позволять их, и программы, выполняющиеся на этих системах, продолжат работать. В источнике текстового файла, действие синтаксического анализа с пробелом как разделителем (например, BL WORD) обрабатывает управляющие символы так же как пробелы. Это эффективно подразумевает, что Вы не можете использовать управляющие символы в именах определений из источника текстового файла, так как текстовый интерпретатор обработает управляющие символы как разделители. Заметьте, что это "сворачивание управляющего символа" применяется только когда разделитель пробел, таким образом фраза "CHAR) WORD" может забирать строку содержащую управляющие символы.

b) Хранение и поиск

Символы передаются со стека данных в память с помощью C! и из памяти на стек данных с помощью C@. Число младших значащих бит эквивалентное звисящей-от-реализации ширине символа передается из вытолкнутого элемента стека данных в адрес выполнением C! без воздействия на любые биты в адресе назначения, которые могут включать старшую значащую часть ячейки; однако действие C@ очищает все биты старшей значащей части элемента стека данных, который оно туда помещает - вне зависящей-от-реализациой ширины символа (который может включать информацию дисплея определенную реализацией в битах старшей значащей части). Программист должен иметь в виду, что использование с произвольными элементами стека слов предназначенных для символьного типа данных может приводить к усечению таких данных.

c) Манипуляции на стеке

В дополнение к C@ и C!, символы перемещаются на стек, из стека и на стеке данных следующими словами:

```
>R  ?DUP  DROP  DUP  OVER  PICK  R>  R@  ROLL  ROT  SWAP
```

d) Дополнительные операции

Следующие математические операторы допустимы для символьных данных:

```
+  -  *  /  /MOD  MOD
```

Следующие сравнения и поразрядные операторы могут быть допустимы для символов, имеющих в виду, что отображаемая информация, кэшируемая в самых старших битах символов определенным реализацией способом, может быть замаскирована или иначе имеет дело с:

AND OR > < U> U< = <> 0= 0<> MAX MIN

LSHIFT RSHIFT

A.3.1.3 _____
Типы одна-ячейка

Элемент стека одна-ячейка, рассматриваемый без отношения к типизации - фундаментальный тип данных Forth. Все другие типы данных фактически представлены одним или более элементами стека одна-ячейка.

a) Хранение и поиск

Данные одна-ячейка передаются со стека в память посредством !; из памяти на стек посредством @. Все биты передаются в обоих направлениях, и никакой контроль соответствия типов любой разновидности не выполняется; также Стандартная система не проверяет что адрес памяти, используемый ! или @ должным образом выровнен или имеет должный размер для размещения данной величины, таким способом передаваемой.

b) Манипуляции на стеке

Вот выбор наиболее важных слов, которые перемещают данные одна-ячейка на стек, из стека и на стеке данных:

! @ >R ?DUP DROP DUP OVER PICK R> R@ ROLL ROT SWAP

c) Операторы сравнения

Следующие операторы сравнения универсально допустимы для одной или более одна-ячеек:

= <> 0= 0<>

A.3.1.3.1 _____
Флаги

FALSE флаг - величина данных одна-ячейка со всеми сброшенными битами, и TRUE флаг - величина данных одна-ячейка со всеми установленными битами. Там где слова Forth, которые проверяют флаги, принимают любой ненулевой битовый образец как true, там существует понятие правильно построенного флага. Если операция, чей результат должен использоваться как флаг, может производить любую битовую маску иную, чем TRUE или FALSE, рекомендуемый порядок состоит в преобразовании результата к правильно построенному флагу посредством слова Forth 0<> так, чтобы результат любых последующих логических операций на флаге был предсказуемый.

В дополнение к словам, которые перемещают, выбирают и сохраняют элементы одна-ячейка, следующие слова допустимы для операций на одном или более флаговых данных, находящихся на стеке данных:

AND OR XOR INVERT

A.3.1.3.2 _____
Целые числа

Величина данных одна-ячейка может быть обработана Стандартной программой как целое число со знаком. Перемещение и сохранение таких данных выполняется как для любых данных одна-ячейка. В дополнение к универсально применимым операторам для данных одна-ячейка, определенным выше, следующие математические и операторы сравнения допустимы для целых чисел со знаком одна-ячейка:

```
*  */  */MOD  /MOD  MOD  +  +!  -  /  1+  1-  ABS  MAX  MIN
NEGATE  0<  0>  <  >
```


Учитывая одно и то же число битов, целые числа без знака обычно представляют вдвое большую величину абсолютного значения, представимую целыми числами со знаком.

Величина данных одна-ячейка может быть обработана Стандартной программой как целое число без знака. Перемещение и сохранение таких данных выполняется как для любых данных одна-ячейка. Кроме того, следующие математические и операторы сравнения допустимы для целых чисел без знака одна-ячейка:

```
UM*  UM/MOD  +  +!  -  1+  1-  *  U<  U>
```

A.3.1.3.3 Адреса

Адрес уникально представлен как число без знака одна-ячейка и может быть обработан как такой когда перемещается на стек, из стека или на стеке. И наоборот, каждое число без знака представляет уникальный адрес (который - не обязательно адрес доступной памяти). Это один к одному отношение между адресами и числами без знака устанавливает эквивалентность между адресной арифметикой и соответствующими операциями на числах без знака.

Несколько операторов предусмотрено специально для адресной арифметики:

```
CHAR+  CHARS  CELL+  CELLS
```

и, если набор слов с плавающей точкой присутствует:

```
FLOAT+  FLOATS  SFLOAT+  SFLOATS  DFLOAT+  DFLOATS
```

Стандартная программа может никогда не предполагать специфическое соответствие между адресом Forth и физическим адресом, на который он отображен.

A.3.1.3.4 Строки со счетчиком

Тенденция в ANS Forth должна переместиться к непротиворечивому использованию "c-addr u" представления строк на стеке. Использование альтернативного представления стека "адрес строки со счетчиком" не одобряется. Традиционные слова Forth WORD и FIND продолжают использовать представление "адрес строки со счетчиком" по историческим причинам. Новое слово C", добавленное как помощь перенесения для существующих программ, также использует представление строки со счетчиком.

Строки со счетчиком остаются полезными как способ сохранить строки в памяти. Это использование - не запрещается, но когда ссылки к таким строкам появляются на стеке, предпочтительно использовать "c-addr u" представление.

A.3.1.3.5 Идентификаторы исполнения

Ассоциация между идентификатором исполнения и определением статическая. Однажды сделанная, она не изменяется с изменениями в порядке поиска или чем-нибудь еще. Однако она не может быть уникальна, например, фразы

```
' 1+
```

и

```
' CHAR+
```

могут возвращать одно и то же значение.

3.1.4 _____
Типы пара-ячеек

a) Хранение и поиск

Чтобы выбирать и сохранять пары-ячеек, предусмотрены два оператора:

```
2@  2!
```

b) Манипуляции на стеке

Дополнительно, эти операторы могут использоваться для перемещения пары-ячеек из стека, на стек и на стеке:

```
2>R  2DROP  2DUP  2OVER  2R>  2SWAP  2ROT
```

c) Сравнение

Следующие операции сравнения универсально допустимы для пары-ячеек:

```
D=  D0=
```

A.3.1.4.1 _____
Целые числа две-ячейки

Если целое число две-ячейки должно быть обработано как знаковое, допустимы следующие операции сравнения и математические операции:

```
D+  D-  D<  D0<  DABS  DMAX  DMIN  DNEGATE  M*/  M+
```

Если целое число две-ячейки должно быть обработано как без знаковое, допустимы следующие операции сравнения и математические операции:

```
D+  D-  UM/MOD  DU<
```

A.3.1.4.2 _____
Символьные строки

См.: 3.1.3.4 Строки со счетчиком.

A.3.2 _____
Среда Реализации

A.3.2.1 _____
Числа

Традиционно, Forth был реализован на машинах с дополнительным кодом (дополнение до двух), где есть взаимно-однозначное соответствие чисел со знаком и чисел без знака - любой элемент одна-ячейка может рассматриваться или как знаковое или без знаковое число. Действительно, знаковое представление любого положительного числа идентично эквивалентному представлению без знака. Далее, адреса обрабатываются как числа без знака: нет никакого отдельного указателя типа.

Арифметика, предписываемая на машинах с дополнительным кодом позволяет

выполнять + и - со знаковыми и без знаковыми числами. Это арифметическое поведение глубоко внедрено в обычную практику Forth. Как следствие этих поведений, возможные диапазоны знаковых и без знаковых чисел, для реализаций базирующихся на каждой из допустимых арифметических архитектур, таковы:

Арифметическая архитектура	Числа со знаком	Числа без знака
Дополнительный код	-n-1 to n	0 to 2n+1
Обратный код	-n to n	0 to n
Величина со знаком	-n to n	0 to n

где n - самое большое положительное число со знаком. Для всех трех архитектур, знаковые числа в диапазоне от 0 до n поразрядно идентичны соответствующим числам без знака. Заметьте, что числа без знака на машине с арифметикой величины со знаком эквивалентны знаковым не отрицательным числам как следствие необходимого соответствия между адресами и числами без знака и требуемым поведением + и -.

Для ссылок, эти представления чисел могут быть определены способом, в котором применено NEGATE:

```
Дополнительный код:       : NEGATE  INVERT 1+ ;
Обратный код:             : NEGATE  INVERT ;
Величина со знаком:       : NEGATE  HIGH-BIT XOR ;
```

где HIGH-BIT - битовая маска только на самый старший бит. Заметьте, что все эти системы счисления соглашаются в представлении неотрицательных чисел.

Посредством 3.2.1.1 Внутреннее представление числа и 6.1.0270 0=, разработчик должен гарантировать, что никакое стандартное или поддерживаемое слово не возвращает отрицательный нуль для любого числового (не булевого или флагового) результата. Иначе будет нарушено много существующих программистских допущений.

Не требуется реализовывать ни круговую без знаковую арифметику, ни устанавливать диапазон чисел без знака на полный размер ячейки. Есть исторический прецедент для ограничения диапазона u величиной +n, который является допустимым, когда размер ячейки больше 16 бит.

A.3.2.1.2 _____
Преобразование цифр

Например, реализация могла бы преобразовывать символы с "a" по "z" тождественно в символы с "A" по "Z", или она могла бы обрабатывать символы с " [" по "~" как дополнительные цифры с десятичной величиной с 36 по 71, соответственно.

A.3.2.2 _____
Арифметика

A.3.2.2.1 _____
Целочисленное деление

Forth-79 Стандарт определяет что знаковые операторы деления (/, /MOD, MOD, */MOD, и */) округляют частные нецелого числа к нулю (симметричное деление). Forth 83 изменяет семантику этих операторов на округление к отрицательной бесконечности (минимальное деление).

Некоторые в семействе Forth отказались преобразовывать системы и приложения из Forth-79 деления в Forth-83 деление. Чтобы разрешить эту проблему, ANS Forth системе разрешается поставлять или минимальные или симметричные операторы.

Кроме того, ANS Forth системы должны предоставлять примитив минимального деления (FM/MOD), примитив симметричного деления (SM/REM), и оператор умножения смешанной точности (M*).

Этот компромисс защищает инвестиции, сделанные в текущие приложения Forth; Forth-79 и Forth-83 программы автоматически совместимы с ANS Forth относительно деления. На практике, направление округления редко имеет значение для приложений. Однако если программа требует специфического направления округления, она может использовать примитив минимального деления FM/MOD или примитив симметричного деления SM/REM, для создания оператора деления желательной разновидности. Эта простая методика может использоваться для преобразования Forth-79 и Forth-83 программ в ANS Forth, без какого либо анализа исходных программ.

A.3.2.2.2 _____
Другие целочисленные операции

Происходит ли антипереполнение, зависит от типа данных результата. Например, фраза 1 2 - дает антипереполнение, если результат без знака и производит допустимый знаковый результат -1.

A.3.2.3 _____
Стеки

Единственный тип данных в Forth, который скорее имеется реально, чем существует абстрактно - это вход стека. Даже эта примитивная типизация Forth предписывается только жесткой действительностью антипереполнения или переполнения стека. Программист должен иметь ясное представление относительно количества стековых входов, которые будут использованы выполнением слова и количества входов, которые будут помещены назад на стек выполнением слова. Наблюдение аномальных случаев на стеке данных - первая линия обороны, посредством чего программист может распознать ошибки в прикладной программе. Также стоит помнить, что множество ошибок стека вызванных ошибочным прикладным кодом, имеют часто равную и противоположную величину, вызывая комплиментарный (и вводящий в заблуждение) результат.

По этим причинам и массе других причин, одна недвусмысленная, бесспорная, и необходимая дисциплина программирования, наблюдаемая начиная с самых ранних дней Forth - это обеспечение стековых диаграмм для всех добавлений к прикладному словарю за исключением статических конструкций типа VARIABLE и CONSTANT.

A.3.2.3.2 _____
Стек потока управления

Самое простое использование слов потока управления это реализовать базовые управляющие структуры, показанные на рисунке A.1.

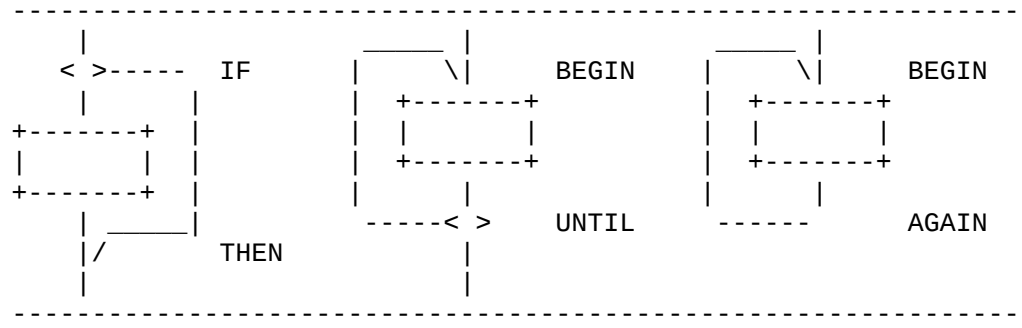


Рисунок A.1 - Базовые образцы потока управления.

В потоке управления каждый переход или передача управления должны заканчиваться в некотором месте назначения. Естественная реализация использует стек, чтобы помнить начала переходов вперед и назначения ветвлений назад. Как минимум, только местоположение каждого начала или назначения должно быть обозначено, хотя другая зависящая-от-реализации информация также может быть сохранена.

Начало - это непосредственно местоположение перехода. Назначение - то, где управление будет продолжено, если переход был принят. Назначение необходимо для разрешения адреса перехода для каждого начала, и наоборот, если каждая цепочка потока управления завершена, не может остаться никакое неиспользованное назначение.

С добавлением только трех слов (AHEAD, CS-ROLL и CS-PICK), базовые слова потока управления поставляют примитивы, необходимые для компиляции разнообразных переносимых структур управления. Требуемые способности - компиляция прямого и обратного условного и безусловного переходов и организация во время компиляции начал и назначений переходов. Таблица 1 показывает желаемое поведение.

Требование, чтобы слова потока управления были должным образом сбалансированы другими словами потока управления, делает разумным описание определенного реализацией стека потока управления во время компиляции. Нет никакого предписания относительно того, как стек потока управления реализован, например, стек данных, связанный список, специальный массив. Каждый элемент упомянутого выше стека потока управления - одинакового размера.

Таблица A.1 - Поведение компиляции слов потока управления

во время компиляции, слово:	поставки:	разрешения:	используется для:
IF	orig		отмечает начало условного перехода вперед
THEN		orig	разрешает IF или AHEAD
BEGIN	dest		отмечает назначение назад
AGAIN		dest	разрешает безусловный переход назад
UNTIL		dest	разрешает условный переход назад
AHEAD	orig		отмечает начало безусловного перехода вперед
CS-PICK			копирует элемент на стеке потока управления
CS-ROLL			переупорядочивает элементы на стеке потока управления

С этим инструментарием могут быть определены оставшиеся базовые элементы управляющих структур, показанные на рисунке A.2; стековая нотация, используемая здесь для слов немедленного исполнения - это (компиляция / выполнение).

```
: WHILE  ( dest -- orig dest / flag -- )
  \ условный выход из циклов
  POSTPONE IF      \ условный переход вперед
  1 CS-ROLL        \ сохранить dest на верху
; IMMEDIATE
```



```
: REPEAT  ( orig dest -- / -- )
  \ разрешение единственного WHILE  и возврат к BEGIN
  POSTPONE AGAIN   \ безусл. переход назад к dest
  POSTPONE THEN    \ разрешение перехода вперед из orig
; IMMEDIATE

: ELSE  ( orig1 -- orig2 / -- )
  \ разрешение IF обеспечивающего альтернативное выполнение
  POSTPONE AHEAD   \ безусловный переход вперед orig2
```

```
  1 CS-ROLL        \ положить orig1 снова наверх
  POSTPONE THEN    \ разрешение перехода вперед из orig1
; IMMEDIATE
```

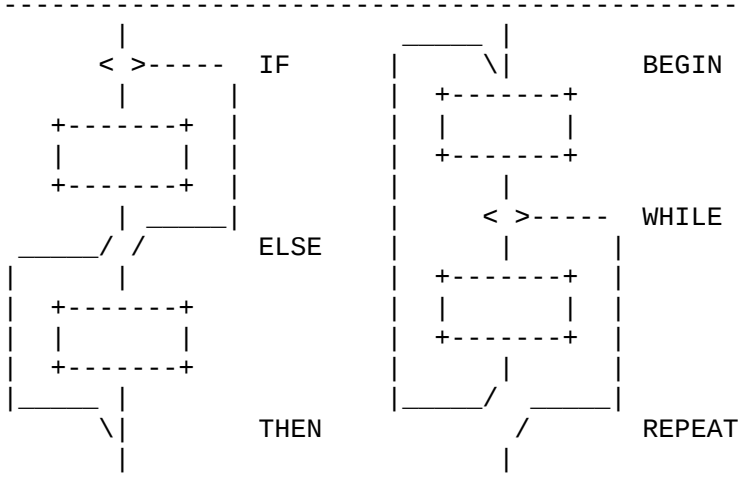
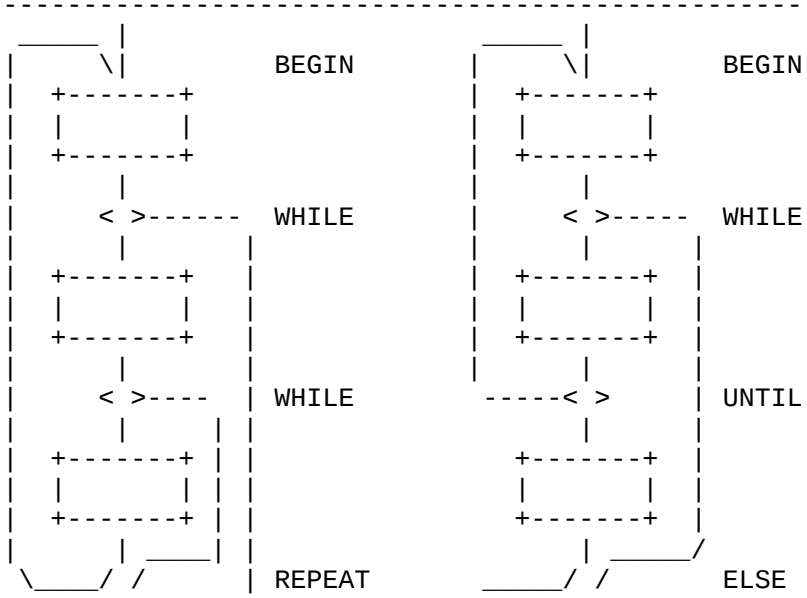


Рисунок A.2 - Дополнительные базовые образцы потока управления.

Поток управления Forth предоставляет решение для известной проблематики строго структурированного программирования.

Базовые управляющие структуры могут быть дополнены, как показано в примерах на рисунке A.3, с дополнительными WHILE в BEGIN ... UNTIL и BEGIN ... WHILE ... REPEAT структурах. Однако, для каждого дополнительного, WHILE должен быть THEN в конце структуры. THEN завершает синтаксис с WHILE и указывает где продолжить выполнение, когда WHILE передаст управление. Использование больше чем одного дополнительного WHILE возможно, но не общепринято. Заметьте, что если пользователь находит это использование THEN нежелательным, может быть определен псевдоним с более приятным именем.

Дополнительные действия могут быть выполнены между словом потока управления (REPEAT или UNTIL) и THEN, которое соответствует дополнительному WHILE. Далее, если для нормального завершения и раннего завершения желательны дополнительные действия, альтернативные действия могут быть разделены обычным Forth ELSE. Действия завершения - все определены после тела цикла.



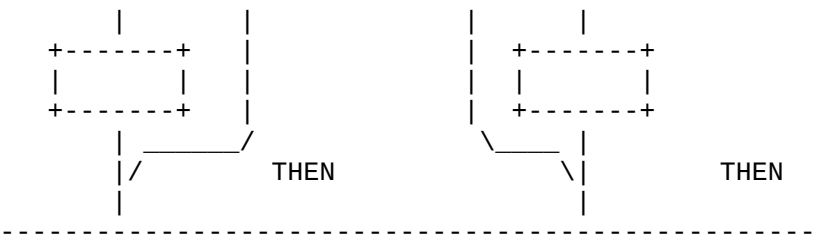


Рисунок A.3 - Примеры расширения образцов потока управления.

Заметьте, что REPEAT создает аномалию при сопоставлении WHILE с ELSE или THEN, более примечательную по сравнению с BEGIN...UNTIL случаем. То есть будет меньше одним ELSE или THEN чем есть WHILE, потому что REPEAT разрешает один THEN. Как и выше, если пользователь находит это число несоответствий нежелательным, REPEAT могло быть заменено его собственным действующим определением.

Могут быть определены другие слова потока управления выхода из цикла, и даже другие циклы. Единственное требование - это что стек потока управления должным образом обслуживается и управляется.

Простая реализация ANS Forth CASE структуры ниже - пример расширения управляющей структуры. Заметьте, обслуживание стека данных не допускает вмешательство со стороны возможного использования стека потока управления.

```
0 CONSTANT CASE IMMEDIATE  ( иниц. счетчика OF )

: OF  ( #of -- orig #of+1 / x -- )
  1+    ( счетчик OF )
  >R    ( переместить со стека в случае, если )
        ( стек потока управления это стек данных. )
  POSTPONE OVER  POSTPONE = ( копир. и тест параметра case )
  POSTPONE IF    ( добавить orig на стек потока управления )
  POSTPONE DROP  ( удалить case параметр если = )
  R>             ( мы должны теперь вернуть счетчик назад )
; IMMEDIATE
```



```
: ENDOF ( orig1 #of -- orig2 #of )
  >R    ( переместить со стека в случае, если )
        ( стек потока управления это стек данных. )
  POSTPONE ELSE
  R>    ( мы должны теперь вернуть счетчик назад )
; IMMEDIATE

: ENDCASE  ( orig1..orign #of -- )
  POSTPONE DROP  ( удалить case параметр )
  0 ?DO
    POSTPONE THEN
  LOOP
; IMMEDIATE
```

A.3.2.3.3 Стек возвратов

Ограничения в разделе, 3.2.3.3 Стек возвратов - являются необходимым, если реализации разрешают помещать параметры цикла на стеке возвратов.

A.3.2.6 Запросы к окружению

Размер в адресуемых элементах различных типов данных может быть определен фразами типа 1 CHARS. Точно так же выравнивание может быть определено фразами

типа 1 ALIGNED.

Запросы к окружению разделены на две группы: те, что всегда производят одно и тот же значение и те, которые не могут. Первые группы включают входы типа MAX-N. Эта информация фиксирована аппаратными средствами или конструкцией системы Forth; пользователю гарантируется, что выяснение вопроса однажды является достаточным.

Другая группа запросов - для вещей, которые могут законно изменяться через какое-то время. Например, приложение могло бы проверить присутствие набора слов двойных чисел, используя запрос среды. Если он отсутствует, система могла бы вызывать зависимый от системы процесс для загрузки набора слов. Системе разрешается изменить ENVIRONMENT? базу данных так, чтобы последующие запросы об этом указывали, что он присутствует.

Заметьте что запрос, который возвращает "unknown" ответ, может производить "known" результат на последующий запрос.

A.3.3 _____
Словарь Forth

Стандартная Программа может переопределять стандартное слово на слово с нестандартным определением. Программа все еще Стандартная (так как она может быть построена на любой Стандартной системе), но результат делает объединенный объект (Стандартная система плюс Стандартная программа) нестандартной системой.

A.3.3.1 _____
Область имени

A.3.3.1.2 _____
Имена определений

Язык в этом разрезе должен гарантировать переносимость Стандартных Программ. Если программа использует кое-что вне Стандарта, что она не предоставляет самостоятельно, нет гарантии, что другая реализация будет иметь то, что необходимо программе для работы. Нет намерения, вообще предполагать, что все программы Forth будут так или иначе недостаточными или худшими, потому что они - не стандартные; некоторые из самых прекрасных драгоценных камней программистского искусства будут нестандартными. В то же самое время, комитет пытается гарантировать что программа, помеченная как "стандартная" будет соответствовать некоторым ожиданиям, особенно в отношении переносимости.

Во многих системных средах входной источник не способен поставлять некоторые неграфические символы из-за внешних факторов, типа использования этих символов для управления потоком данных или редактирования. Кроме того, при интерпретации из текстового файла, функция синтаксического анализа специфически обрабатывает неграфические символы подобно пробелам; таким образом, слова полученные текстовым интерпретатором не будут содержать внедренные неграфические символы. Разрешение реализациям в таких средах называть себя Стандартными, является необходимым незначительным ограничением на Стандартные программы.

Стандартной системе позволяется разрешать создание имен определений, содержащих неграфические символы. Исторически, такие имена использовались для функций редактирования с клавиатуры, и "невидимых" слов.

A.3.3.2 _____
Область кода

A.3.3.3 _____
Область данных

Слова #TIB, >IN, BASE, BLK, SCR, SOURCE, SOURCE-ID, STATE, и TIB содержат информацию, используемую системой Forth в ее операциях, и могут быть использованы приложениями. Любое предположение сделанное приложением о данных доступных в системе Forth, которые хранятся иначе, чем данные только что перечисленные - зависимость от окружения.

Нет никакого смысла в определении (в Стандарте) что является, и что не является адресуемым.

Стандартная Программа НЕ может адресовать:
- Непосредственно в стеки данных или возвратов;
- В поле данных определения, если не сохраненное приложением.

Ограничения только для чтения возникают потому, что некоторые системы Forth работают из ROM и некоторого разделения буферов ввода-вывода с другими пользователями или системами. Переносимые программы не могут знать, какие области затронуты, отсюда общие ограничения.

A.3.3.3.1 _____
Выравнивание адреса

Много процессоров имеют ограничения на адреса, которые могут использоваться командами доступа к памяти. Например, на Motorola 68000, 16-бит или 32-бит данные могут быть доступны только в четных адресах. Другие примеры включают архитектуру RISC, где 16-бит данные могут быть загружены или сохранены только в четных адресах и 32-бит данные только в адресах, которые кратны четырем.

Разработчик ANS Forth может справляться с этими ограничениями выравнивания одним из двух способов. Слова доступа к памяти Forth (@, !, +!, и т.д.) мог быть реализованы в терминах инструкций доступа меньшей-ширины, которые не имеют ограничений выравнивания. Например, на 68000 Forth с 16-бит ячейками, @ могло быть реализовано двумя 68000 командами выборки байта и повторной сборкой байтов в 16-бит ячейке. Хотя это скрывает аппаратные ограничения от программиста, это неэффективно, и может иметь непреднамеренные побочные эффекты в некоторых аппаратных средах. Альтернативная реализация ANS Forth могла определять каждое слово доступа к памяти, с использованием родных команд, которые наиболее близко соответствуют функциям слов. В 68000 Forth с 16-бит ячейками, @ использовало бы 68000 16-бит команду перемещения. В этом случае, ответственность за передачу к @ правильно выровненного адреса ложится на программиста. Переносимая ANS Forth программа должна предполагать, что выравнивание может требоваться и следовать требованиям этого раздела.

A.3.3.3.2 _____
Непрерывные области

Область данных системы Forth входит в прерывистые области! Местоположение некоторых областей предоставлено системой, некоторых - программой. Область данных непрерывна в пределах областей, позволяя адресной арифметике генерировать допустимые адреса только в пределах отдельной области. Стандартная Программа не может делать никакие предположения об относительном размещении множественных областей в памяти.

Раздел 3.3.3.2 предписывает условия, при которых могут быть получены непрерывные зоны области данных. Например:

```
CREATE TABLE   1 C, 2 C, ALIGN 1000 , 2000 ,
```

делает таблицу, чей адрес возвращается TABLE. При доступе к этой таблице,

```
TABLE C@                          возвратит 1
TABLE CHAR+ C@                    возвратит 2
```

```
TABLE 2 CHARS + ALIGNED @           возвратит 1000
TABLE 2 CHARS + ALIGNED CELL+ @     возвратит 2000.
```

Точно так же,

```
CREATE DATA   1000 ALLOT
```

делает массив в размере 1000 адресуемых элементов. Более переносимая стратегия определила бы массив в прикладных элементах, типа:

```
500 CONSTANT NCELLS
CREATE CELL-DATA  NCELLS CELLS ALLOT
```

Этот массив может быть индексирован подобно этому:

```
: LOOK   NCELLS 0 DO  CELL-DATA I CELLS + ? LOOP ;
```

A.3.3.3.6 _____
Другие временные области

Во многих существующих системах Forth, эти области - в HERE или сразу вне ее, отсюда многие ограничения.

(2*n)+2 - размер символьной строки, содержащей без знаков препинания двоичное представление максимального двойного числа с ведущим знаком "минус" и завершающим пробелом.

Примечание реализации: Так как минимальное значение n - это 16, абсолютный минимальный размер выходной строки отображаемого числа - 34 символа. Но если ваша реализация имеет больший n, вы должны также увеличить размер выходной строки отображаемого числа.

A.3.4 _____
Интерпретатор текста Forth

A.3.4.3 _____
Семантика

"Семантика инициирования" соответствуют коду, который выполняется после входа в определение, аналогично коду, выполняемому EXIT после выхода из определения. "Семантика времени-выполнения" соответствует кодовым фрагментам, типа литералов или ветвлений, которые откомпилированы внутри определения через двоеточие словами с явной семантикой компиляции.

В кросс-компиляторе Forth может быть определена семантика выполнения, чтобы иметь место только в главной системе, только в целевой системе, или в обеих системах. Например, это может быть свойственно словам типа CELLS выполняющимся на главной системе и возвращающим значение описывающее целевую, для определений через двоеточие выполняющимся только на целевой, и для CONSTANT и VARIABLE имеющих поведение при выполнении на обеих системах. Подробности поведения кросс-компилятора вне контекста этого Стандарта.

A.3.4.3.2 _____
Семантика интерпретации

Ввиду разнообразных причин, этот Стандарт не определяет семантику интерпретации для каждого слова. Примеры этих слов - >R, ." , DO, и IF. Ничто в этом Стандарте не препятствует реализациям обеспечивать семантику интерпретации для этих слов, типа интерактивные слова потока управления. Однако, Стандартные Программы не могут использовать их в состоянии интерпретации.

A.3.4.5 _____
Компиляция

Рекурсия компилятора на уровне определения потребляет чрезмерные ресурсы, особенно поддержка locals. Технический Комитет не полагает, что выгоды покрывают затраты. Вложение определений - также не общая практика и не будет работать на многих системах.

A.4 _____
Документационные требования

A.4.1 _____
Системная документация

A.4.2 _____
Программная документация

A.5 _____
Соглашение и наименование

A.5.1 _____
ANS Forth системы

Раздел 5.1 определяет критерии, которым система должна удовлетворять, чтобы оправдывать обозначение "ANS Forth Система". Кратко, минимальное требование - это система должна "реализовать" Основной набор слов. Есть несколько путей, которыми это требование может быть выполнено. Наиболее очевидное - что все Основные слова могут быть в пред-откомпилированном ядре. Однако это не единственный путь удовлетворения требования. Например, некоторые слова могут быть предоставлены в исходных блоках или файлах с командами, объясняющими как добавлять их в систему, если они необходимы. Пока слова предоставлены таким способом, что пользователь может получить доступ к ним с ясной и прямой процедурой, они могут рассматриваться как присутствующие.

Кросс-компилятор Forth имеет много характерного общего с ANS Forth Системой, в том числе оба используют подобные средства компиляции для обработки программы. Однако чтобы полностью определить ANS Forth кросс-компилятор, было бы необходимо обращаться к сложным проблемам, имеющим дело с компиляцией и семантикой выполнения в ведущих и целевых средах также как проблемам ROM-базирования. Степень усилий для выполнения этого должным образом, оказывается не практичной в это время. В результате, хотя может быть возможно для Forth кросс-компилятора правильно готовить ANS Forth программу к выполнению в целевой среде, неуместно для кросс-компилятора быть обозначенным ANS Forth Системой.

A.5.2 _____
ANS Forth программы

A.5.2.2 _____
Программное наименование

Объявление зависимости от окружения не должно рассматриваться нежелательным, просто подтверждение, что автор воспользовался преимуществом некоторой принятой архитектуры. Например, большинство компьютеров в обычном использовании основано на двоичной арифметике дополнительного кода (дополнение до двух). Подтверждая зависимость от окружения на этой архитектуре, программист становится имеющим право использовать число -1, чтобы представить весь набор битов без значительного ограничения переносимости программы.

Поскольку все программы требуют пространства для данных и команд, и времени, чтобы выполнить эти команды, они зависят от присутствия среды, предоставляющей эти ресурсы. Невозможно предсказать как мало некоторых из этих ресурсов

(например, стековое пространство) может быть необходимо для исполнения некоторой задачи, поэтому этот Стандарт так не делает.

С другой стороны, поскольку программа требует увеличивающихся уровней ресурсов, будет вероятно постепенно меньшее количество систем, на которых она выполнится успешно. Алгоритм требующий массива 109 ячеек мог бы работать на меньшем количестве компьютеров чем требующий только 103.

Так как нет также никакого способа знать, какой минимальный уровень ресурсов будет реализован в системе полезной, по крайней мере, для некоторых задач, любая программа, выполняющая реальную работу, и помеченная просто "ANS Forth Программа" вряд ли помечена правильно.

A.6 Словарь

В этом и последующих разделах мы представляем объяснения для обработки специфических слов: почему мы включили их, почему мы поместили их в некоторые наборы слов, или почему мы так определили их имена или значение.

Слова в этом разделе организованы в наборы слов, с сохранением их индексных номеров для простой системы перекрестных ссылок к словарю.

Исторически, много систем Forth были написаны в Forth. Многие из слов в Forth первоначально имели как первичную цель непосредственно поддержку системы Forth. Например, WORD и FIND часто используются как основные инструменты интерпретатора текста Forth, и CREATE во многих системах - примитив для формирования входов словаря. В определении слов типа этих стандартным способом, мы пытались не препятствовать их использованию разработчиками. Одна из возможностей Forth, который вызвал любовь к нему его пользователей - это то, что одни и те же инструментальные средства, использующиеся для реализации системы, являются доступными прикладному программисту - результат этого подхода это компактность и эффективность, которая характеризует большинство реализаций Forth.

A.6.1 Основные слова

A.6.1.0070 '
Типичное использование:

```
... ' name .
```

Много систем Forth используют это изящное слово. Многие нет. ANS Forth наследует использование Forth 83.

См.: 3.4.3..2 Семантика интерпретации, A.6.1.1550 FIND.

A.6.1.0080 (
Типичное использование:

```
... ( ccc) ...
```

A.6.1.0140 +LOOP
Типичное использование:

```
: X ... limit first DO ... step +LOOP ;
```

A.6.1.0150 ,
Использование , (запятая) для компиляции идентификаторов исполнения - не переносимое.

См.: 6.2.0945 COMPILE,.

A.6.1.0190 ."
Типичное использование:
: X ... ." ccc" ... ;

Реализация может определять семантику интерпретации для ." если желательно. В одной вероятной реализации, интерпретация ." отобразила бы разграниченное сообщение. В другой вероятной реализации, интерпретация ." компилировала бы код для отображения сообщения позже. В еще другой вероятной реализации, интерпретация ." была бы обработана как исключение. Обусловленная этими вариантами Стандартная Программа не может использовать ." при интерпретации. Точно так же Стандартная Программа не может компилировать POSTPONE ." внутри нового слова, и затем использовать это слово при интерпретации.

A.6.1.0320 2*
Исторически 2* было реализовано на машинах с дополнительным кодом (дополнение до двух) как логическая команда левого сдвига. Умножение на два - эффективный побочный эффект на этих машинах. Однако смещение подразумевает знание значения и позиции битов в ячейке. В то время как имя подразумевает умножение, большинство разработчиков использует аппаратный левый сдвиг для реализации 2*.

A.6.1.0330 2/
Это слово имеет то же самое общее использование и следствие неверного названия как 2*. Оно часто реализовано на машинах с дополнительным кодом (дополнение до двух) с аппаратным правым сдвигом, который распространяет знаковый разряд.

A.6.1.0350 2@
Для 2@ порядок сохранения определен Стандартом.

A.6.1.0450 :
Типичное использование:
: name ... ;

В Forth 83, это слово было определено для изменения порядка поиска. Эта спецификация явно удалена в этом Стандарте. Мы полагаем, что в большинстве случаев это не имеет никакого эффекта; однако системы, которые разрешают много порядков поиска, нашли Forth-83 поведение двоеточия очень нежелательным.

Заметьте, что двоеточие самостоятельно не вызывает компилятор. Двоеточие устанавливает состояние компиляции так, чтобы более поздние слова в области анализа были откомпилированы.

A.6.1.0460 ;
Типичное использование:
: name ... ;

Одна функция выполняемая ; и ;CODE это позволить текущему определению быть найденным в словаре. Если текущее определение было создано с помощью :NONAME текущее определение не имеет имя определения и таким образом не может быть найдено в словаре. Если :NONAME реализовано, компилятор Forth должен сохранить достаточно информации о текущем определении, чтобы позволить ; и ;CODE определить, действительно ли должно быть предпринято какое либо действие, чтобы позволить ему быть найденным.

A.6.1.0550 >BODY
a-addr - адрес, который HERE возвратило бы, если бы было выполнено немедленно после выполнения CREATE которое определило xt.

A.6.1.0680 ABORT"
Типичное использование:
: X ... test ABORT" ccc" ... ;

A.6.1.0695 ACCEPT
Предыдущие стандарты определяли, что накопление входной строки заканчивается, когда получен "возврат" или когда было получено +n1 символов. Завершение, когда было получено +n1 символов, трудно, дорого, или невозможно осуществить в некоторых системных средах. Следовательно, множество существующих реализаций не подчиняется с этим требованиям.

Так как редактирование строки и функции накопления часто реализуются системными компонентами вне контроля реализацией Forth, этот Стандарт не налагает никакое такое требование. Стандартная Программа может только предполагать, что она может получать ввод строки от ACCEPT или EXPECT. Детальная последовательность пользовательских действий, необходимых для подготовки и передачи этой строки, вне контекста этого Стандарта.

Спецификация ненулевого, положительного целочисленного счетчика (+n1) для ACCEPT позволяет некоторым разработчикам продолжать их практику использования нуля или отрицательной величины как флага для вызова специального поведения. Поскольку такое поведение вне Стандарта, Стандартные программы не могут зависеть от этого, но Технический Комитет не желает излишне препятствовать этому. Так как фактические значения - почти всегда маленькие целые числа, никакие функциональные возможности не повреждаются этим ограничением.

ACCEPT и EXPECT исполняют схожие функции. ACCEPT рекомендуется для новых программ, и будущее использование EXPECT, не одобряется.

Рекомендуется, чтобы все неграфические символы были зарезервированы для функций редактирования или управления и не были сохранены во входной строке.

Обычно, когда пользователь готовит входную строку, которая будет передана программе, система позволяет пользователю редактировать эту строку и исправлять ошибки перед передачей заключительной версии строки. Функция редактирования предоставлена иногда непосредственно Forth системой, и иногда внешним системным программным обеспечением или аппаратными средствами. Таким образом, управляющие символы и функции не могут быть доступны на всех системах. В обычном случае конец процесса редактирования и заключительной передачи строки обозначается пользователем нажатием клавиши "Return" или "Enter".

Как в предыдущих стандартах, EXPECT возвращает входную строку немедленно после ввода требуемого числа символов, а также когда получен признак конца строки. Широко рассматриваемое поведение "автоматического завершения, после того как было введено определенное количество символов" нежелательно, потому что пользователь "потеряет контроль" процесса входного редактирования в потенциально неизвестное время (пользователь не обязательно знает, сколько символов требовалось от, EXPECT). Таким образом, EXPECT и SPAN были сделаны устаревшими и существуют в Стандарте только как уступка существующим реализациям. Если EXPECT существует в Стандартной Системе, оно должно иметь поведение "автоматического завершения".

ACCEPT не имеет поведения "автоматического завершения" EXPECT. Однако, потому что внешние системные аппаратные средства и программное обеспечение могут исполнять функцию ACCEPT, когда получен признак конца строки действие курсора, и, следовательно, отображение - определенное реализацией. Рекомендуется, чтобы курсор остался немедленно после введенного текста после получения признака конца строки.

A.6.1.0705 ALIGN
В этом Стандарте мы попытались предусмотреть переносимость между различными CPU архитектурами. Одна из частых причин проблем переносимости - требование выровненных на ячейку адресов на некоторых CPU. На этих системах ALIGN и ALIGNED могут требоваться, чтобы формировать и прослеживать структуры данных, сформированные с помощью C,. Разработчики могут определять эти слова как пустые

команды на системах, для которых они - не функциональны.

A.6.1.0706 ALIGNED
См.: A.6.1.0705 ALIGN.

A.6.1.0760 BEGIN
Типичное использование:
```
        : X ... BEGIN ... test UNTIL ;
```
или
```
        : X ... BEGIN ... test WHILE ... REPEAT ;
```

A.6.1.0770 BL
Поскольку пробел используется повсюду в Forth как стандартный разделитель, это слово единственный путь программе найти и использовать системное значение "пробела". Значение символа пробела не может быть получено с помощью CHAR, например.

A.6.1.0880 CELL+
Как ALIGN и ALIGNED, слова CELL и CELL+ были добавлены, чтобы помочь в переносимости между системами с различными размерами ячейки. Они предназначены для использования в управлении индексами и адресами в целых числах размеров ячеек.
Пример:
```
        2VARIABLE DATA
        0 100 DATA 2!
        DATA @ . 100
        DATA CELL+ @ .  0
```

A.6.1.0890 CELLS
См.: A.6.1.0880 CELL+.
Пример:
```
        CREATE NUMBERS  100 CELLS ALLOT
( Назначает пространство в массиве NUMBERS для 100 ячеек данных.)
```

A.6.1.0895 CHAR
Типичное использование:
```
        ... CHAR A CONSTANT "A" ...
```

A.6.1.0950 CONSTANT
Типичное использование:
```
        ... DECIMAL 10 CONSTANT TEN ...
```

A.6.1.1000 CREATE
Адрес области данных слова, определенного CREATE, дается указателем области данных немедленно после выполнения CREATE.
Резервирование пространства поля данных типично делается с помощью ALLOT.

Типичное использование:
```
        ... CREATE SOMETHING ...
```


A.6.1.1240 DO
Типичное использование:
```
        : X ... limit first DO ... LOOP ;
```
или
```
        : X ... limit first DO ... step +LOOP ;
```

A.6.1.1250 DOES>
Типичное использование:

```
: X ... DOES> ... ;
```

Относительно DOES>, Стандартная Программа не может делать какие-либо предположения относительно способности находить имя определения содержащего DOES> или любое предыдущее определение чье имя может быть скрыто этим. DOES> эффективно заканчивает одно определение и начинает другое насколько заинтересованы переменные local и структуры потока управления. Поведение компиляции проясняет, что пользователь не имеет право помещать DOES> внутри любых структур потока управления.

A.6.1.1310 ELSE
Типичное использование:
```
: X ... test IF ... ELSE ... THEN ;
```

A.6.1.1345 ENVIRONMENT?
В Стандартной Системе, которая содержит только Основной набор слов, эффективное использование ENVIRONMENT? требует или его использования в пределах определения, или использования обеспеченного пользователем вспомогательного определения. Основной набор слов испытывает недостаток прямого метода сборки строки в состоянии интерпретации (11.6.1.2165 S" находится в дополнительном наборе слов), а также средства для проверки возвращенного флага в состоянии интерпретации (например, дополнительный 15.6.2.2532 [IF]).

Комбинация 6.1.1345 ENVIRONMENT?, 11.6.1.2165 S", 15.6.2.2532 [IF], 15.6.2.2531 [ELSE], and 15.6.2.2533 [THEN] составляет эффективный набор слов для условной компиляции, которая работает в состоянии интерпретации.

A.6.1.1360 EVALUATE
Технический Комитет знает, что эта функция обычно записывается как EVAL. Однако существуют реализации, которые могут страдать от определения слова сделанного так как здесь. Мы также находим EVALUATE более читабельным и ясным. Было некоторое мнение для вызова INTERPRET, но это также имело бы нежелательные действия на существующий код. Длинное написание не считали существенным, так как это - не то слово, которое должно использоваться часто в исходном тексте.

A.6.1.1380 EXIT
Типичное использование:
```
: X ... test IF ... EXIT THEN ... ;
```


A.6.1.1550 FIND
Одна из более трудных проблем, которые Комитет решал, была проблема отделения спецификации механизмов реализации от спецификации Forth языка. Могут быть быстро перечислены три основных подхода реализации:

1)
Механизмы шитого кода. Это традиционные подходы к реализации Forth, но могут использоваться другие методы.

2)
Подпрограммный шитый с "макрорасширением" (копирование кода). Короткие подпрограммы, подобно коду для DUP, скорее скопированы в определение, чем скомпилирована JSR ссылка.

3)
Непосредственное кодирование с оптимизацией. Это может включать оптимизацию стека (заменяющую такие фразы как SWAP ROT + одной или двумя машинными командами, например), распараллеливание (тенденция в более новых чипах RISC - это иметь несколько функциональных подэлементов которые могут выполняться параллельно), и так далее.

Первоначальное требование (унаследованное от Forth 83) это адреса компиляции

откомпилированы в словарь, исключая типы реализации 2 и 3.

Типа 3 механизмы и оптимизации типа 2 реализаций, были затруднены явной спецификацией непосредственного исполнения или не непосредственного исполнения всех стандартных слов. POSTPONE позволяет де-спецификацию непосредственного исполнения или не непосредственного исполнения для всех кроме нескольких слов Forth, чье поведение должно быть STATE -независимо.

Один тип 3 реализации, cmForth Чарльза Мура, имеет версии компиляции и интерпретации многих слов Forth. В настоящем, это, кажется общим подходом для типа 3 реализации. Комитет чувствовал, что этот подход реализации должен быть разрешенным. Следовательно, возможно, что слова без семантики интерпретации могут быть найдены только в течение компиляции, и другие слова могут существовать в двух версиях: версия компиляции и версия интерпретации. Следовательно, значения возвращенные FIND, могут зависеть от STATE, и ' и ['] могут быть не способны найти слова без семантики интерпретации.

A.6.1.1561 FM/MOD
Представляя требование для "минимального" деления, Forth 83 произвел много споров и беспокойств со стороны тех, кто предпочли более обычную практику, поддерживаемую в других языках, реализации деления согласно поведению ведущего CPU, которое наиболее часто симметричное (округленное к нулю). В попытке найти компромиссную позицию, этот Стандарт предусматривает примитивы для обеих общих разновидностей, минимального и симметричного (см. SM/REM). FM/MOD - минимальная версия.

Технический Комитет рассмотрел обеспечение двух законченных наборов явно именованных операторов деления, и отказался делать так на том основании, что это незаконно увеличило бы и усложнило Стандарт. Вместо этого, разработчики могут определять нормальные слова деления в терминах или FM/MOD или SM/REM, предоставляя обоснование их выбора. Пожелание людей иметь явно именованные наборы операторов поощряются. FM/MOD может использоваться, например, для определения:


```
: /_MOD ( n1 n2 -- n3 n4)  >R S>D R> FM/MOD ;
: /_  ( n1 n2 -- n3)  /_MOD SWAP DROP ;
: _MOD ( n1 n2 -- n3)   /_MOD DROP ;
: */_MOD ( n1 n2 n3 -- n4 n5)  >R M* R> FM/MOD ;
: */_  ( n1 n2 n3 -- n4 )   */_MOD SWAP DROP ;
```

A.6.1.1700 IF
Типичное использование:

```
: X ... test IF ... THEN ... ;
```

или

```
: X ... test IF ... ELSE ... THEN ... ;
```

A.6.1.1710 IMMEDIATE
Типичное использование:

```
: X  ...  ;  IMMEDIATE
```

A.6.1.1720 INVERT
Слово NOT было первоначально предусмотрено в Forth как оператор флага, чтобы делать структуры управления читаемыми. При его назначенном использовании следующие два определения произвели бы идентичный результат:

```
: ONE  ( flag -- )
   IF ." true" ELSE ." false" THEN ;

: TWO ( flag -- )
   NOT IF ." false" ELSE ." true" THEN ;
```

Это было обычное использование до Forth-83 Стандарта, который переопределил NOT как в ячейку шириной операцию обратного кода (дополнение до единицы), функционально эквивалентную фразе -1 XOR. В то же самое время, тип данных, управляемый этим словом был изменен от флага до набора битов в ячейку шириной, и стандартное значения для true было изменено с "1" (набор только бита младшего разряда) на "-1" (весь набор битов). Поскольку эти определения TRUE и NOT были несовместимы с их предыдущими определениями, много пользователей Forth продолжают полагаться на старые определения. Следовательно, обе версии находятся в общем использовании.

Поэтому использование NOT не может быть стандартизировано в это время. Два традиционных значения NOT - это отрицание смысла флага и выполнение операции обратного кода (дополнение до единицы) - сделаны доступными благодаря 0= и INVERT, соответственно.

A.6.1.1730 J
J может только использоваться со вложенными, DO...LOOP, DO...+LOOP, ?DO...LOOP, or ?DO...+LOOP, например, в форме:

```
: X ... DO ... DO ... J ... LOOP ... +LOOP ... ;
```


A.6.1.1760 LEAVE
Заметьте, что LEAVE немедленно завершает цикл. Никакие слова после LEAVE в пределах цикла не будут выполнены.
Типичное использование:

```
: X ... DO ... IF ... LEAVE THEN ... LOOP ... ;
```

A.6.1.1780 LITERAL
Типичное использование:

```
: X  ... [ x ] LITERAL ...  ;
```

A.6.1.1800 LOOP
Типичное использование:

```
: X ... limit first DO ... LOOP ... ;
```

или

```
: X ... limit first ?DO ... LOOP ... ;
```

A.6.1.1810 M*
Это слово - полезный ранний шаг в вычислениях, давая реальную дополнительную точность. Это было в использовании, начиная с Forth систем начала 1970-ых.

A.6.1.1900 MOVE
CMOVE и CMOVE> - первичные операторы перемещения в Forth 83. Они определяют поведение для перемещения, которое предполагает размножение, если перемещение вызвано соответствующим образом. В некоторых аппаратных средствах, это специфическое поведение не может быть достигнуто, используя самую лучшую команду перемещения. Кроме того, CMOVE и CMOVE> перемещают символы; ANS Forth нуждается в команде перемещения, способной иметь дело с адресуемыми элементами. Таким образом, MOVE было определено и добавлено к Основному набору слов, и CMOVE и CMOVE> были перемещены в Строковый набор слов.

A.6.1.2033 POSTPONE
Типичное использование:

```
: ENDIF  POSTPONE THEN ;  IMMEDIATE
: X  ... IF ... ENDIF ... ;
```

POSTPONE заменяет большинство функциональных возможностей COMPILE и [COMPILE]. COMPILE и [COMPILE] используются для той же самой цели: отложить поведение компиляции следующего слова в области анализа. COMPILE было предназначено для применения к словам не немедленного исполнения и [COMPILE] к словам немедленного исполнения. Это отягощает программиста необходимостью знать, какие

слова в системе немедленного исполнения. Следовательно, стандарты Forth должны были определить немедленное исполнение или не немедленное исполнение всех слов, охваченных Стандартом. Это излишне сдерживает разработчиков.

Вторая проблема с COMPILE - это что некоторые программисты договорились предполагать и использовать специфическую реализацию, а именно:

```
:  COMPILE  R>  DUP  @  ,  CELL+  >R  ;
```


Эта реализация не будет работать на системе Forth с непосредственным кодом. В непосредственном коде Forth использующем inline расширение кода и локальную оптимизацию, размер произведенного объектного кода изменяется; эту информацию трудно сообщить "тупому" COMPILE. "Разумный" (то есть, немедленного исполнения) COMPILE, не имел бы этой проблемы, но это запрещали предыдущие стандарты.

По этим причинам, COMPILE не был включен в Стандарт, и [COMPILE] был перемещен в пользу POSTPONE. Дополнительное обсуждение может быть найдено в Hayes, J.R., "Postpone", Доклады 1989 Rochester Forth Conference.

A.6.1.2120 RECURSE
Типичное использование:

```
: X ... RECURSE ... ;
```

Это - оператор рекурсии Forth; на некоторых реализациях оно вызывает САМО СЕБЯ. Обычный пример - кодирование функции факториала.

```
: FACTORIAL ( +n1 -- +n2)
   DUP 2 < IF  DROP 1 EXIT  THEN
   DUP 1-  RECURSE  *
;
```

n2 = n1(n1-1)(n1-2)...(2)(1), произведение n1 со всеми положительными целыми числами меньше чем оно само (как специальный случай, нулевой факториал равняется одному). Хотя возлюбленная компьютерными учеными, рекурсия делает необычно тяжелым использование обоих стеков и должна, поэтому быть использована с осторожностью. См. альтернативное определение A.6.1.2140 REPEAT.

A.6.1.2140 REPEAT
Типичное использование:

```
: FACTORIAL ( +n1 -- +n2)
   DUP 2 < IF  DROP 1 EXIT  THEN
   DUP
   BEGIN  DUP 2 > WHILE
     1-  SWAP OVER *  SWAP
   REPEAT  DROP
;
```

A.6.1.2165 S"
Типичное использование:

```
: X  ... S" ccc" ... ;
```

Это слово найдено во многих системах под именем " (кавычка). Однако текущая практика состоит в почти равномерном разделении в использовании ": многие системы, используют семантику выполнения учитываемую здесь, в то время как другие возвращают адрес строки со счетчиком. Мы делаем здесь попытку удовлетворить оба лагеря, предусматривая два слова, S" и Расширение Основного слова C" так, чтобы пользователи могли иметь любое поведение ими ожидаемое с простой операцией переименования.

A.6.1.2214 SM/REM
См. предыдущее обсуждение деления под FM/MOD. SM/REM - примитив симметричного деления, который позволяет программам определять следующие операторы симметричного деления:

```
: /-REM  ( n1 n2 -- n3 n4 )  >R  S>D  R> SM/REM ;
: /-  (  n1 n2 -- n3 )  /-REM SWAP DROP ;
: -REM  ( n1 n2 -- n3 )  /-REM DROP ;
: */-REM  (  n1 n2 n3 -- n4 n5 )  >R  M*  R> SM/REM ;
: */-  ( n1 n2 n3 -- n4 )  */-REM SWAP DROP ;
```

A.6.1.2216 SOURCE
SOURCE упрощает процесс непосредственного доступа к входному буферу, скрывая различия между его местоположением для различных входных источников. Это также дает разработчикам больше гибкости в их реализациях механизмов буферизации для различных входных источников. Комитет удалился от входной буферной спецификации состоящей из коллекции индивидуальных переменных, объявляя TIB и #TIB устаревшими.

SOURCE в этой форме существует в F83, POLYFORTH, LMI's Forths и других. В обычных системах это эквивалентно фразе

```
BLK @  IF BLK @ BLOCK 1024  ELSE TIB #TIB @ THEN
```

A.6.1.2250 STATE
Хотя EVALUATE, LOAD, INCLUDE-FILE, и INCLUDED не перечислены как слова, которые изменяют STATE, текст, интерпретируемый любым из этих слов, может включать одно или более слов, которые явно изменяют STATE. EVALUATE, LOAD, INCLUDE-FILE, и INCLUDED сами не делают изменения в STATE.

STATE не вложено при вложении текстового интерпретатора. Например, последовательность кода:

```
: FOO  S" ]" EVALUATE ;       FOO
```

оставит систему в состоянии компиляции. Точно так же после выполнения LOAD с блоком содержащим], система будет в состоянии компиляции.

Заметьте, что] не затрагивает область анализа и что единственное влияние которое : оказывает на область анализа, это выделить слово. Это дает право программе использовать эти слова для установки состояния с известными побочными эффектами на область анализа. Например:

```
: NOP  : POSTPONE ; IMMEDIATE ;

NOP ALIGN    NOP ALIGNED
```

Некоторые не-ANS Forth совместимые системы имеют] вызывающее цикл компилятора в дополнение к установке STATE. Такая система неуместно пыталась бы компилировать второе использование NOP.

Также заметьте, что ничто в Стандарте не мешает программе искать идентификаторы исполнения] или [и использовать их для изменения STATE. Эти факты предлагают, что реализации] будут делать только установку STATE и единственный цикл интерпретатора/компилятора будет контролировать STATE.

A.6.1.2270 THEN
Типичное использование:

```
: X ... test IF ... THEN ... ;
```

или

```
        : X ... test IF ... ELSE ... THEN ... ;
```

A.6.1.2380 UNLOOP
Типичное использование:
```
        : X  ...
         limit first
         DO
           ... test IF ... UNLOOP EXIT THEN ...
         LOOP ...
        ;
```

UNLOOP позволяет использование EXIT в пределах контекста, DO ... LOOP и связанной do-loop конструкции. UNLOOP как функция был назван UNDO. UNLOOP более выражает действие: ничто не отменяется - мы просто прекращаем делать это.

A.6.1.2390 UNTIL
Типичное использование:
```
        : X ... BEGIN ... test UNTIL ... ;
```

A.6.1.2410 VARIABLE
Типичное использование:
```
        ... VARIABLE XYZ ...
```

A.6.1.2430 WHILE
Типичное использование:
```
        : X ... BEGIN ... test WHILE ... REPEAT ... ;
```

A.6.1.2450 WORD
Типичное использование:
```
        char WORD ccc<char>
```


A.6.1.2500 [
Типичное использование:
```
        : X ... [ 4321 ] LITERAL ... ;
```

A.6.1.2510 [']
Типичное использование:
```
        : X  ... ['] name ... ;
```

См.: A.6.1.1550 FIND.

A.6.1.2520 [CHAR]
Типичное использование:
```
        : X  ...  [CHAR] ccc  ...  ;
```

A.6.1.2540]
Типичное использование:
```
        : X ... [ 1234 ] LITERAL ... ;
```

A.6.2 Расширения основных слов

Слова в этой коллекции относятся к нескольким категориям:
- Слова, которые находятся в общем использовании, но считаются менее обязательными чем Основные слова (например, 0<>);
- Слова, которые находятся в общем использовании, но могут быть тривиально определены из Основных слов (например, FALSE);
- Слова, которые являются прежде всего полезными в узко определенных типах приложений или менее часто используются (например, PARSE);
- Слова, которые осуждаются в пользу новых слов, представленных для решения специфических проблем (например, CONVERT).

Из-за различия оснований для включения этих слов, Технический Комитет не поощряет разработчиков предлагать законченную коллекцию, но выбирать слова считающиеся наиболее ценными для их клиентуры.

A.6.2.0060 #TIB
Функция #TIB была заменена SOURCE.

A.6.2.0200 .(
Типичное использование:

```
        .( ccc)
```

A.6.2.0210 .R
В .R, "R" сокращение для RIGHT.

A.6.2.0340 2>R
Исторически, 2>R использовался для реализации DO. Отсюда порядок параметров на стеке возвратов.

Основная польза 2>R в том, что оно помещает верхний элемент стека на вершину стек возвратов. Например, число две-ячейки может быть перемещено на стек возвратов и все еще иметь самую старшую ячейку доступной на вершине стека возвратов.

A.6.2.0410 2R>
Заметьте, что 2R> - не эквивалентен R> R>. Вместо этого, это отражает действие 2R> (см. A.6.2.0340).

A.6.2.0455 :NONAME
:NONAME позволяет пользователю создавать идентификатор исполнения с семантикой определения через двоеточие без связанного имени. Ранее, только : (двоеточие) могло создавать идентификатор исполнения с этой семантикой. Таким образом, код Forth мог быть откомпилирован только, используя синтаксис :, так:

```
        : NAME  ...  ;
```

:NONAME устраняет это ограничение и помещает компилятор Forth в руки программиста.

:NONAME может использоваться для создания специфических для применения языков программирования. Одна методика - это смешать кодовые фрагменты Forth со специфическими для применения конструкциями. Специфические для применения конструкции используются :NONAME для компиляции Forth кода и сохранения соответствующего идентификатора исполнения в структурах данных.

Функциональные возможности :NONAME могут быть встроены в любую систему Forth. Годами эксперты Forth программирования использовали детальное знание их систем для генерации безымянных фрагментов кода. Теперь эта функция была именована и может использоваться в переносимой программе.

Например :NONAME может использоваться для формирования таблицы кодовых фрагментов, где индексация в таблице позволяет выполнять специфический фрагмент. Синтаксис объявления таблицы:

```
        :NONAME .. code for command 0 .. ;  0 CMD !
        :NONAME .. code for command 1 .. ;  1 CMD !
           ...
        :NONAME .. code for command 99 .. ; 99 CMD !

        ... 5 CMD @ EXECUTE ...
```

Определения таблицы, формирующей слова:

```
CREATE CMD-TABLE  \ таблица для команд идентификаторов исполнения
  100 CELLS ALLOT

: CMD ( n -- a-addr ) \ энный адрес элемента в таблице
  CELLS CMD-TABLE + ;
```

Как дальнейший пример, может быть создано определяющее слово, чтобы позволять контроль выполнения. В примере ниже подсчитывается число раз выполнения слова. : должен быть сначала переименован, чтобы позволить определение нового ; .


```
: DOCOLON ( -- )
\ Изменить созданное CREATE слово, чтобы выполниться подобно
\ определению двоеточия
  DOES> ( i*x a-addr -- j*x )
   1 OVER +!          \ выполнение счетчика
   CELL+ @ EXECUTE    \ выполнение :NONAME определения
;

: OLD: : ;            \ только псевдоним

OLD: : ( "name" -- a-addr xt colon-sys )
\ начинает счетное-выполнение определения через двоеточие
   CREATE  HERE 0 ,   \ память для счетчика выполнения
   0 ,                \ память для идентификатора исполнения
   DOCOLON            \ установить время выполнения для созданного
                      \ CREATE слова
   :NONAME            \ начать неименованное определение через двоеточие
;

( Примечание размещения DOES>: DOES> должно изменить созданное CREATE
  слово а не :NONAME определение, так что DOES> должно выполниться
  прежде :NONAME.)

OLD: ; ( a-addr xt colon-sys -- )
\ заканчивает счетное-выполнение определения через двоеточие
   POSTPONE ;         \ завершает компиляцию определения через двоеточие
   SWAP CELL+ !       \ сохраняет идентификатор исполнения
;  IMMEDIATE
```

Новые : и ; используются точно так же как стандартные для определения слов:

```
... : xxx  ... ;  ...  xxx  ...
```

Теперь однако, эти слова можно "ticked", чтобы отыскать счетчик (и идентификатор исполнения):

```
... ' xxx >BODY ? ...
```

A.6.2.0620 ?DO
Типичное использование:

```
: FACTORIAL ( +n1 -- +n2 )  1 SWAP 1+ ?DO  I *  LOOP ;
```

Это слово было добавлено в ответ на многие запросы для разрешения трудности внесенной Forth-83 словом DO, которое на 16-бит системе будет выполняться 65,535 раз, если даны равные параметры. Так как этот Стандарт также поощряет 32-разрядные системы, то это поведение может быть невыносимым. Технический Комитет рассмотрел применение этой семантики к DO, но отклонил на том основании, что это могло бы нарушить существующий код.

A.6.2.0700 AGAIN
Типичное использование:
```
        : X ... BEGIN ... AGAIN ... ;
```

Если последовательность слов не имеет способ закончиться, это бесконечный цикл.

A.6.2.0855 C"
Типичное использование:
```
        : X  ...  C" ccc"  ...  ;
```

Просто преобразовать строки со счетчиком к указателю/длине, но трудно сделать наоборот. C" является единственным новым словом, которое использует представление стека "адрес строки со счетчиком". Оно предусмотрено как помощь перенесению существующих программ на ANS Forth системы. Относительно трудно реализовать C" в терминах других стандартных слов, рассматривая их семантику "компиляция строки в текущее определение".

Пользователи C" поощряются к перемещению их прикладного кода к непротиворечивому использованию преимущественно "c-addr u" стекового представления с помощью дополнительного слова S". Это может быть выполнено преобразованием прикладных слов с входными параметрами со строкой со счетчиком для использования преимущественно "c-addr u" представления, таким образом устраняя потребность в C".

См.: 3.1.3.4 Строки со счетчиком.

A.6.2.0873 CASE
Типичное использование:
```
        : X ...
           CASE
             test1 OF ... ENDOF
             testn OF ... ENDOF
             ... ( default )
           ENDCASE ...
        ;
```

A.6.2.0945 COMPILE,
COMPILE, - эквивалент EXECUTE для компиляции. Во многих случаях возможно компилировать слово с использованием POSTPONE без использования COMPILE,. Однако использование POSTPONE требует, чтобы имя слова было известно во время компиляции, тогда как COMPILE, позволяет слову быть размещенным в любое время. В некоторых случаях возможно использовать EVALUATE для компиляции слова, чье имя не известно до времени выполнения. Это имеет две возможных проблемы:

- EVALUATE медленнее чем COMPILE, , потому что требуется поиск в словаре.
- Текущий порядок поиска затрагивает результат, EVALUATE.

В традиционных реализациях шитого кода, компиляция выполняется с помощью , (запятая). Это использование - не переносимое; это не работает для подпрограммного шитого кода, непосредственного кодирования, или перемещаемых реализаций. Использование COMPILE, является переносимым.

В большинстве систем возможно реализовать COMPILE, так, что оно сгенерирует код оптимизированный до той же самой степени как код сгенерированный нормальным процессом компиляции. Однако в некоторых реализациях есть два различных "идентификатора", соответствующих специфическому имени определения: нормальный "идентификатор исполнения", который используется при интерпретации или с EXECUTE, , и "идентификатор компиляции", который используется при компиляции. И не всегда возможно получить идентификатор компиляции от идентификатора исполнения. В этих реализациях COMPILE, не могло бы генерировать код являющийся

столь же эффективным как обычно оттранслированная программа.

A.6.2.0970 CONVERT
CONVERT может быть определено следующим образом:

```
: CONVERT   CHAR+ 65535 >NUMBER DROP ;
```

A.6.2.1342 ENDCASE
Типичное использование:

```
: X ...
   CASE
     test1 OF ... ENDOF
     testn OF ... ENDOF
     ... ( default )
   ENDCASE ...
;
```

A.6.2.1343 ENDOF
Типичное использование:

```
: X ...
   CASE
     test1 OF ... ENDOF
     testn OF ... ENDOF
     ... ( default )
   ENDCASE ...
;
```

A.6.2.1390 EXPECT
Спецификация положительного целочисленного счетчика (+n) для EXPECT, позволяет некоторым разработчикам продолжать их практику использования нулевого или отрицательного значения как флага для вызова специального поведения. Поскольку такое поведение - вне Стандартного, Стандартные Программы не могут зависеть от этого, но Технический Комитет не желает этому излишне препятствовать. Так как фактические значения - почти всегда маленькие целые числа, никакие функциональные возможности не ослабляются этим ограничением.

A.6.2.1850 MARKER
Поскольку реализации словаря стали более сложными, и в некоторых случаях используют множественные адресные пространства, FORGET стал предельно трудным или невозможным для реализации на многих системах Forth. MARKER очень ослабляет проблему, делая возможным для системы помнить "ориентирующую информацию" заранее, которая конкретно отмечает места, где словарь может быть перестроен в некотором будущем.

A.6.2.1950 OF
Типичное использование:

```
: X ...
   CASE
     test1 OF ... ENDOF
     testn OF ... ENDOF
     ... ( default )
   ENDCASE ...
;
```

A.6.2.2000 PAD
PAD была доступна как рабочая память для строк начиная с самых ранних Forth реализаций. Было доведено до нашего внимания, что много программистов отказываются от использования PAD, опасаясь несовместимости в системном использовании. PAD конкретно предназначен как удобство для программиста, однако мы документировали факт, что никакие стандартные слова не используют его.

A.6.2.2008 PARSE
Типичное использование:
        char PARSE ccc<char>

Традиционное слово Forth для синтаксического анализа это WORD. PARSE решает следующие проблемы WORD:

a) WORD всегда пропускает ведущие разделители. Это поведение соответствует использованию текстовым интерпретатором, который ищет последовательности непустых символов, но не соответствует для использования словами типа (, .(, и ." . Рассмотрите следующее (испорченное) определение .(:

        : .([CHAR]) WORD COUNT TYPE ; IMMEDIATE

Это работает прекрасно, когда используется в строке подобно:

        .(HELLO)   5 .

но рассмотрите то, что случается, если пользователь вводит пустую строку:

        .( )   5 .

Определение .(показанное выше обработало бы) как ведущий разделитель, пропустило его, и продолжило потреблять символы, пока оно не обнаружило бы другой) который следовал за не-) символом, или пока не опустела бы область анализа. В показанном примере 5 . было бы обработано как часть из строки, которая будет напечатана.

С PARSE, мы могли бы записать правильное определение .(:

        : .([CHAR]) PARSE TYPE ; IMMEDIATE

Это определение избегает аномалии "пустой строки".

b) WORD возвращает свой результат как строку со счетчиком. Это имеет четыре плохих эффекта:

1) Символы принятые WORD должны быть скопированы из входного буфера во временный буфер, чтобы создать место для символа счетчика, который должен быть в начале строки со счетчиком. Шаг копирования неэффективен в сравнении с PARSE, который оставляет строку во входном буфере и не должен копировать ее куда-нибудь.

2) WORD должно быть внимательным, чтобы не сохранить слишком много символов во временный буфер, и таким образом не перезаписать кое-чего вне конца буфера. Это добавляется к накладным расходам из шага копирования. (WORD, вероятно, придется просматривать много символов перед поиском конечного разделителя.)

3) Символ счетчика ограничивает длину строки, возвращаемой WORD до 255 символов (более длинные строки могут легко быть сохранены в блоках!). Это ограничение не существует для PARSE.

4) Временный буфер типично перезаписывается следующим использованием WORD. Это создает временную зависимость; значение, возвращенное WORD допустимо только для ограниченной продолжительности. PARSE также имеет временную зависимость связанную с продолжительностью жизни входного буфера, но это менее серьезно в большинстве случаев, чем временная зависимость WORD.

Поведение WORD относительно пропуска ведущих разделителей полезно для синтаксического анализа ограниченных пробелом имен. Много системных реализаций включают дополнительное слово для этой цели, подобное PARSE относительно

возвращаемого значения "c-addr u", но без явного параметра разделителя (набор разделителей - неявно "пробельные символы"), и который пропускает ведущие разделители. Обычное описание для этого слова:

```
PARSE-WORD  ( "<spaces>name" -- c-addr u )
```

Пропускает ведущие пробелы и выделяет имя ограниченное пробелом. c-addr - адрес в пределах входного буфера и u - длина выбранной строки. Если область анализа пуста, результирующая строка имеет нулевую длину.

Если и PARSE и PARSE-WORD присутствуют, потребность в WORD в значительной степени устранена.

A.6.2.2030 PICK
0 PICK эквивалентно DUP, и 1 PICK эквивалентно OVER.

A.6.2.2040 QUERY
Функция QUERY может быть выполнена с ACCEPT и EVALUATE.

A.6.2.2125 REFILL
Это слово - полезное обобщение QUERY. Переопределение QUERY для выполнения этой спецификации сломала бы существующий код. REFILL разработан, чтобы вести себя разумно для всех возможных входных источников. Если входной источник исходит от пользователя, как с QUERY, REFILL может все еще возвращать значение false если, например, канал связи закрыт, так чтобы система знала, что ввод больше не будет доступен.

A.6.2.2150 ROLL
2 ROLL эквивалентно ROT, 1 ROLL эквивалентно SWAP, и 0 ROLL - пустая операция.

A.6.2.2182 SAVE-INPUT
SAVE-INPUT и RESTORE-INPUT позволяют ту же самую степень переустановления входного источника в пределах текстового файла, как это доступно с BLOCK вводом. SAVE-INPUT и RESTORE-INPUT "скрывают подробности" операций, необходимых для совершения этого переустановления, и используются одинаковым способом со всеми входными источниками. Это упрощает переустановку входного источника для программ, потому что они не должны проверять несколько переменных и предпринимать различные действия в зависимости от значений этих переменных.

SAVE-INPUT и RESTORE-INPUT предназначены для переустановления в пределах одного входного источника; например, следующий сценарий НЕ позволяется для Стандартной Программы:

```
: XX
   SAVE-INPUT  CREATE
   S" RESTORE-INPUT" EVALUATE
   ABORT" couldn't restore input"
;
```

Это неправильно, потому что во время выполнения RESTORE-INPUT входной источник - строка через EVALUATE, которая - не тот же самый входной источник, который был в действительности, когда было выполнено SAVE-INPUT .

Следующий код позволяется:

```
: XX
   SAVE-INPUT  CREATE
   S" .( Hello)" EVALUATE
   RESTORE-INPUT ABORT" couldn't restore input"
;
```

После возвращения из EVALUATE, спецификация входного источника восстановлена к ее предыдущему состоянию, таким образом SAVE-INPUT и RESTORE-INPUT вызваны действительно с одним и тем же входным источником.

В вышеупомянутых примерах, EVALUATE фраза могла быть заменена фразой включающей INCLUDE-FILE, и применились бы те же самые правила.

Стандарт не определяет то, что случается, если программа нарушает вышеупомянутые правила. Стандартная Система могла бы проверять нарушение и возвращать индикацию исключения из RESTORE-INPUT, или она могла бы терпеть неудачу непредсказуемым способом.

Возвращаемое значение из RESTORE-INPUT прежде всего предназначено для сообщения о случае, где программа пытается восстановить позицию входного источника, чья позиция не может быть восстановлена. Таким входным источником могла бы быть клавиатура.

Вложение SAVE-INPUT и RESTORE-INPUT позволяется. Например, следующая ситуация работает как ожидается:

```
: XX
   SAVE-INPUT
      S" f1" INCLUDED
      \ The file "f1" includes:
      \   ... SAVE-INPUT ... RESTORE-INPUT ...
      \ End of file "f1"
   RESTORE-INPUT  ABORT" couldn't restore input"
;
```

В принципе, RESTORE-INPUT могло быть реализовано как "всегда сбой", например:

```
: RESTORE-INPUT  ( x1 ... xn n -- flag )
   0 ?DO DROP LOOP TRUE
;
```

Такая реализация не была бы полезна в большинстве случаев. Было бы предпочтительнее для системы скорее оставить SAVE-INPUT и RESTORE-INPUT неопределенными, чем создавать бесполезную реализацию. В отсутствии слов, прикладной программист может выбирать, действительно ли создавать "фиктивные" реализации или обойти проблему некоторым другим путем.

Примеры того, как реализация могла бы использовать возвращаемые значения из SAVE-INPUT для выполнения функции сохранения/восстановления:

Входной источник	возможные значения стека
блок	>IN @ BLK @ 2
EVALUATE	>IN @ 1
клавиатура	>IN @ 1
текстовый файл	>IN @ lo-pos hi-pos 3

Это только примеры; Стандартная Программа не предполагает какое либо конкретное значение для отдельных элементов стека, возвращаемых SAVE-INPUT.

A.6.2.2290 TIB
Функция TIB была заменена SOURCE.

A.6.2.2295 TO
Исторически, некоторые реализации TO не имеют явного выделения имени. Вместо этого они устанавливают флаг режима, который проверяется последующим выполнением имени. ANS Forth явно требует, чтобы TO выделял имя, так чтобы эффект TO был предсказуем, когда он используется в конце области анализа.

Типичное использование:
```
x TO name
```

A.6.2.2298 TRUE
TRUE эквивалентно фразе 0 0=.

A.6.2.2405 VALUE
Типичное использование:

```
0 VALUE DATA

: EXCHANGE ( n1 -- n2 ) DATA SWAP TO DATA ;
```

EXCHANGE оставляет n1 в DATA и возвращает предшествующее значение n2.

A.6.2.2440 WITHIN
Мы описываем WITHIN без упоминания кругового числового пространства (неопределенный термин) или предоставления кода. Вот - числовая строка с точкой переполнения (o) в крайнем правом положении и точкой антипереполнения (u) в крайнем левом:

```
u-----------------------------------------------------------------o
```

Есть два случая для рассмотрения: либо диапазон n2|u2..n3|u3 охватывает точки переполнения/антипереполнения, либо нет. Позвольте исследовать сначала не охватывающий случай:

```
u------------------[.....................)------------------------o
```

[обозначает n2|u2,) обозначает n3|u3, и точки и [- это числа В ПРЕДЕЛАХ (WITHIN) диапазона. n3|u3 больше чем n2|u2, так что следующие проверки определят, является ли n1|u1 В ПРЕДЕЛАХ (WITHIN) n2|u2 и n3|u3:

```
n2|u2 < n1|u1 и n1|u1 < n3|u3.
```

В случае, где диапазон сравнения колеблется между точками переполнения/антипереполнения:

```
u..............)----------------------------[...................o
```

n3|u3 - меньше чем n2|u2, и следующие проверки определят, является ли n1|u1 В ПРЕДЕЛАХ (WITHIN) n2|u2 и n3|u3:

```
n2|u2 < n1|u1 или n1|u1 < n3|u3.
```


WITHIN должен работать для знаковых и без знаковых параметров. Одна очевидная реализация не работает:

```
: WITHIN  ( test low high -- flag )
   >R  OVER < 0= ( test flag1 )  SWAP R> < ( flag1 flag2 ) AND
;
```

Примите арифметику дополнительного кода (дополнение до двух) на 16-бит машине, и рассмотрите следующую проверку:

```
        33000  32000 34000  WITHIN
```

Вышеупомянутая реализация возвращает false для этой проверки, даже притом, что число без знака 33000 - ясно в пределах диапазона {{32000 .. 34000}}.

Проблема состоит в том, что на неправильной реализации, знаковое сравнение < дает неправильный ответ, когда 32000 сравнивается с 33000, потому что когда эти числа рассматриваются как числа со знаком, 33000 рассматривается как отрицательное 32536, в то время как 32000 остается положительным.

Замена < на U< в вышеупомянутой реализации заставляет ее работать с числами без знака, но вызывает проблемы с некоторыми диапазонами чисел со знаком; в частности испытание:

```
        1  -5  5  WITHIN
```

дало бы неправильный ответ.

Для машин с дополнительным кодом (дополнение до двух), которые игнорируют арифметическое переполнение (большинство машин), следующая реализация работает во всех случаях:

```
        :  WITHIN  ( test low high -- flag )   OVER - >R - R>  U<  ;
```

A.6.2.2530 [COMPILE]
Типичное использование:

```
        : name2 ... [COMPILE] name1 ... ;  IMMEDIATE
```

A.6.2.2535 \
Типичное использование:

```
        5 CONSTANT THAT  \ЭТО - КОММЕНТАРИЙ О THAT
```

A.7 Дополнительный Блочный набор слов

Ранние системы Forth выполнялись автономно, без ведущей OS. Блоки 1024 байт были разработаны как удобный элемент диска, и большинство автономных систем Forth все еще используют их. Это относительно простой для записи непосредственный дисковый драйвер, который отображает адреса головка/дорожка/сектор на номера блоков. Такие дисковые драйверы чрезвычайно быстры по сравнению с обычными файл-ориентированными операционными системами, и защита высока, потому что нет никого доверия до карты диска.

Сегодня много реализаций Forth выполняются под ведущими операционными системами, потому что совместимость, которую они предлагают пользователю, перевешивает ухудшение производительности. Много людей использующих такие системы предпочитают использовать только файлы ведущей OS; однако люди, которые используют, автономные и не автономные Forth нуждаются в совместимом способе доступа к диску. Блочный набор слов включает наиболее общие слова для доступа к источнику программы и данным на диске.

Чтобы гарантировать, что Стандартные Программы, нуждающиеся в доступе к запоминающему устройству, имеют механизм соответствующий автономным и не автономным реализациям, ANS Forth требует, что бы Блочный набор слов был доступен, если предусмотрены любые средства запоминающего устройства. На не автономных реализациях, блоки обычно находятся в файлах ведущей OS.

A.7.2 Дополнительные термины

блок
Много систем Forth используют блоки, чтобы содержать источник программы.

Традиционно такие блоки форматированы для редактирования как 16 строк по 64 символа. Исходные блоки часто упоминаются как "экраны".

A.7.6 Словарь

A.7.6.2.2190 SCR
SCR сокращение для экрана.

A.8 Дополнительный набор слов двойных чисел

Системы Forth на 8-бит и 16-бит процессорах часто находят необходимым иметь дело с числами двойной-длины. Но много Forth на маленьких внедренных системах не желают, и много пользователей Forth на системах с размером ячейки 32-бит или больше находят, что потребность в числах двойной длины очень незначительная. Поэтому мы выделили слова, которые управляют объектами двойной длины в этот дополнительный набор слов.

Пожалуйста заметьте, что соглашение об именах, используемое в этом наборе слов передает некоторую важную информацию:

1. Слова, чьи имена имеют форму 2xxx, имеют дело с парой-ячеек, где отношения между ячейками не определено. Они могут быть двумя векторами, числами двойной длины, или любой парой ячеек, которыми удобно манипулировать вместе.

2. Слова с именами формы Dxxx имеют дело определенно с целыми числами двойной длины.

3. Слова с именами формы Mxxx имеют дело с некоторой комбинацией одинарных и двойных целых чисел. Порядок, в котором они появляются на стеке, определен давней общей практикой.

Обратитесь к A.3.1 для обсуждения типов данных в Forth.

A.8.6 Словарь

A.8.6.1.0360 2CONSTANT
Типичное использование:
```
x1 x2 2CONSTANT name
```

A.8.6.1.0390 2LITERAL
Типичное использование:
```
: X ... [ x1 x2 ] 2LITERAL ... ;
```

A.8.6.1.0440 2VARIABLE
Типичное использование:
```
2VARIABLE name
```

A.8.6.1.1070 D.R
B D.R, "R" сокращение для RIGHT.

A.8.6.1.1090 D2*
См.: A.6.1.0320 2* для соответствующего обсуждения.

A.8.6.1.1100 D2/
См.: A.6.1.0330 2/ для соответствующего обсуждения.

A.8.6.1.1140 D>S
Существуют представления чисел, например представление величины со знаком, где сокращение от двойной к одинарной точности не может быть сделано просто с помощью DROP. Это слово, эквивалентное DROP на системах с дополнительным кодом (дополнение до двух), уменьшает чувствительность кода приложения к

представлению чисел и облегчает переносимость.

A.8.6.1.1820 M*/
M*/ было однажды описано Chuck Moore как наиболее полезный арифметический оператор в Forth. Оно является главной рабочей лошадкой в большинстве вычислений, вовлекающих числа две-ячейки. Заметьте, что некоторые системы разрешают знаковые делители. Это может много стоить в производительности на некоторых CPUs. Не доказано, что требование положительного делителя - это проблема.

A.8.6.1.1830 M+
M+ - это классический метод для интегрирования.

A.9 Дополнительный набор слов исключений

CATCH и THROW предусматривают надежный механизм для обработки исключительных ситуаций, без того чтобы иметь необходимость размножать флаги исключения через многочисленные уровни вложения слова. Это подобно основной тенденции механизмов "нелокального возвращения" многих других языков, типа C setjmp() и longjmp(), и LISP CATCH и THROW. В контексте Forth, THROW может быть описано как "многоуровневый EXIT" с CATCH, отмечающим местоположение, к которому THROW может возвращаться.

Немного подобные "многоуровневые EXIT" Forth схемы обработки особых ситуаций были описаны и использовались в прошлые годы. Не возможно осуществить такую схему, используя только стандартные слова (иные, чем CATCH и THROW), потому что нет никакого переносимого способа "отмотать" стек возвратов к предопределенному месту.

THROW также предоставляет удобную методику реализации для стандартных слов ABORT и ABORT", позволяя приложению определять, с помощью использования CATCH, поведение при событии системного ABORT.

Эта типовая реализация CATCH и THROW использует нестандартные слова, описанные ниже. Они или их эквиваленты доступны во многих системах. Другие стратегии реализации, включающие непосредственное сохранение значения DEPTH возможны, если такие слова не доступны.

SP@ (-- addr) возвращает адрес, соответствующий вершине стека данных.

SP! (addr --) устанавливает указатель стека на addr, таким образом восстанавливая глубину стека на ту же самую глубину, которая существовала как раз перед тем как был получен addr выполнением SP@.

RP@ (-- addr) возвращает адрес, соответствующий вершине стека возвратов.

RP! (addr --) устанавливает указатель стека возвратов на addr, таким образом восстанавливая глубину стека возвратов на ту же самую глубину, которая существовала как раз перед тем как был получен addr выполнением RP@.

```
VARIABLE HANDLER   0 HANDLER !  \ последний обработчик особых ситуаций

: CATCH  ( xt -- exception# | 0 ) \ возвращает addr на стеке
   SP@ >R         ( xt ) \ сохраняет указатель вершины стека данных
   HANDLER @ >R   ( xt ) \ и предыдущий обработчик
   RP@ HANDLER !  ( xt ) \ устанавливает текущий обработчик
   EXECUTE        ( )    \ выполняет возврат если не THROW
   R> HANDLER !   ( )    \ восстанавливает предыдущий обработчик
   R> DROP        ( )    \ удаляет сохраненный стек ptr
   0              ( 0 )  \ нормальное завершение
;
```

```
: THROW  ( ??? exception# -- ??? exception# )
   ?DUP IF          ( exc# ) \ 0 THROW - пустая команда
     HANDLER @ RP!  ( exc# ) \ восстанавливает предыд. стек возвратов
     R> HANDLER !   ( exc# ) \ восстанавливает предыд. обработчик
     R> SWAP >R     ( saved-sp ) \ exc# на стеке возвратов
     SP! DROP R>    ( exc# ) \ восстановление стека
     \ Возвращение к программе вызвавшей CATCH, потому что стек
     \ возвратов
     \ восстановлен к состоянию, которое существовало
     \ когда CATCH начал выполнение
   THEN
;
```

В многозадачной системе, переменная HANDLER должна быть переменной в в-задачной области (то есть, пользовательская переменная).

Эта типовая реализация не обрабатывает явно случай, в котором CATCH никогда не вызывается (то есть, поведение ABORT). Одно решение состоит в том, чтобы добавить следующий код после IF в THROW:

```
HANDLER @ 0= IF ( освободить стек ) QUIT THEN
```

Другое решение состоит в том, чтобы выполнить CATCH в пределах QUIT так, чтобы был всегда в наличии "обработчик особых ситуаций последнего средства". Например:

```
: QUIT
    ( освободить стек возвратов и )
    ( установить входной источник на пользовательское устройство ввода
      данных )
   POSTPONE [
   BEGIN
     REFILL
   WHILE
     ['] INTERPRET  CATCH
     CASE
        0 OF STATE @ 0= IF ." OK" THEN CR  ENDOF
       -1 OF ( Прервано ) ENDOF
       -2 OF ( отображает сообщение от ABORT" ) ENDOF
       ( по умолчанию ) DUP ." Exception # "  .
     ENDCASE
   REPEAT BYE
;
```

Этот пример предполагает существование системной реализации слова INTERPRET, которое воплощает семантику текстового интерпретатора, описанную в 3.4 Интерпретатор текста Forth. Заметьте, что эта реализация QUIT автоматически обрабатывает освобождение стека и стека возвратов, из-за свойственного THROW восстановления стеков данных и возвратов. Данным этим определением QUIT, просто определить:

```
: ABORT  -1 THROW ;
```

В системах с другими стеками в дополнение к стекам данных и возвратов, реализации из CATCH и THROW должны также сохранять и восстанавливать эти указатели вершин стеков. Такая "расширенная версия" может быть надстроена сверху этой основной реализации. Например, с другим указателем вершины стека доступным с помощью FP@ и FP! должен быть переопределен только CATCH:

```
: CATCH  ( xt -- exception# | 0 )
```

```
FP@ >R  CATCH  R> OVER IF FP! ELSE DROP THEN ;
```

В этом случае нет необходимости, ни в каких изменениях THROW. Заметьте что, как со всеми переопределениями, переопределенная версия CATCH будет доступна только определениям, откомпилированным после переопределения CATCH.

CATCH и THROW предоставляют удобный путь для реализаций "очистки" состояния открытых файлов, если исключение происходит в течении интерпретации текстового файла с помощью INCLUDE-FILE. Реализация INCLUDE-FILE может предохранять (с CATCH) слово, которое исполняет текстовую интерпретацию, и если CATCH возвращает код исключения, файл может быть закрыт и исключение переброшено (reTHROWn) так, чтобы файлы, включаемые во внешнем уровне вложенности, могли быть также закрыты.

Заметьте, что Стандарт позволяет, но не требует от INCLUDE-FILE закрывать его открытые файлы, если происходит исключение. Однако он требует от INCLUDE-FILE понизить уровень вложения спецификации входного источника, если исключение обработано THROW.

A.9.3 Дополнительные условия применения

Одно важное использование обработчика особых ситуаций - это поддержание управления программой под многими условиями, которые вызывают ABORT. Это реально только, если зарезервирован диапазон кодов. Заметьте, что приложение может перезагружать много стандартных слов таким способом, чтобы THROW неопределенные ситуации не нормальным образом обрабатывались с помощью THROW специфической системой.

A.9.3.6 Обработка особых ситуаций

Метод выполнения этой связи зависящий от реализации. Например, LOAD может "знать" о CATCH и THROW (используя CATCH самостоятельно, например), или CATCH и THROW могут "знать" о LOAD (поддерживая информацию вложенности входного источника в структуре данных известной в THROW, например). При этих обстоятельствах не возможно для стандартной Программы определить слова типа LOAD полностью переносимым способом.

A.9.6 Словарь

A.9.6.1.2275 THROW

Если THROW выполнено с не нулевым параметром, эффект такой, как будто это возвратило соответствующее CATCH . В том случае глубина стека такая же, какая она была как раз перед началом выполнения CATCH . Значения параметров стека i*x могли изменяться произвольно в течение выполнения xt. Вообще, ничего полезного не может быть сделано с этими элементами стека, но так как их число известно (потому что глубина стека детерминирована), приложение может DROP их, чтобы возвратиться к предсказуемому состоянию стека.

Типичное использование:

```
: could-fail ( -- char )
   KEY DUP [CHAR] Q =  IF  1 THROW THEN ;

: do-it ( a b -- c)   2DROP could-fail ;

: try-it ( --)
   1 2 ['] do-it  CATCH  IF
     ( x1 x2 ) 2DROP ." Было исключение" CR
   ELSE ." Был символ " EMIT CR
   THEN
;
```

```
: retry-it ( -- )
   BEGIN  1 2 ['] do-it CATCH  WHILE
     ( x1 x2) 2DROP  ." Исключение, пробуем продолжать" CR
   REPEAT ( char )
   ." Был символ " EMIT CR
;
```


A.10 Дополнительный сервисный набор слов

A.10.6 Словарь

A.10.6.1.0742 AT-XY
Большинство разработчиков поставляют метод позиционирования курсора на экране CRT, но есть большое разногласие в именах и параметрах стека. Эта версия поддерживается, по крайней мере, одним главным поставщиком.

A.10.6.1.1755 KEY?
Технический Комитет ходил по кругу некоторое время по поводу стекового эффекта. Любое решение нарушит чью-то практику и накажет некоторую машину. Этот путь не касается опережающего ввода с клавиатуры на некоторых системах, когда требуется реализация односимвольного буфера на машинах, где опрос клавиатуры неизбежно оканчивается разрушением символа.

Использование KEY или KEY? указывает, что приложение не желает беспокоиться о не символьных событиях, так что они отвергаются в ожидании, в конечном счете, получения допустимого символа. Приложения, желающие обработать не символьные события, должны использовать EKEY и EKEY?. Возможно смешивать использование KEY? / KEY и EKEY? / EKEY в пределах отдельного приложения, но приложение должно использовать KEY? и KEY только, когда оно желает удалить не символьные события до получения допустимого символа.

A.10.6.2.1305 EKEY
EKEY предоставляет стандартное слово для обращения к зависимому от системы набору "необработанных" событий клавиатуры, включая события соответствующие членам стандартного набора символов, события соответствующие другим членам определенного реализацией набора символов, и нажатия клавиш которые не соответствуют членам набора символов.

EKEY не предполагает никакого специфического числового соответствия между специфическими значениями кода события и значениями, представляющими стандартные символы. На некоторых системах, оно может допускать две отдельные клавиши, которые соответствуют одному и тому же стандартному символу, которые нужно отличить друг от друга.

В системах, которые объединяют события клавиатуры и мыши в один "поток событий", единственное число, возвращенное EKEY, может быть неадекватно для представления полного диапазона возможного ввода. В таких системах, отдельная "запись события" может включать временную отметку, x,y координаты позиции мыши, состояния клавиатуры, и состояния кнопок мыши. В таких системах могло бы быть соответствующее EKEY для возвращения адреса "записи события" из которой могла быть извлечена другая информация.

Также рассмотрите гипотетическую систему Forth, выполняющуюся под MS-DOS на PC совместимом компьютере. Предположите, что набор символов определенный реализацией - "нормальный" PC 8-бит набор символов. В этом наборе символов коды от 0 до 127 соответствуют ASCII символам. Коды от 128 до 255 представляют символы из различных не английских языков, математические символы, и некоторые графические символы, используемые для рисования в строке. В добавление к этим символам, клавиатура может генерировать различные другие "скэн-коды", представляющие такие не символьные события как клавиши курсора и функциональные

клавиши.

Могут быть составные клавиши, с различными скэн-кодами, соответствующими одному и тому же стандартному символу. Например, символ представляющий число "1" часто появляется в строке цифровых клавиш выше алфавитных клавиш, и также в отдельной цифровой клавиатуре.

Когда программа опрашивает операционную систему MS-DOS для события клавиатуры, она получает единственный ненулевой байт представляющий символ, или нулевой байт, сопровождаемый байтом "скэн-кода", представляющим не символьное событие клавиатуры (например, функциональная клавиша).

EKEY представляет каждое событие клавиатуры скорее как отдельное число, чем как последовательность чисел. Для системы описанной выше, следующее было бы разумной реализацией EKEY и связанных слов:

Запрос к окружению MAX-CHAR возвратил бы 255.

Предположите существование слова DOS-KEY (-- char) которое выполняет системный вызов MS-DOS "Прямой Ввод STDIN" (Прерывание 21h, Функция 07h) и слово DOS-KEY? (-- flag) которое выполняет системный вызов MS-DOS "Проверка Состояния STDIN" (Прерывание 21h, Функция 0Bh).

```
: EKEY?  ( -- flag )  DOS-KEY?  0<>  ;

: EKEY  ( -- u )  DOS-KEY  ?DUP 0= IF  DOS-KEY 256 +  THEN ;

: EKEY>CHAR  ( u -- u false | char true )
   DUP 255 > IF          ( u )
     DUP 259 = IF         \ 259 это Ctrl-@ (ASCII NUL)
       DROP 0 TRUE EXIT   \ если так - замена символом
     THEN FALSE EXIT      \ иначе расширенный символ
   THEN  TRUE             \ нормальный символ ASCII.
;

VARIABLE PENDING-CHAR   -1 PENDING-CHAR !

: KEY?  ( -- flag )
   PENDING-CHAR @ 0< IF
      BEGIN  EKEY? WHILE
        EKEY EKEY>CHAR IF
          PENDING-CHAR !  TRUE EXIT
        THEN DROP
      REPEAT  FALSE EXIT
   THEN  TRUE
;

: KEY  ( -- char )
   PENDING-CHAR @ 0< IF
      BEGIN  EKEY  EKEY>CHAR 0= WHILE
        DROP
      REPEAT  EXIT
   THEN  PENDING-CHAR @  -1 PENDING-CHAR !
;
```


Это - полнофункциональная реализация, предоставляющая прикладной программе простой путь, или обрабатывать не символьные события (с помощью EKEY), или игнорировать их и только рассматривать "реальные" символы (с помощью KEY).

Заметьте, что EKEY отображает скэн-коды от 0 до 255 в числа от 256 до 511. EKEY отображает число 259, представляя комбинацию клавиатуры Ctrl-Shift-@ в символ, чье числовое значение - 0 (ASCII NUL). Много ASCII клавиатур генерируют ASCII NUL для Ctrl-Shift-@, так что мы используем эту комбинацию клавиш для ASCII NUL (который иначе недоступен из MS-DOS, потому что нулевой байт показывает, что следует другой байт скэн-кода).

Одно следствие использования системного вызова "Прямого Ввода STDIN" (функция 7) вместо системного вызова "Ввода STDIN" (функция 8) - это что нормальное поведение DOS "прерывания Ctrl-C" - заблокировано, когда система ожидает ввода (Ctrl-C все еще вызвало бы прерывание, пока символы выводятся). С другой стороны, если системный вызов "Ввод STDIN" (функция 8) использовался для реализации EKEY, Ctrl-C прерывание было бы доступно, но Ctrl-Shift-@ также вызвало бы прерывание, потому что операционная система обработает второй байт из 0,3 последовательности как Ctrl-C даже притом, что эти 3 - действительно скэн-код, а не символ. Одно решение "лучшее из обоих миров" - это использовать функцию 8 для первого байта получаемого EKEY, и функцию 7 для байта скэн-кода. Например:

```
: EKEY  ( -- u )
   DOS-KEY-FUNCTION-8  ?DUP  0=  IF
     DOS-KEY-FUNCTION-7  DUP 3  =  IF
       DROP 0  ELSE  256 +
     THEN
   THEN
;
```

Конечно, если разработчик Forth выбирает передачу Ctrl-C через программу, без использования ее обычной функции обработки прерывания, тогда функция 7 DOS соответствует обоим случаям (и некоторая дополнительная забота должна быть предпринята для предотвращения опережающего ввода с клавиатуры Ctrl-C от прерывания системы Forth в течение операций вывода).

Система Forth могла бы также выбирать более простую реализацию KEY, без реализации EKEY, следующим образом:

```
: KEY   ( -- char )  DOS-KEY  ;

: KEY?  ( -- flag )  DOS-KEY? 0<>  ;
```

Недостатки более простой версии:

a) Прикладная программа, которая использует KEY, ожидая получения только допустимых символов, могла бы получить последовательность байт (например, нулевой байт, сопровождаемый байтом с таким же числовым значением как у символа "A") которая, кажется, содержит допустимый символ, даже притом, что пользователь нажал клавишу (например, функциональная клавиша 4) которая не соответствует никакому допустимому символу.

b) Прикладная программа, которая желает обработать не символьные события, будет должна выполнить KEY дважды, если она возвращает нуль первый раз. Это, могло бы казалось, быть разумной и простой для воплощения вещью. Однако, такой код - не переносимый на другие системы, которые не используют нулевой байт как "escape" код.

Используя подход EKEY, алгоритм для обработки событий клавиатуры может быть один и тот же для всех систем; системные зависимости могут быть уменьшены до таблицы или набора списка констант системно зависимых кодов клавиш используемых для доступа к специфическим прикладным функциям. Без EKEY алгоритм, не только таблица, вероятно будет системно зависимым.

Другой подход к EKEY на MS-DOS состоит в том, чтобы использовать функцию BIOS "Read Keyboard Status" (Прерывание 16h, Функция 01h) или связанную функцию "Check Keyboard" (Прерывание 16h, Функция 11h). Преимущество этой функции состоит в том, что она позволяет программе делать различие между различными клавишами, которые соответствуют одному и тому же символу (например, две клавиши "1"). Недостаток - то, что функция клавиатуры BIOS доступна только для чтения клавиатуры. Она не может быть "переадресована" к другому источнику "стандартного ввода", как может функция DOS Ввод STDIN.

A.10.6.2.1306 EKEY>CHAR
EKEY>CHAR транслирует событие клавиатуры в соответствующего члена набора символов, если такое соответствие существует для этого события.

Возможно, что несколько различных событий клавиатуры могут соответствовать одному и тому же символу, и другие события клавиатуры могут не соответствовать никакому символу.

A.10.6.2.1325 EMIT?
Неопределенная задержка - это состояние связанное с устройством, типа принтер недоступен, которое требует вмешательства оператора, перед тем как устройство примет новые данные.

A.10.6.2.1905 MS
Каждая система имеет часы, хотя их частоты различаются. Так как много программ нуждаются в интервалах времени, предлагается это слово. Использование миллисекунд как внутреннего элемента времени - это практичный "наименьший общий знаменатель" внешнего элемента. Принятое разработчиками, оно будет использовать "такты системных часов" (безотносительно их размера) как внутренний элемент и соответствующим образом преобразовывать.

A.10.6.2.2292 TIME&DATE
Большинство систем имеет часы/календарь реального времени. Это слово дает переносимый доступ к нему.

A.11 Дополнительный набор слов доступа к файлам

Много систем Forth поддерживает доступ к базовой файловой системе, и много из них поддерживает интерпретацию Forth из файлов исходного текста. Forth-83 Стандарт не адресовал файлы базовой OS. Тем не менее, степень подобия существует среди современных реализаций.

Например, файлы должны быть открыты и закрыты, созданы и удалены. Реализации файловой системы Forth отличаются главным образом в обработке и расположении кодов исключений, и в формате строк идентифицирующих файлы. Основной механизм для создания блоков управления файлом мог бы или не мог бы быть видимым. Мы хотели сохранить его невидимым.

Файлы должны также читаться и писаться. Текстовые файлы, если поддерживается, должны читаться и писаться одной строкой за раз. Интерпретация текстовых файлов подразумевает, что они так или иначе интегрированы в механизм ввода текстового интерпретатора. Эти и другие требования оформлены в расширение набора слов доступа к файлу.

Большинство изученных существующих реализаций используют простые Английские слова для общих базовых файловых функций: OPEN, CLOSE, READ, и т.д. Хотя мы предпочли бы делать аналогично, было бы так много незначительных изменений в реализации этих слов, что принятие любого специфического значения сломала бы много существующего кода. Мы использовали названия с суффиксом -FILE для большинства этих слов. Мы поощряем разработчиков приспосабливать их однословные примитивы к ANS поведению, и надеемся что, если это сделано на широко

распространенном основании мы можем принять лучшие имена определений в будущем стандарте.

Специфические разъяснения для членов этого набора слов следуют далее.

A.11.3 Дополнительные условия применения

A.11.3.2 Блоки в файлах

Много систем многократно используют идентификаторы файла; когда файл закрыт, впоследствии открытый файл может получить тот же самый идентификатор. Если первоначальный файл имеет блоки все еще в блочных буферах, они будут неправильно связаны с недавно открытым файлом с бедственным результатом. Система блочных буферов должна быть сброшена на диск, чтобы избежать этого.

A.11.6 Словарь

A.11.6.1.0765 BIN
Некоторые операционные системы требуют, чтобы файлы были открыты в различном режиме, чтобы обратиться к их содержанию как неструктурированному потоку двоичных данных скорее, чем как к последовательности строк.

Параметры для READ-FILE и WRITE-FILE - массивы символьных запоминающих элементов, каждый элемент состоит, по крайней мере, из 8 бит. Технический Комитет подразумевает что, в режиме BIN, содержание этих запоминающих элементов может быть записано в файл и позже прочитано назад без чередования. Технический Комитет отказался решать проблемы относительно воздействия "wide" символов в Файловых и Блочных наборах слов.

A.11.6.1.1010 CREATE-FILE
Типичное использование:
```
: X .. S" TEST.FTH" R/W CREATE-FILE  ABORT" CREATE-FILE FAILED"
... ;
```

A.11.6.1.1717 INCLUDE-FILE
Вот - две альтернативы реализации для сохранения спецификации входного источника при наличии ввода текстового файла:

1) Сохранить позицию файла (как возвращено FILE-POSITION) начала интерпретируемой строки. Для восстановления спецификации входного источника, искать эту позицию и перечитать строку во входной буфер.

2) Распределить отдельный буфер строки для каждого активного текстового входного файла, используя этот буфер как входной буфер. Этот метод избегает шагов "искать и перечитывать", и позволяет использовать "псевдофайлы" типа pipe и другие каналы связи только-последовательного-доступа.

A.11.6.1.1718 INCLUDED
Типичное использование:
```
... S" filename" INCLUDED ...
```

A.11.6.1.1970 OPEN-FILE
Типичное использование:
```
: X .. S" TEST.FTH" R/W OPEN-FILE  ABORT" OPEN-FILE FAILED" ... ;
```

A.11.6.1.2080 READ-FILE
Типичный последовательный алгоритм обработки файла мог бы напоминать:

```
BEGIN                   (  )
... READ-FILE THROW     ( длина )
?DUP WHILE              ( длина )
```

```
        ...                     (  )
        REPEAT                  (  )
```

В этом примере THROW используется для обработки (неожиданных) исключительных ситуаций, которые сообщаются как ненулевые значения возвращаемого значения ior из READ-FILE. Конец файла сообщается как нулевое значение "длины" возвращаемого значения.

A.11.6.1.2090 READ-LINE
Реализациям разрешено сохранять ограничитель строки в буфере памяти, чтобы позволить использовать функции чтения строки, предоставленные базовыми операционными системами, некоторые из которых сохраняют ограничитель. Без этого условия, мог бы быть необходим временный буфер. Двух символьное ограничение достаточно для подавляющего большинства существующих операционных систем. Реализации на базовых операционных системах, чья последовательность ограничителей строки больше двух символов, могут иметь специальное действие для предотвращения запоминания больше двух символов ограничителей.

Стандартные Программы не могут зависеть от присутствия никакой такой последовательности ограничителей в буфере.

Типичный последовательный строчно-ориентированный алгоритм обработки файла мог бы напоминать:

```
        BEGIN                   (  )
        . . . READ-LINE  THROW ( длина not-eof-flag )
        WHILE                   ( длина )
        . . .                   (  )
        REPEAT DROP             (  )
```


В этом примере THROW используется для обработки (неожиданной) исключительной ситуации I/O, которая сообщается как ненулевое значение возвращаемого значения "ior" из READ-LINE.

READ-LINE нуждается в отдельном флаге конца файла, потому что пустая (нулевая длина) строка - стандартное событие, таким образом строка нулевой длины не может использоваться для обозначения конца файла.

A.11.6.1.2165 S"
Типичное использование:

```
        ... S" ccc" ...
```

Семантика интерпретации для S" предназначена для предоставления простого механизма ввода строки в состоянии интерпретации. Так как реализация может предоставлять только один буфер для интерпретируемых строк, интерпретируемая строка подвержена перезаписи следующим выполнением S" в состоянии интерпретации. Подразумевается, что нет стандартного слова иного, чем S" могущего быть причиной перезаписи интерпретируемой строки. Однако, так как слова типа EVALUATE, LOAD, INCLUDE-FILE и INCLUDED могут приводить к интерпретации произвольного текста, возможно включая экземпляры S", интерпретируемая строка может быть сделана недействительной некоторыми использованиями этих слов.

Когда может возникать возможность перезаписи строки, благоразумно копировать строку в "безопасный" буфер, распределенный приложением.

Программам, желающим производить анализ способом S" советуют использовать PARSE или WORD COUNT вместо S", предотвращая перезапись интерпретируемого строкового буфера.

A.12 Дополнительный набор слов для плавающей точки

Технический Комитет рассмотрел много предложений, имеющих дело с включением и составом Наборов Слов с Плавающей точкой в ANS Forth. Хотя это обсуждалось, что ANS Forth не должен адресовать арифметику с плавающей точкой, и многочисленные приложения Forth не нуждаются в плавающей точке, имеется растущее число важных приложений Forth от электронных таблиц до научных вычислений, которые требуют использования арифметики с плавающей точкой. Первоначально Технический Комитет принял предложения, которые составили Forth Vendors Group Floating-Point Standard, впервые изданный в 1984, основу для включения плавающей точки в ANS Forth. Есть значительная общая практика и опыт с Forth Vendors Group Floating-Point Standard. Впоследствии технический Комитет принял предложения, которые помещали основную арифметику с плавающей точкой, стек и слова поддержки в наборе слов с плавающей точкой и трансцендентные функции с плавающей точкой в Расширении набора слов с плавающей точкой. Технический комитет также принял такие предложения:

- изменены имена для ясности и последовательности; например, REALS на FLOATS, и REAL+ на FLOAT+.
- удалены слова; например, FPICK.
- добавлены слова для законченности и увеличенных функциональных возможностей; например, FSINCOS, F~, DF@, DF!, SF@ и SF!.

Несколько проблем относительно набора слов с плавающей точкой были разрешены консенсусом в Техническом Комитете:

Стек с плавающей точкой: По умолчанию стек с плавающей точкой отделен от стеков данных и возвратов; однако реализация может сохранять числа с плавающей точкой на стеке данных. Программа может определять, сохраняются ли числа с плавающей точкой на стеке данных передачей строки FLOATING-STACK в ENVIRONMENT?. Это опыт нескольких членов Технического Комитета, что с надлежащей практикой кодирования, возможно написать код с плавающей точкой, который будет работать тождественно на системах с отдельным стеком с плавающей точкой и с числами с плавающей точкой сохраняемыми на стеке данных.

Ввод с плавающей точкой: Текущее основание должно быть DECIMAL. Ввод с плавающей точкой - не разрешен в произвольном основании. Все числа с плавающей точкой, которые нужно интерпретировать ANS Forth Системой должны содержать индикатор экспоненты "E" (см. 12.3.7 Преобразование входных чисел текстового интерпретатора). Консенсус в Техническом Комитете считал эту форму ввода плавающей точки в более общем использовании, чем альтернатива, которая имела бы режим ввода с плавающей точкой, который позволил бы числам с внедренными десятичными точками быть обработанными как числа с плавающей точкой.

Представление с плавающей точкой: Хотя формат и точность мантиссы и формат и диапазон экспоненты числа с плавающей точкой - определенное реализацией в ANS Forth, Расширение набора слов с плавающей точкой содержит слова DF@, SF@, DF!, и SF! для выборки и сохранения в памяти чисел с двойной и с одинарной точностью формата IEEE с плавающей точкой. Формат IEEE с плавающей точкой обычно используется числовыми математическими сопроцессорами и для обмена данными с плавающей точкой между программами и системами.

A.12.3 Дополнительные условия применения

A.12.3.5 Выравнивание адреса

В определении специализированных структур данных с плавающей точкой знайте, что CREATE не обязательно оставляет указатель области данных, выровненный для различных типов данных с плавающей точкой. Программы могут выполнять требование для различных видов выравнивания с плавающей точкой, определяя соответствующее выравнивание во время компиляции и во время выполнения.
Например:

```
: FCONSTANT ( F:  r -- )
   CREATE FALIGN  HERE  1 FLOATS ALLOT  F!
   DOES> ( F:  -- r )  FALIGNED F@ ;
```

A.12.3.7 Преобразование входных чисел текстового интерпретатора

Технический Комитет более чем однажды получал предложение, что текстовый интерпретатор в Стандартных системах Forth должен обрабатывать числа, которые имеют внедренную десятичную точку, но не экспоненту, как числа с плавающей точкой скорее, чем числа две-ячейки. Это предложение, хотя это имеет достоинство, всегда проваливалось, потому что это будет ломать очень много существующего кода; много существующих реализаций помещают полную цифровую строку на стек как двойное число и используют другие средства для информирования приложения о местоположении десятичной точки.

A.12.6 Словарь

A.12.6.1.0558 >FLOAT
>FLOAT дает возможность программам читать данные с плавающей точкой в четком ASCII формате. Оно принимает намного более широкий синтаксис, чем текстовый интерпретатор, так как последний определяет правила для создания исходных текстов программ, тогда как >FLOAT определяет правила для приема данных. >FLOAT определен так широко, как возможно разрешить ввод данных из ANS Forth Систем также как другие широко используемые стандартные среды программирования.

Это синтез обычной практики FORTRAN. Внедренные пробелы явно запрещаются в большинстве научных использований, как другие разделители полей типа запятой или наклонной черты вправо.

В то время как >FLOAT не требует обрабатывать строку пробелов как нуль, это поведение строго поощряется, так как будущая версия ANS Forth может включать такое требование.

A.12.6.1.1427 F.
Например, 1E3 F. отображает 1000..

A.12.6.1.1492 FCONSTANT
Типичное использование:
```
r FCONSTANT name
```

A.12.6.1.1552 FLITERAL
Типичное использование:
```
: X ... [ ... ( r ) ] FLITERAL ... ;
```

A.12.6.1.1630 FVARIABLE
Типичное использование:
```
FVARIABLE name
```

A.12.6.1.2143 REPRESENT
Это слово предоставляет примитив для отображения чисел с плавающей точкой. Некоторые форматы с плавающей точкой, в том числе определенные IEEE-754, позволяют представление чисел вне диапазона определенного реализацией. Они включают плюс и минус бесконечности, денормализованные числа, и другие. В этих случаях мы ожидаем, что REPRESENT будет обычно реализовано способным возвращать соответствующе символьные строки, типа "+infinity" или "nan", возможно обрезанные.

A.12.6.2.1489 FATAN2
FSINCOS и FATAN2 - комплиментарная пара операторов, которые преобразовывают углы в 2-вектора и наоборот. Они обязательны для большинства геометрических и

физических приложений, так как они правильно и однозначно обрабатывают это преобразование во всех случаях кроме нулевых векторов, даже когда тангенс угла был бы бесконечен.

FSINCOS возвращает Декартовый единичный вектор в направлении данного угла, измеренного против часовой стрелки от положительной оси X. Порядок результата на стеке, а именно y ниже x, разрешает 2-векторному типу данных дополнительно рассматриваться и использоваться как отношение аппроксимирующее тангенс угла. Таким образом, фраза FSINCOS F/ -функционально эквивалентна FTAN, но полезна только для ограниченного и прерывистого диапазона углов, тогда как FSINCOS и FATAN2 полезны для всех углов.

Это расположение было найдено удобным в течение почти двух десятилетий, и имеет дополнительную выгоду - простоту запоминания. Заключение из этого наблюдения - это что векторы вообще должны появляться на стеке в этом порядке.

Порядок параметров для FATAN2 тот же самый, преобразовывает вектор в обычное представление скалярного угла. Таким образом, для всех углов FSINCOS FATAN2 - это тождество в пределах точности арифметики и диапазона параметра FSINCOS. Заметьте что, в то время как FSINCOS всегда возвращает допустимый вектор элемента, FATAN2 примет любой ненулевой вектор. Неопределенная ситуация существует, если векторный параметр для FATAN2 имеет нулевую величину.

A.12.6.2.1516 FEXPM1
Эта функция позволяет точное вычисление, когда его параметры - близки к нулю, и предоставляет полезное ядро для стандартных показательных функций. Гиперболические функции, такие как cosh(x), могут быть эффективно и точно реализованы использованием FEXPM1; точность теряется в этой функции для малых значений x, если используется слово FEXP.

Важное приложение этого слова в финансах; говорите, что ссуда возмещена в 15 % в год; какова суточная дача? На компьютере с единичной точностью (6 десятичных цифр) точность:

1. Использование FLN и FEXP:
FLN от 1.15 = 0.139762,
деление на 365 = 3.82910E-4,
формирование экспоненты, используя FEXP = 1.00038, и
вычитание единицы (1), и преобразование в проценты = 0.038%.

Таким образом, мы имеем точность только две цифры.

2. Использование FLNP1 и FEXPM1:
FLNP1 от 0.15 = 0.139762, (это - то же самое значение как в первом
примере, хотя с параметром ближе к нулю оно не может быть таким)

деление на 365 = 3.82910E-4,
формирование экспоненты, и вычитание единицы (1) ,
используя FEXPM1 = 3.82983E-4,
и преобразование в проценты = 0.0382983%.

Это - точность полных 6 цифр.

Присутствие этого слова позволяет гиперболическим функциям быть вычисленными с пригодной для использования точностью. Например, гиперболический синус может быть определен как:

```
: FSINH  ( r1 -- r2 )
   FEXPM1  FDUP  FDUP 1.0E0 F+  F/  F+  2.0E0 F/ ;
```

A.12.6.2.1554 FLNP1
Эта функция позволяет точную компиляцию, когда ее параметры близки к нулю, и предоставляет полезное ядро для стандартных логарифмических функций. Например, FLN может быть реализована как:


```
: FLN   1.0E0 F-  FLNP1 ;
```

См.: A.12.6.2.1516 FEXPM1.

A.12.6.2.1616 FSINCOS

См.: A.12.6.2.1489 FATAN2.

A.12.6.2.1640 F~
Оно предоставляет три типа "равенства с плавающей точкой" в общем использовании -- "закрытое" в абсолютных терминах, точное равенство как представлено, и "относительно закрытое".

A.13 Дополнительный Locals набор слов

Технический Комитет имел проблему с locals. было убедительно аргументировано, что ANS Forth не должен ничего говорить о locals, так как:

- нет, ясной принятой практики в этой области;
- не все программисты Forth используют их или даже знают то, чем они являются;
- и немного реализаций используют один и тот же синтаксис, не говоря уже о тех же широких правилах использования и общих подходах.

Также было аргументировано, это будет казаться одинаково убедительно, что нехватка каких либо стандартных подходов к locals - это точно причина для нехватки принятой практики, так как locals являются очень нетривиальными для реализации переносимым и практичным способом. Было далее аргументировано, что пользователи, которые выбрали стать зависящими от locals, имеют тенденцию быть связанными с единственным продавцом и имеют мало стимулов для присоединения к группе, которая, на это надежда, "широко примет" ANS Forth, вместо Стандартного обращения к его проблемам.

Так как Технический Комитет был не способен достигнуть твердого согласия относительно того игнорировать locals или при принять синтаксис какого либо специфического продавца, он искал некоторый путь решения проблемы, которую был не способен просто отклонить. Понимание, что никакой отдельный механизм или синтаксис не может одновременно удовлетворить желания, выраженные во всех предложениях locals, которые были получены, упростило постановку задачи, чтобы определить механизм locals, так:

- является независимым от любого специфического синтаксиса;
- является расширяемым пользователем;
- допускает использование произвольных идентификаторов local в контексте отдельного определения;
- поддерживает фундаментальные типы данных размера ячейки Forth; и
- работает согласовано, особенно что касается реентерабельности и рекурсии.

Техническому Комитету кажется, что так много тех, кто активно используют locals, затрудняются в работе с ними, и это в настоящее время согласие Технического Комитета что, если ANS Forth имеет, что ни будь сказать относительно предмета, то это приемлемая вещь для того, чтобы это говорить.

Этот подход, определения (LOCAL), предложен как единственный, который может использоваться с небольшим количеством пользовательского кодирования для реализации некоторых, но не всех, схем locals в использовании. Следующие примеры кодирования иллюстрируют, как это может использоваться для реализации

двух синтаксисов.

- Синтаксис, определенный этим Стандартом и используемый в системах Creative Solutions, Inc.:

```
: LOCALS|  ( "name...name |" -- )
   BEGIN
      BL WORD   COUNT OVER C@
      [CHAR] | - OVER 1 - OR  WHILE
      (LOCAL)
   REPEAT 2DROP   0 0 (LOCAL)
;  IMMEDIATE

: EXAMPLE  ( n -- n**2 n**3 )
   LOCALS| N |   N  DUP N *  DUP N * ;
```

- Предложенный синтаксис: (LOCAL name) с дополнительными правилами использования:

```
: LOCAL  ( "name" -- )  BL WORD COUNT (LOCAL) ;  IMMEDIATE

: END-LOCALS  ( -- )  0 0 (LOCAL) ;  IMMEDIATE

: EXAMPLE  ( n -- n n**2 n**3 )
   LOCAL N  END-LOCALS   N  DUP N *  DUP N * ;
```

Другие синтаксисы могут быть реализованы, хотя некоторые, вероятно, будут требовать значительно большие усилия или в некоторых случаях преобразование программы. Все же другие подходы locals полностью несовместимы из-за больших различий в правилах использования и в некоторых случаях даже контекста идентификаторов. Например, законченная схема local использованная Johns Hopkins имела сложную семантику, которая не может быть повторена в терминах этой модели.

Для подкрепления намерений раздела 13, вот два примера фактического использования locals. Первый иллюстрирует правильное использование:

```
a) : {  ( "name ... }" - )
   BEGIN  BL WORD COUNT
      OVER C@ [CHAR] } -  OVER 1 -  OR WHILE
      (LOCAL)
   REPEAT 2DROP 0 0 (LOCAL)
;  IMMEDIATE

b) : JOE  ( a b c -- n )
   >R 2* R> 2DUP + 0
   { ANS 2B+C C 2B A }
   2 0 DO  1 ANS + I + TO ANS  ANS . CR  LOOP
   ANS . 2B+C . C . 2B . A . CR  ANS
;

c) 100 300 10 JOE .
```


Слово { в a) определяет синтаксис объявления local, который окружает список locals фигурными скобками. Оно не делает чего ни будь фантастического, типа переупорядочивания locals или предоставления начальных значений для некоторых из них, так locals инициализируются со стека в заданном по умолчанию порядке. Определение JOE в b) иллюстрирует использование этого синтаксиса. Заметьте, что

работа выполнена во время выполнения в этом определении прежде, чем объявлены locals. Хорошо использовать стек возвратов, пока что-либо помещенное туда, удалится до начала объявления.

Заметьте, что перед объявлением locals, удвоено B, вычислено подвыражение (2B+C), и предусмотрено начальное значение (нуль) для ANS. После того, как locals были объявлены, JOE продолжает использовать их. Заметьте, что к locals можно обращаться и модифицировать в пределах do-loops. Эффект интерпретации строки c) должен отобразить следующие значения:

```
1 (ANS первый раз через цикл),
3 (ANS второй раз),
3 (ANS), 610 (2B+C), 10 (C), 600 (2B), 100 (A), and
3 (ANS оставлено на стеке JOE).
```

Имена locals исчезают после того, как JOE было откомпилировано. Память и значение locals появляются, когда объявлены locals JOE и исчезают как JOE возвращается к его вызывающей программе в ; (точка с запятой).

Второй набор примеров иллюстрирует различные вещи, которые нарушают правила. Мы принимаем, что определения LOCAL и END-LOCALS выше присутствуют наряду с { из предшествующего примера.

```
d) : ZERO   0 POSTPONE LITERAL POSTPONE LOCAL ; IMMEDIATE

e) : MOE  ( a b )
     ZERO TEMP  LOCAL B  1+ LOCAL A+  ZERO ANSWER ;

f) : BOB  ( a b c d )  { D C }  { B A } ;
```

Вот - два определения с различными нарушениями правила 13.3.3.2a. В e) объявление из TEMP законно и создает local, чье начальное значение нулевое. Это - хорошо потому что выполняемый код, который генерирует ZERO, предшествует первому использованию (LOCAL) в определении. Однако, 1+ предшествующее объявлению A+ незаконно. Аналогично использование ZERO для определения ANSWER незаконно, потому что оно генерирует выполняемый код между использованиями (LOCAL). Наконец, MOE заканчивается незаконно (не END-LOCALS). BOB в f) нарушает правило против объявления двух наборов locals.

```
g) : ANN  ( a b  -- b )  DUP >R  DUP IF { B A } THEN  R> ;

h) : JANE  ( a b -- n )  { B A }  A B + >R  A B -  R> / ;
```

ANN в g) нарушает два правила. IF ... THEN вокруг объявления ее locals нарушает 13.3.3.2b, и копия B, оставленная на стеке возвратов перед объявлением locals, нарушает 13.3.3.2c. JANE в h) нарушает 13.3.3.2d, обращаясь к locals после размещения суммы A и B на стеке возвратов без первого удаления этой суммы.

```
i) : CHRIS  ( a b)
   { B A }  ['] A EXECUTE  5 ['] B >BODY !  [ ' A ] LITERAL LEE ;
```


CHRIS в i) иллюстрирует три нарушения 13.3.3.2e. Попытка EXECUTE local вызванное A несовместимо с некоторыми реализациями. Запись в B через >BODY вероятно вызовет трагический результат во многих реализациях; кроме того, если locals находятся в регистрах, они не могут быть адресованы как память независимо оттого, что написано.

Третье нарушение, в котором идентификатор исполнения для определения local передается как параметр к слову LEE, если позволено, имел бы неприятное следствие, что LEE может EXECUTE лексему и получить значение для A от специфического выполнения CHRIS , которое вызывает на сей раз LEE.

13.3 Дополнительные условия применения

Правило 13.3.3.2d может быть ослаблено без воздействия на целостность остальной части этой структуры. 13.3.3.2c не может.

13.3.3.2b запрещает использование стека данных для хранения local, потому что нет четко определенных правил использования для пользователей программистов в таком случае. Конечно, если стек данных как-то используется таким способом, что нет никаких правил использования, при которых locals невидимы для программиста, логически являются не на стеке, и соответствует реализации.

Минимальное требуемое число locals может (и должно) быть откорректировано, чтобы минимизировать стоимость соглашения для существующих пользователей locals.

Обращение к предварительно объявленным переменным local, запрещено Разделом 13.3.3.2d до того как любые данные, помещенные на стек возвратов приложением будут удалены, из-за возможного использования стека возвратов для хранения locals.

Разрешение для Стандартной Программы управлять стеком возвратов (например, через >R R>) в то время как активны переменные local, чрезмерно сдерживает возможности реализации. Соглашение пользователей locals состояло в том, что Local средства представляют эффективную функциональную замену для манипуляции стеком возвратов, и было разумно ограничение стандартного использования только для одного метода.

Обращение к locals в пределах, DO..LOOP явно разрешаются Разделом 13.3.3.2g, как дополнительное требование приспосабливания систем. Хотя слова, типа (LOCALS), написаны системным Разработчиком, может требоваться сокровенное знание внутренней структуры стека возвратов, такое знание не требуется пользователю совместимых систем Forth.

A.13.6 Словарь

A.13.6.1.2295   TO
Типичное использование:
        x TO name

См.: A.6.2.2295 TO.

A.13.6.2.1795   LOCALS|
Возможная реализация этого слова и примера использования дана в A.13, выше. Это предназначено только как пример; приемлема любая реализация, соглашающаяся с описанной семантикой.

A.14 дополнительный набор слов Распределения памяти

Набор слов Распределения памяти предоставляет средства для запроса памяти отличной от непрерывной области данных распределенной ALLOT. Во многих средах операционных систем неуместно для процесса пред-распределять большие количества непрерывной памяти (как было бы необходимо для использования ALLOT). Набор слов Распределения памяти может запрашивать память из системы в любое время, без знания заранее адреса памяти, где она будет получена.

A.15 Дополнительный набор слов утилит

Эти слова были в широко распространенном общем использовании, начиная с самых ранних систем Forth.

Хотя есть зависимости от окружений, присущие программам использующим ассемблер, фактически все системы Forth предоставляют такую возможность. Поскольку так много программ Forth предназначенных для приложений реального времени и внутренне непереносимых по этой причине, технический Комитет верит, что предоставление стандартного окна в ассемблеры - это полезное содействие программистам Forth.

Точно так же помощники программирования DUMP, и т.д., являются ценными инструментальными средствами даже притом, что их специфические форматы будут колебаться в зависимости от реализаций Forth и CPU. Эти слова - прежде всего, предназначены для использования программистом, и редко вызываются в программах.

Одна из первоначальных целей Forth состояла в том, чтобы стереть границу между "пользователем" и "программистом" - чтобы дать всю возможную мощь любому, кто имел случай использовать компьютер. Ничто из выше отмеченного или высказанного не должно быть истолковано, так что эта цель была оставлена.

A.15.6 Словарь

A.15.6.1.0220 .S
.S - удобство отладки, найденное почти на всех системах Forth. Оно - повсюду упомянуто в текстах Forth.

A.15.6.1.2194 SEE
SEE действует онлайновая форма документации слов, позволяя модификацию слов декомпиляцию и восстановление с соответствующими изменениями.

A.15.6.1.2465 WORDS
WORDS - удобство отладки, найденное почти на всех системах Forth. Оно - повсюду упомянуто в текстах Forth.

A.15.6.2.0470 ;CODE
Типичное использование:

```
: namex ... <create> ... ;CODE ...
```

Где namex - определяющее слово, и <create> - это CREATE или любое определяемое пользователем слово, которое вызывает CREATE.

A.15.6.2.0930 CODE
Некоторые системы Forth реализуют ассемблерную функцию, добавляя список слов ASSEMBLER к порядку поиска, используя текстовый интерпретатор для выделения постфиксного ассемблера с лексическими характеристиками, подобными исходному тексту Forth. Как правило, в таких системах, ассемблирование заканчивается, когда интерпретируется слово END-CODE.

A.15.6.2.1015 CS-PICK
Намерение состоит в том, чтобы повторить dest на стеке потока управления так, чтобы оно могло быть разрешено больше чем однажды. Например:

```
\ Условная передача управление на начало цикла
\ Это подобно духу "C" инструкции "continue".

: ?REPEAT  ( dest -- dest ) \ Компиляция
           ( flag -- )      \ Выполнение
   0 CS-PICK   POSTPONE UNTIL
; IMMEDIATE

: XX  ( -- ) \ Пример использования ?REPEAT
   BEGIN
     ...
   flag ?REPEAT  ( Возврат к BEGIN если flag - false )
```

```
          ...
          flag ?REPEAT  ( Возврат к BEGIN если flag - false )
            ...
          flag UNTIL    ( Возврат к BEGIN если flag - false )
          ...
       ;
```

A.15.6.2.1020 CS-ROLL
Намерение состоит в том, чтобы изменить порядок, в котором origs и dests на стеке потока управления должны быть разрешены последующими словами потока управления. Например, WHILE могло быть реализовано, в терминах IF и CS-ROLL, следующим образом:

```
       : WHILE  ( dest -- orig dest )
          POSTPONE IF  1 CS-ROLL
       ; IMMEDIATE
```


A.15.6.2.1580 FORGET
Типичное использование:

```
       ... FORGET name ...
```

FORGET предполагает, что вся информация необходимая для восстановления словаря к его предыдущему состоянию выводима как-нибудь из забываемого слова. В то время как это может быть верно в простой линейной модели словаря, это трудно реализовать в других системах Forth; например в системах с множественными адресными пространствами. Например, если Forth внедрен в ROM, как FORGET узнать, сколько RAM восстановится, когда массив забыт? Общее и привилегированное решение предусмотрено MARKER.

A.15.6.2.2531 [ELSE]
Типичное использование:

```
       ... flag [IF] ... [ELSE] ... [THEN] ...
```

A.15.6.2.2532 [IF]
Типичное использование:

```
       ... flag [IF] ... [ELSE] ... [THEN] ...
```

A.15.6.2.2533 [THEN]
Типичное использование:

```
       ... flag [IF] ... [ELSE] ... [THEN] ...
```

Программное обеспечение, которое выполняется на нескольких системных окружениях часто, содержит некоторый исходный текст, который зависит от окружения. Условная компиляция - это выборочное включение или исключение частей исходного текста во время компиляции - является одной методикой, которая часто используется для помощи в обслуживании такого исходного текста.

Условная компиляция иногда делается с "разумными комментариями" - определениями, которые пропускают или не пропускают остаток строки, основываясь на некоторой проверке. Например:

```
       \ Если 16-Bit? содержит TRUE, строка с предшествующим 16BIT\
       \ будет пропущена. Иначе, она не будет пропущена.

       VARIABLE 16-BIT?

       : 16BIT\  ( -- )  16-BIT? @  IF  POSTPONE \  THEN
       ;  IMMEDIATE
```

Эта методика работает на строке строчного базиса, и хороша для коротких изолированных разновидностей кодовых последовательностей.

Более сложные проблемы условной компиляции предлагают метод вложенности, который может охватить больше одной исходной строки одновременно. Слова, включенные в ANS Forth расширение дополнительного набора слов утилит полезны для этой цели. Реализация, данная ниже, работает с любым входным источником (клавиатура, EVALUATE, BLOCK, или текстовый файл).


```
: [ELSE]  ( -- )
   1 BEGIN                                \ level
     BEGIN  BL WORD COUNT  DUP  WHILE     \ level adr len
       2DUP  S" [IF]"  COMPARE 0= IF      \ level adr len
         2DROP 1+                         \ level'
       ELSE                               \ level adr len
         2DUP  S" [ELSE]"  COMPARE 0= IF  \ level adr len
            2DROP 1- DUP IF 1+ THEN       \ level'
         ELSE                             \ level adr len
           S" [THEN]"  COMPARE 0= IF      \ level
             1-                           \ level'
           THEN
         THEN
       THEN ?DUP 0=  IF EXIT THEN         \ level'
     REPEAT  2DROP                        \ level
   REFILL 0= UNTIL                        \ level
   DROP
;  IMMEDIATE

: [IF]  ( flag -- )
   0= IF POSTPONE [ELSE] THEN
;  IMMEDIATE

: [THEN]  ( -- )  ;  IMMEDIATE
```

A.16 Дополнительный набор слов порядка поиска

Спецификация Порядка поиска и механизмы управления широко различаются. FIG-Forth, Forth-79, polyFORTH, и Forth-83 словарь и механизмы порядка поиска - все взаимно несовместимы. Законченный список несовместимых механизмов, используемых или предлагаемых, много длиннее. ALSO/ONLY схема, описанная в Forth-83 Экспериментальных Предложениях, имеет прочную поддержку семейства. Однако многие полагают, что она была фундаментально порочна, и энергично против этого выступают.

Признавая это разнообразие этот Стандарт определяет новый "примитивный" набор утилит, из которых могут быть созданы различные схемы. Этот примитивный набор слов порядка поиска предназначен, чтобы быть переносимым "набором конструкций" от которого могут быть сформированы слова порядка поиска скорее, чем интерфейс пользователя. ALSO/ONLY или другие схемы "словаря", поддерживаемые главными продавцами Forth могут быть определены в терминах примитивного набора слов порядка поиска.

Кодирование для идентификаторов списка слов wid могло бы быть маленьким-целым индексом в массиве записей определений списка слов, адрес области данных такой записи, смещение пользовательской области, идентификатор исполнения Forth-83 стиля непонятного словаря, адрес поля связи первого определения в списке слов, или что-нибудь еще. Это - полностью к системному разработчику.

В некоторых системах интерпретация числовых литералов управляется включением "псевдо списка слов" которые распознают числа в конце порядка поиска. Эта методика обеспечивается поведением "порядка поиска по умолчанию" SET-ORDER когда дан параметр -1. В системе, использующей традиционную реализацию ALSO/ONLY, минимальный порядок поиска был бы эквивалентен слову ONLY.

Никогда не было переносимого способа восстановить сохраненный порядок поиска. F83 (не Forth 83) представил слово PREVIOUS, которое почти сделало возможным "разгрузить" порядок поиска неоднократно выполняя фразу CONTEXT @ PREVIOUS. Порядок поиска мог быть "перезагруженным", повторением ALSO CONTEXT !. К сожалению, не было никакого переносимого пути для определения, сколько списков слов были в порядке поиска.

ANS Forth удалил слово CONTEXT, потому что во многих системах его содержание относится к более чем одному списку слов, составляя проблемы переносимости.

Заметьте, что : (двоеточие) больше не затрагивает порядок поиска. Предыдущее поведение, где список слов компиляции заменяет первый список слов порядка поиска, может быть эмулировано следующим переопределением : (двоеточие).

```
: :  GET-ORDER SWAP DROP  GET-CURRENT  SWAP SET-ORDER  : ;
```

A.16.2 Дополнительные термины

Порядок поиска
Заметьте, что использование термина "список" не обязательно подразумевает реализацию как связанного списка.

A.16.3.3 Поиск определений имен

Другими словами не гарантируется, что следующее будет работать:

```
: FOO  ... [ ... SET-CURRENT ] ... RECURSE ...
;  IMMEDIATE
```

RECURSE , ; (точка с запятой), и IMMEDIATE могут или не могут нуждаться в информации, сохраненной в списке слов компиляции.

A.1'.' Словарь

A.16.6.1.2192 SEARCH-WORDLIST

Строковый параметр для SEARCH-WORDLIST представленный c-addr u, скорее чем только c-addr как в FIND. Комитет желает установить c-addr u как привилегированное представление строки на стеке, и приняло это представление для всех новых функций, которые принимают строковые параметры. В то время как это решение может вызывать реализацию SEARCH-WORDLIST несколько более трудную в существующих системах, комитет чувствует, что дополнительная трудность незначительна.

Когда SEARCH-WORDLIST не в состоянии найти слово, оно не возвращает строку, как делает FIND. Это - в соответствии с общим принципом, что слова Forth потребляют их параметры.

A.16.6.2.0715 ALSO
Вот - реализация ALSO/ONLY в терминах примитивного набора слов порядка поиска.


```
WORDLIST CONSTANT ROOT   ROOT SET-CURRENT

: DO-VOCABULARY  ( -- ) \ Фактор реализации
  DOES>  @ >R           (  ) ( R: widnew )
   GET-ORDER  SWAP DROP ( wid1 ... widn-1 n )
   R> SWAP SET-ORDER
```

```
;

: DISCARD  ( x1 .. xu u - ) \ Фактор реализации
   0 ?DO DROP LOOP          \ DROP u+1 элементов стека
;

CREATE FORTH  FORTH-WORDLIST , DO-VOCABULARY

: VOCABULARY  ( name -- )  WORDLIST CREATE ,  DO-VOCABULARY ;
: ALSO  ( -- )  GET-ORDER  OVER SWAP 1+ SET-ORDER ;
: PREVIOUS  ( --  )  GET-ORDER  SWAP DROP 1- SET-ORDER ;
: DEFINITIONS  ( -- )  GET-ORDER  OVER SET-CURRENT DISCARD ;
: ONLY ( -- )  ROOT ROOT  2 SET-ORDER ;

\ Forth-83 версия; только удаляет ONLY
: SEAL  ( -- )  GET-ORDER 1- SET-ORDER DROP ;

\ F83 and F-PC версия; оставляет только CONTEXT
: SEAL  ( -- )  GET-ORDER OVER 1 SET-ORDER DISCARD ;
```

Предшествующее определение ONLY в терминах "ROOT" списка слов следует использованию в F83, и предполагает, что заданный по умолчанию порядок поиска только включает ROOT и FORTH. Более переносимое определение FORTH и ONLY, без предположений:

```
<опускает  ... WORDLIST CONSTANT ROOT ...  строку>

CREATE FORTH GET-ORDER OVER , DISCARD DO-VOCABULARY

: ONLY  ( -- )  -1 SET-ORDER ;
```

Вот - простая реализация GET-ORDER и SET-ORDER, включая соответствующее определение FIND. Реализации WORDLIST, SEARCH-WORDLIST, GET-CURRENT и SET-CURRENT зависят от системных подробностей и не даны здесь.

```
16 CONSTANT #VOCS
VARIABLE #ORDER
CREATE CONTEXT  #VOCS CELLS ALLOT

: GET-ORDER  ( -- wid1 .. widn n )
   #ORDER @ 0 ?DO
     #ORDER @  I - 1- CELLS CONTEXT + @
   LOOP
   #ORDER @
;
```



```
: SET-ORDER  ( wid1 .. widn n -- )
   DUP -1 = IF
     DROP  <помещает системный список слов по умолчанию и n>
   THEN
   DUP #ORDER !
   0 ?DO  I CELLS CONTEXT + ! LOOP
;

: FIND  ( c-addr -- c-addr 0 | w 1 | w -1 )
   0                     ( c-addr 0 )
   #ORDER @ 0 ?DO
     OVER COUNT          ( c-addr 0 c-addr' u )
     I CELLS CONTEXT + @ ( c-addr 0 c-addr' u wid)
     SEARCH-WORDLIST     ( c-addr 0; 0 | w 1 | w -1 )
     ?DUP IF             ( c-addr 0; w 1 | w -1 )
```

```
            2SWAP 2DROP LEAVE ( w 1 | w -1 )
          THEN                ( c-addr 0 )
        LOOP                  ( c-addr 0 | w 1 | w -1 )
    ;
```

В реализации, где механизм поиска в словаре использует хеш-таблицу или кэш поиска для уменьшения времени поиска, SET-ORDER мог бы быть должен восстановить хеш-таблицу или сбросить на диск кэш.

A.17 Дополнительный строковый набор слов

A.17.6 Словарь

A.17.6.1.0245 /STRING
/STRING используется для удаления или добавления символов относительно "левого" конца символьной строки. Положительные значения n исключат символы со строки, в то время как отрицательные значения n будут включать символы слева от строки. /STRING - естественный агент WORD и обычно доступный.

A.17.6.1.0910 CMOVE
Если c-addr2 находится в пределах исходной области, (то есть, когда c-addr2 - не меньше чем c-addr1 и c-addr2 - меньше чем количество c-addr1 u CHARS +), происходит размножение памяти.

Типичное использование:
Предположите символьную строку по адресу 100: "ABCD".
Тогда после

```
    100 DUP  CHAR+  3 CMOVE строка по адресу 100 - "AAAA".
```

Объяснение для CMOVE и CMOVE> вытекает из MOVE.

A.17.6.1.0920 CMOVE>
Если c-addr1 находится в пределах области назначения, (то есть, когда c-addr1 больше чем или равно c-addr2 и c-addr2 - меньше чем количество c-addr1 u CHARS +), происходит размножение памяти.

Типичное использование:
Предположите символьную строку по адресу 100: "ABCD".
Тогда после

```
    100 DUP CHAR+ SWAP 3 CMOVE>, строка по адресу 100 - "DDDD".
```

A.17.6.1.0935 COMPARE
Существующие системы Forth исполняют строковые операции сравнения, использующие слова, которые отличаются проверкой правописания, параметрами ввода и вывода, и чувствительностью к регистру. Была выбрана одна широко используемая.

A.17.6.1.2191 SEARCH
Существующие системы Forth исполняют операции поиска строки, использующие слова, которые отличаются проверкой правописания, параметрами ввода и вывода, и чувствительностью к регистру. Была выбрана одна широко используемая.

A.17.6.1.2212 SLITERAL
Текущие функциональные возможности 6.1.2165 S" могут быть предоставлены следующим определением:

```
    : S" ( "ccc<quote>" -- )
       [CHAR] " PARSE   POSTPONE SLITERAL
```

```
        ; IMMEDIATE
```

B. Библиография (информационное приложение)

Промышленные стандарты

Forth-77 Стандарт, Forth Users Group, FST-780314.
Forth-78 Стандарт, Forth International Standards Team.
Forth-79 Стандарт, Forth Standards Team.
Forth-83 Стандарт и Приложения, Forth Standards Team.

Стандарты, упомянутые в этом разделе, были разработаны Forth Standards Team, группой добровольцев, которая включила и разработчиков и пользователей. Это была организация добровольцев, работающая под ее собственным уставом и без любых формальных связей с ANSI, IEEE или любого подобного органа стандартов. Несколько членов Forth Standards Team также были членами X3J14 Технического Комитета.

Книги

Brodie, L. Starting FORTH (2nd ed). Englewood Cliffs, NJ: Prentice Hall, 1987.

Brodie, L. Thinking FORTH. Englewood Cliffs, NJ: Prentice Hall, 1984.

Feierbach, G. and Thomas, P. Forth Tools & Applications. Reston, VA: Reston Computer Books, 1985.

Haydon, Dr. Glen B. All About FORTH, Third Edition. La Honda, CA: 1990.

Kelly, Mahlon G. and Spies, N. FORTH: A Text and Reference. Englewood Cliffs, NJ: Prentice Hall, 1986.

Knecht, K. Introduction to Forth. Indiana: Howard Sams & Co., 1982.

Koopman, P. Stack Computers, The New Wave. Chichester, West Sussex, England: Ellis Horwood Ltd. 1989

Martin, Thea, editor. A Bibliography of Forth References, Third Edition. Rochester, New York: Institute of Applied Forth Research, 1987.

McCabe, C. K. Forth Fundamentals (2 volumes). Oregon: Dilithium Press, 1983.

Pountain, R. Object Oriented Forth. London, England: Academic Press, 1987.

Ouverson, Marlin, editor. Dr. Dobbs Toolbook of Forth. Redwood City, CA: M&T Press, Vol. 1, 1986; Vol. 2, 1987.

Terry, J. D. Library of Forth Routines and Utilities. New York: Shadow Lawn Press, 1986

Tracy, M. and Anderson, A. Mastering FORTH (revised ed). New York: Brady Books, 1989.

Winfield, A. The Complete Forth. New York: Wiley Books, 1983. Journals, magazines and newsletters

Forsley, Lawrence P., Conference Chairman. Rochester Forth Conference Proceedings. Rochester, New York: Institute of Applied Forth Research, 1981 to present.

Forsley, Lawrence P., Editor-in-Chief. The Journal of Forth Application and Research.Rochester, New York: Institute of Applied Forth Research, 1983 to present.

Frenger, Paul, editor. SIGForth Newsletter. New York, NY: Association for Computing Machinery, 1989 to present.

Ouverson, Marlin, editor. Forth Dimensions. San Jose, CA: The Forth Interest Group, 1978 to present.

Reiling, Robert, editor. FORML Conference Proceedings. San Jose, CA: The Forth Interest Group, 1980 to present.

Ting, Dr. C. H., editor. More on Forth Engines. San Mateo, CA: Offete Enterprises, 1986 to present.

Избранные статьи

Hayes, J.R. "Postpone" Proceedings of the 1989 Rochester Forth Conference. Rochester, New York: Institute for Applied Forth Research, 1989.

Kelly, Guy M. "Forth." McGraw-Hill Personal Computer Programming Encyclopedia – Languages and Operation Systems. New York: McGraw-Hill, 1985.

Kogge, P. M. "An Architectural Trail to Threaded Code Systems." IEEE Computer (March, 1982).

Moore, C. H. "The Evolution of FORTH - An Unusual Language." Byte (August 1980).

Rather, E. D. "Forth Programming Language." Encyclopedia of Physical Science & Technology (Vol. 5). New York: Academic Press, 1987.

Rather, E. D. "FORTH." Computer Programming Management. Auerbach Publishers, Inc., 1985.

Rather, E. D.; Colburn, D. R.; Moore, C. H. "The Evolution of FORTH." ACM SIGPLAN Notices. (Vol. 28, No. 3, March 1993).

C. Перспектива (информационное приложение)

Цель этого раздела состоит в том, чтобы предоставить краткий неформальный обзор Forth как языка, иллюстрируя его хронологию, наиболее видные возможности, использование, и общие методы реализаций. Ничто в этом разделе не должно рассматриваться как обязывающее разработчиков или пользователей. Список книг и статей - данный в Приложении B для заинтересованных в больших познаниях о Forth.

C.1 Возможности Forth

Forth предоставляет интерактивную среду программирования. Его первичные использования были - научные и индустриальные приложения типа оснащения аппаратурой, робототехника, управление процессом, графика и обработка изображений, искусственный интеллект и деловые приложения. Основные преимущества Forth включают быструю, интерактивную программную разработку и эффективное использование компьютерных аппаратных средств.

О Forth часто говорят как языке, потому что это его наиболее видимый аспект. Но фактически, Forth это и больше и меньше чем обычный язык программирования: больше в том, что все свойства, обычно связываемые с большим портфелем отдельных программ (компиляторы, редакторы, и т.д.) включены в пределах его диапазона, и меньше, в чем он испытывает недостаток (преднамеренно), сложная характеристика синтаксиса большинства языков высокого уровня.

Первоначальные реализации Forth были автономными системами, которые включали функции, обычно выполняемые отдельными операционными системами, редакторы, компиляторы, ассемблеры, отладчики и другие утилиты. Единственный простой, непротиворечивый набор правил управлял этим полным диапазоном свойств. Сегодня, хотя очень быстрые автономные версии все еще продаются для многих процессоров, есть также много версий, которые работают одновременно с обычными операционными системами типа MS DOS и UNIX.

Forth не получен из какого либо другого языка. В результате, его вид и внутренние характеристики могут казаться незнакомыми новым пользователям. Но простота Forth, высочайшая модульность, и интерактивная природа смещали начальную странность, делая его легким в изучении и в использовании. Новый программист Forth должен вложить некоторое время, осваивая его большой командный набор. После приблизительно месяца полных рабочих дней использования Forth, программист мог помнить большее количество его внутренней работы, чем возможно с обычными операционными системами и компиляторами.

Наиболее нетрадиционная функция Forth - его расширяемость. Процесс программирования в Forth состоит из определения новых "слов" - фактически новых команд на языке. Они могут быть определены в терминах предварительно определенных слов, так как почти каждый обучает детей понятиям, объясняя их в терминах предварительно усвоенных понятий. Такие слова названы "высокоуровневыми определениями". Альтернативно, новые слова могут также быть определены в ассемблерном коде, поскольку большинство реализаций Forth включает ассемблер для базового процессора.

Эта расширяемость облегчает разработку языков специального применения для специфических прикладных областей или дисциплин.

Расширяемость Forth выходит за пределы только добавления новых команд к языку. С эквивалентной непринужденностью, можно также добавлять новые виды слов. То есть можно создать слово, которое непосредственно определит слова. Создавая такое определяющее слово, программист может определить специализированное поведение для слов, которые оно создаст, и которые будут эффективны во время компиляции, во время выполнения, или в обоих случаях.

Эта возможность позволяет определять специализированные типы данных, с полным контролем над структурой и поведением. Поскольку поведение времени-выполнения таких слов может быть определено высокоуровневым или в коде, слова созданные этим новым определяющим словом эквивалентны в работе всем другим видам слов Forth. Кроме того, также просто добавить новые директивы компилятора для реализации специальных видов циклов или других управляющих структур.

Большинство профессиональных реализаций Forth написано в Forth. Много систем Forth включают "транслятор метаязыка", который позволяет пользователю изменять непосредственно внутреннюю структуру системы Forth.

C.2 Хронология Forth

Forth был изобретен Charles H. Moore. Непосредственный результат работы Moore в 1960-е, первая программа, которая названа Forth была написана около 1970. Первая законченная реализация использовалась в 1971 в National Radio Astronomy Observatory на 11-метровом радио телескопе в Аризоне. Эта система была ответственна за наведение и слежение телескопа, сбор данных и запись их на

магнитной ленте, и поддержку интерактивного графического терминала, на котором астроном мог анализировать предварительно зарегистрированные данные. Многозадачная природа системы позволила всем этим функциям быть выполненными одновременно, без разрешения конфликтов или другого вмешательства - очень продвинутая концепция для того времени.

Система была настолько полезна, что астрономы со всех континентов начали просить о копиях. Его использование быстро распространилось, и в 1976 Forth был принят как стандартный язык International Astronomical Union.

В 1973, Moore и коллеги сформировали Forth, Inc для изучения коммерческого использования языка. Forth, Inc разработала многопользовательские версии Forth на миникомпьютерах для разнообразных проектов, в пределах от баз данных до научных приложений типа обработки изображений. В 1977, Forth, Inc разработала версию для недавно представленных 8-бит микропроцессоров названную "microFORTH", которая успешно использовалась во встроенных в микропроцессор приложениях в Соединенных Штатах, Англии и Японии.

Стимулированная объемом сбыта microFORTH, группа людей с компьютерным хобби в Северной Калифорнии заинтересовалась Forth, и в 1978 сформировала Forth Interest Group (FIG). Они разработали упрощенную модель, которую они реализовали на нескольких микропроцессорах и издали распечатки и диски очень низкой стоимости. Интерес к Forth распространился быстро, и сегодня есть отделения Forth Interest Group по всей США и в более чем пятнадцати странах.

К 1980, множество новых продавцов Forth пришло на рынок с версиями Forth основанными на модели FIG. Прежде всего, предназначенные для персональных компьютеров, эти относительно недорогие системы Forth были распространены очень широко.

C.3 Аппаратные реализации Forth

Внутренняя архитектура Forth моделирует компьютер с двумя стеками, набором регистров, и другими стандартизированными возможностями. В результате, это было почти неизбежно, что кто-то пытался бы построить аппаратное представление фактического компьютера Forth.

В начале 1980-ых, Rockwell произвела 6502-вариант с примитивами Forth во встроенной ROM, Rockwell 65F11. Этот чип использовался успешно во многих внедренных микропроцессорных приложениях. В середине 1980х Zilog разработала z8800 (Super8) которая предложила ENTER (nest), EXIT (unnest) и NEXT в микропрограмме.

В 1981, Moore взялся проектировать реализацию на уровне кристалла виртуальной Forth машины. Работая сначала в Forth, Inc и впоследствии с компанией запуска NOVIX, сформированной для разработки чипа, Moore завершил конструкцию в 1984, и первые прототипы были произведены в начале 1985. Более ранние Forth процессоры были разработаны Harris Semiconductor Corp., Johns Hopkins University, и другими.

C.4 Попытки стандартизации

Первым главным усилием по стандартизации Forth была встреча в Утрехте в 1977. Посетители произвели предварительный стандарт, и согласились встретиться на следующий год. 1978 встречу также посетили члены недавно сформированной Forth Interest Group. В 1979 и 1980 ряд встреч, посещаемых и пользователями и продавцами, произвели более всесторонний стандарт названный Forth 79.

Хотя Forth 79 был очень влиятелен, много пользователей Forth и продавцов нашли в нем серьезные недостатки, и в 1983 был выпущен новый стандарт названный Forth 83.

Поощренная широко распространенным принятием Forth 83, группа пользователей и продавцов встретилась в 1986, чтобы исследовать выполнимость American National Standard. X3J14 Технический комитет по ANS Forth проводил ее первую встречу в 1987. Этот Стандарт - это результат.

C.5 Программирование в Forth

Forth - подобный английскому язык, чьи элементы (называемые "слова") - это именованные элементы данных, процедуры, и определяющие слова, способные создавать элементы данных с настраиваемыми характеристиками. Процедуры и определяющие слова могут быть определены в терминах предварительно определенных слов или в машинном коде, используя встроенный ассемблер.

Forth "слова" функционально аналогичны подпрограммам на других языках. Они - также эквивалентны командам на других языках - Forth размывает различия между лингвистическими элементами и функциональными элементами.

Слова посылаются по имени или с клавиатуры или из источника программы. В результате, термин "слово" применяется и к программному (и лингвистическому) элементу и к их текстовым именам. В синтаксическом анализе текста Forth полагает, что слово будет любой строкой символов, ограниченных пробелами. Есть несколько специальных символов, которые не могут быть включены в слово или начало слова: пробел (универсальный разделитель), CR (который заканчивает ввод с терминала), и возврат или DEL (для возврата в течение ввода с клавиатуры). Много групп принимают соглашения об именах, чтобы улучшить удобочитаемость. Слова, встречающиеся в тексте, относятся к трем категориям: определенные слова (то есть, процедуры Forth), числа, и неопределенные слова. Например, вот - четыре слова:

```
HERE      DOES>      !      8493
```


Первые три - стандартно-определенные слова. Это означает, что они имеют входы в Forth словарь, описанный ниже, объясняя, что Forth должен делать, когда эти слова встретятся. Число "8493" возможно не будет найдено в словаре, и Forth преобразует его в двоичный код и поместит его в его магазинный стек для параметров. Когда Forth сталкивается с неопределенным словом и не может преобразовывать его к числу, слово возвращается пользователю с сообщением исключения.

Архитектурно, слова Forth строго придерживаются принципов "структурного программирования":

- Слова должны быть определены прежде их использования.
- Логический поток ограничен последовательными, условными, и итерационными конструкциями. Включены слова для реализации наиболее полезных управляющих структур программы.
- Программист работает со многими маленькими, независимыми модулями (словами) для максимальной контролируемости и надежности.

Forth характеризуется пятью главными элементами: словарь, два магазинных стека, интерпретаторы, ассемблер, и виртуальная память. Хотя каждый из них может быть найден в других системах, комбинация производит совместные усилия, которые создают мощную и гибкую систему.

C.5.1 словарь Forth

Программа Forth организована в словарь, который занимает большую часть памяти, используемой системой. Этот словарь - связанный список элементов переменной длины, каждый из которых определяет слово. Содержание каждого определения зависит от типа слова (элемент данных, константа, последовательность операций,

и т.д.). Словарь расширяем, обычно растет в область верхней памяти. На некоторых многопользовательских системах индивидуальные пользователи имеют частные словари, каждый из которых связан с общедоступным системным словарем.

Слова добавляются в словарь "определяющими словами", более употребительное из которых - это : (двоеточие). Когда : выполняется, оно создает определение для слова, которое следует за ним. В классических реализациях содержимое этого определения - строка адресов предварительно определенных слов, которые будут выполнены по очереди всякий раз, когда вызывается определяемое слово. Определение заканчивается ; (точка с запятой). Например, вот - определение:

```
: RECEIVE  ( -- addr n )  PAD DUP 32 ACCEPT ;
```

Имя нового слова - RECEIVE. Комментарий (в круглых скобках) указывает, что оно не требует никаких параметров и возвратит адрес и счетчик на стеке данных. Когда RECEIVE выполняется, оно последовательно исполнит слова в остатке определения. Слово PAD помещает на стек адрес временной памяти используемой для обработки строки. DUP дублирует верхний элемент стека, так что мы теперь имеем две копии адреса. Число 32 также помещается на стек. Слово ACCEPT берет адрес (предоставленный PAD) и длину (32) на стеке, принимает с клавиатуры строку до 32 символов, которые будут помещены по определенному адресу, и возвращает число полученных символов. Копия адреса временной памяти остается на стеке ниже счетчика так чтобы подпрограмма, которая вызвала RECEIVE, могла использовать его, чтобы подобрать полученную строку.

C.5.2 Магазинные стеки

Пример выше иллюстрирует использование магазинных стеков для передачи параметров между словами Forth. Forth поддерживает два магазинных стека, или списков LIFO. Они предоставляют связь между словами Forth плюс эффективный механизм для управления логическим потоком. Стек содержит 16-бит элементы на 8-бит и 16-бит компьютерах, и 32-бит элементы на 32-бит процессорах. Числа две-ячейки занимают две позиции стека, с самой старшей частью на вершине. Элементы на стеке могут быть адреса или элементы данных различных видов. Стеки имеют неопределенный размер, и обычно растут к меньшим адресам памяти.

Хотя структура обоих стеков одинаковая, они имеют очень различное использование. Пользователь взаимодействует наиболее непосредственно со Стеком Данных, который содержит параметры, передаваемые между словами. Эта функция заменяет вызывающие последовательности, используемые обычными языками. Это внутренне эффективно, и делает подпрограммы в действительности реентерабельными. Второй стек назван Стеком Возвратов, поскольку его основная функция это держать адреса возврата для вложенных определений, хотя иногда там временно сохраняются другие виды данных.

Использование Стека Данных (часто называемого просто "стек") ведет к нотации, в которой операнды предшествуют операторам. Слово ACCEPT в примере выше брало адрес и счетчик со стека и оставляло там другой адрес. Точно так же слово по имени BLANK ожидает адрес и счетчик, и поместит определенное число пробелов (20H) в области, начинающейся с этого адреса. Таким образом,

```
PAD 25 BLANK
```

заполнит рабочую область, чей адрес помещен на стек словом PAD, 25 пробелами. Прикладные слова обычно определяются, чтобы работать подобным образом. Например,

```
100 SAMPLES
```

могло быть определено, чтобы делать запись 100 измерений в массиве данных.

Арифметические операторы также ожидают значения и оставляют результат на стеке. Например, + складывает два верхних числа со стека, заменяя их оба их суммой. Так как результаты операций оставляются на стеке, операции могут быть связаны вместе без необходимости определения переменных для использования как временной памяти.

C.5.3 Интерпретаторы

Forth - традиционно интерпретирующая система, так как выполнение программы управляется скорее элементами данных, чем машинным кодом. Интерпретаторы могут быть медленны, но Forth поддерживает высокую скорость, требуемую для приложений реального времени при наличии двух уровней интерпретации.

Первый - текстовый интерпретатор, который выделяет строки с терминала или запоминающего устройства и ищет каждое слово в словаре. Когда слово найдено, оно выполняется вызовом второго уровня, адресного интерпретатора.

Второй - это "адресный интерпретатор". Хотя не все системы Forth реализованы таким образом, это была первая и - все еще основная технология реализации. В виду маленьких затрат на выполнение, адресный интерпретатор может выдавать очень компактную объектную программу, что было главным фактором в широком принятии Forth во внедренных системах и других приложениях, где желателен маленький объектный размер.

Адресный интерпретатор обрабатывает строки адресов или лексем, откомпилированных в определениях созданных : (двоеточие), выполняя определение указанное каждым. Содержание большинства определений - это последовательность адресов предварительно определенных слов, которые будут выполнены адресным интерпретатором в свою очередь. Таким образом, когда слово RECEIVE (определенное выше) выполняется, слово PAD, слово DUP, литерал 32, и слово ACCEPT будут последовательно выполнены. Процесс заканчивается точкой с запятой. Это выполнение не требует исследования словаря, синтаксического анализа, или другой логики, потому что, когда RECEIVE было откомпилировано, словарь просматривался для каждого слова, и его адрес (или другая лексема) были помещены в следующую последовательную ячейку входа. Текст не был сохранен в памяти, ни даже в сжатой форме.

Адресный интерпретатор имеет два важных свойства. Первое, это быстро. Хотя фактическая скорость зависит от специфической реализации, профессиональные реализации высоко оптимизированы, часто требуя только одну или две машинных команды на адрес. На большинстве эталонных тестов, хорошая реализация Forth существенно превосходит по быстродействию интерпретирующие языки типа BASIC или LISP, и сравнивается благоприятно с другими компилирующими языками высокого уровня.

Второе, адресный интерпретатор делает определения Forth чрезвычайно компактными, так как каждая ссылка требует только одной ячейки. В сравнении, вызов подпрограммы, созданной большинством компиляторов, включает команды для обработки вызывающей последовательности (ненужной в Forth из-за стека) перед и после команды CALL или JSR и адреса.

Большинство слов в словаре Forth будет определено : (двоеточие) и интерпретировано адресным интерпретатором. Большинство Forth сами определены таким образом.

C.5.4 Ассемблер

Большинство реализаций Forth включают макроассемблер для CPU, на котором они работают. Используя определяющее слово CODE, программист может создавать определение, чье поведение будет состоять из выполнения фактических машинных команд. Определения CODE могут использоваться для выполнения ввода-вывода,

реализации арифметических примитивов, и выполнения другой машинно-зависимой или критической по времени обработки. При использовании CODE программист имеет полный контроль над CPU, как с любым другим ассемблером, и определения CODE, выполняются с полной машинной скоростью.

Это - важное свойство Forth. Оно разрешает явный машинно-зависимый код в управляемых частях со специфическими соглашениями связи, которые являются машинно-независимыми. Для перемещения приложения на другой процессор требуется перекодирование только CODE слов, которые взаимодействуют с другими словами Forth точно одним и тем же способом.

Ассемблеры Forth настолько компактны (типично несколько Килобайт) что они могут быть резидентными в системе (как - компилятор, редактор, и другие утилиты). Это означает, что программист может вводить короткие определения CODE и выполнять их немедленно. Это возможность особенно ценна в испытании заказных аппаратных средств.

C.5.5 Виртуальная память

Заключительный уникальный элемент Forth - это его способ использования диска или другого запоминающего устройства как разновидности "виртуальной памяти" для данных и источника программы. Как в случае адресного интерпретатора, этот подход - историческая характеристика Forth, но ни в коем случае не универсален.

Диск разделен на 1024-байтовые блоки. В памяти предусмотрено два или более буфера, в которые блоки читаются автоматически, когда упоминаются. Каждый блок имеет установленный номер блока, который в чистых системах является прямой функцией его физического местоположения. Если блок в памяти изменен, он будет автоматически перезаписан, когда ее буфер должен будет повторно использован. Явное чтение и запись не необходимы; программа найдет данные в памяти всякий раз, когда она обращается к ним.

Блочно-ориентированная обработка диска эффективна и проста в реализации для чистой системы Forth. В результате, блоки предоставляют полностью мобильный механизм для обработки источника программы и данных как чистых, так и версий Forth одновременно находящихся в памяти с различными базовыми операционными системами.

Определения в блоках источника программы компилируются в память словом LOAD. Большинство реализаций включают редактор, который форматирует блок для отображения в 16 строк по 64 символа в каждой, и предусматривают команды, изменяющие источник. Пример источника Forth блока - дан на рис. C.1 ниже.

Исходные блоки исторически были важным элементом в стиле Forth. Так же, как Forth определения могут рассматриваться лингвистическим эквивалентом предложений на естественных языках, блок аналогичен параграфу. Блок обычно содержит определения, связанные с общей темой, типа "векторная арифметика". Комментарий в верхней строке блока идентифицирует эту тему. Приложение может выборочно загружать требующиеся блоки.

Блоки также используются для сохранения данных. Маленькие записи могут быть объединены в блок, или большие записи распространяться на несколько блоков. Программист может распределять блоки любым способом, удовлетворяющим приложение, и на чистых системах может увеличивать производительность, организуя данные для минимизирования движения головок диска. Несколько продавцов Forth разработали сложный файл и системы баз данных, основанные на блоках Forth.

Версии Forth, которые выполняются одновременно с OS находящейся в памяти главного компьютера, часто реализуют блоки в файлах OS главного компьютера. Другие используют базовые файлы исключительно. Стандарт требует, что бы блоки

были доступны на системах, предоставляющих любые методы дискового доступа, поскольку они - единственные средства ссылка на диск, который может быть перемещаемым между чистой и одновременно находящейся в памяти с OS реализациями.

C.5.6 Среда программирования

Хотя этот Стандарт не требует, чтобы большинство систем Forth включало резидентный редактор. Он дает возможность программисту редактировать источник и перетранслировать его в выполнимую форму без покидания среды Forth. Поскольку просто организовать приложение по уровням, часто возможно перетранслировать только самый верхний уровень (который обычно является в настоящее время разрабатываемым), процесс который редко занимает больше чем несколько секунд.

Большинство систем Forth также предоставляют резидентные интерактивные средства отладки, включающие не только слова типа тех, что в 15. Дополнительный набор слов Утилит, но также и наличие способности исследовать и изменять содержание VARIABLE и других элементов данных и выполнять с клавиатуры большинство составляющих слов основной системы Forth и разрабатываемого приложения.

Комбинация резидентного редактора, интегрированных средств отладки, и непосредственной выполняемости большинства определенных слов ведет к очень интерактивному стилю программирования, который демонстрирует сокращение времени разработки.

C.5.7 Расширенные возможности программирования

Одна из необычных характеристик Forth - в том, что слова определяемые программистом при построении приложения становятся неотъемлемыми элементами самого языка, добавляя более и более мощные проблемно-ориентированные возможности.

Например, Forth включает слова VARIABLE и 2VARIABLE для имени местоположения, в котором могут быть сохранены данные, также как CONSTANT и 2CONSTANT для имени значения одной и две-ячейки. Предположим, программист находит, что приложение нуждается в массивах, которые были бы автоматически индексированы по количеству элементов с двумя ячейками. Такой массив мог бы быть назван 2ARRAY. Префикс "2" в имени указывает, что каждый элемент в этом массиве займет две ячейки (как было бы содержание 2VARIABLE или 2CONSTANT). Префикс "2", однако, имеет значение только для человека и не более существенен для текстового интерпретатора, чем любой другой символ, который может использоваться для определения имени.

Такое определение имеет две части, как есть два "поведения", связанные с этим новым словом 2ARRAY, одно во время компиляции, и одно в работе или времени выполнения. Они лучшие понятны, если мы посмотрим, как 2ARRAY используется для определения его массивов, и затем как массив мог бы использоваться в приложении. Фактически, это как можно было бы проектировать и реализовать это слово.

В начале нисходящего процесса проектирования, вот как мы хотели бы использовать 2ARRAY:

```
100 2ARRAY RAW   50 2ARRAY REFINED
```

В первом случае, мы определяем массив 100 элементов длиной, чье имя - RAW. Во втором, массив - 50 элементов длиной, и именем REFINED. В каждом случае параметр размера поставляется для 2ARRAY на стеке данных (интерпретатор текста Forth автоматически помещает числа там, когда он сталкивается с ними), и имя слова следует непосредственно. Этот порядок типичен для определяющих слов Forth.

Когда мы используем RAW или REFINED, мы хотели бы поставлять на стеке индекс желаемого элемента, и возвратить на стеке адрес этого элемента. Такая ссылка типично имела бы место в цикле. Вот - типичный цикл, который принимает значение две-ячейки от гипотетического прикладного слова DATA и запоминает его в следующем элементе RAW:

```
: ACQUIRE   100 0 DO DATA  I RAW 2!  LOOP ;
```

Имя этого определения - ACQUIRE. Цикл начинается с DO, заканчивается LOOP, и будет выполняться с индексными значениями, изменяющимися от 0 до 99. В пределах цикла, DATA получает значение. Слово I возвращает текущее значение индекса цикла, которое является параметром для RAW. Адрес выбранного элемента, возвращенного RAW, и значение, которое осталось на стек от DATA, передаются слову 2! (произносимое "two-store"), которое сохраняет два элемента стека по адресу.

Теперь, когда мы точно определили, что делает 2ARRAY, и как ведут себя слова, которые оно определяет, мы готовы записать две части его определения:

```
: 2ARRAY  ( n -- )
   CREATE  2* CELLS ALLOT
   DOES>  ( i a -- a')  SWAP  2* CELLS + ;
```

Часть определения перед словом DOES>, определяет поведение "во время компиляции", то есть что будет делать 2ARRAY, когда оно нами используется для определения слова типа RAW. Комментарий указывает, что эта часть ожидает число на стеке, которое является параметром размера. Слово CREATE создает определение для нового слова. Фраза 2* CELLS преобразовывает параметр размера из элементов с двумя ячейками во внутренние адресуемые элементы системы (обычно символы). ALLOT затем распределяет определенный объем памяти для содержания данных, который будет связан с недавно определенным массивом.

Вторая строка определяет поведение "времени-выполнения", которое будет разделено всеми словами определенными 2ARRAY, типа RAW и REFINED. Слово DOES>, заканчивает первую часть определение и начинает вторую часть. Второй комментарий здесь указывает, что этот код ожидает индекс и адрес на стеке, и возвратит другой адрес. Индекс помещен на стеке вызывающей программой (RAW в примере), в то время как адрес содержимого слова, определенного таким образом (область определенная ALLOT) автоматически помещается на вершину стека перед выполнением этой части кода. Этот код работает таким образом: SWAP изменяет порядок двух элементов стека, для получения индекса на вершине. 2* CELLS преобразовывает индекс к внутренним адресуемым элементам как в разделе времени компиляции, для выдачи смещения от начала массива. Слово + тогда добавляет смещение к адресу начала массива, чтобы дать исполнительный адрес, который является желательным результатом.

Данное основное определение, можно легко изменять, для того чтобы делать более сложные вещи. Например, код времени компиляции может быть изменен для инициализации массива нулями, пробелами, или любыми другими желательными начальными значениями. Размер массива может быть скомпилирован в его начало так, чтобы код времени-выполнения мог сравнивать с ним индекс, чтобы гарантировать его в пределах диапазона, или полный массив может быть создан постоянно находящимся на диске вместо главной памяти. Ни одно из этих изменений не затронуло бы использование времени-выполнения, которое мы определили другим способом. Это иллюстрирует немного гибкости, доступной с этими определяющими словами.

C.5.8 Пример программирования

Рисунок C.1 содержит типичный блок источника Forth. Он представляет часть

приложения, которое управляет банком восьми светодиодов, используемых как лампы индикатора на приборе, и указывает некоторые из путей, которыми определения Forth различных видов объединяются в среду прикладной программы. Этот пример был кодирован для STD-bus системы с 8088 процессором и миллисекундными часами, которые также используются в примере.

Светодиоды связаны с помощью интерфейса через отдельный 8 битный порт, чей адрес 40H. Это местоположение определено как CONSTANT в Строке 1, так чтобы оно могло быть упомянуто по имени; если модифицируется адрес, необходимо только корректировать значение этой константы. Слово LIGHTS возвращает этот адрес на стеке. Определение LIGHT берет значение со стека и посылает его на устройство. Природа этого значения - битовая маска, чьи биты соответствуют непосредственно индивидуальным огням.

Таким образом, команда 255 LIGHT включит все огни, тогда как 0 LIGHT их всех выключит.

```
Block 180
0. ( Светодиодное управление )
1. HEX 40 CONSTANT LIGHTS   DECIMAL
2. : LIGHT ( n -- )  LIGHTS OUTPUT ;
3.
4. VARIABLE DELAY
5. : SLOW  500 DELAY ! ;
6. : FAST  100 DELAY ! ;
7. : COUNTS 256 0 DO I LIGHT  DELAY @ MS  LOOP ;
8.
9. : LAMP ( n -  )  CREATE ,  DOES> ( a -- n )   @ ;
10. 1 LAMP POWER       2 LAMP HV     4 LAMP TORCH
11. 8 LAMP SAMPLING   16 LAMP IDLING
12.
13. VARIABLE LAMPS
14. : TOGGLE ( n -- ) LAMPS @ XOR DUP LAMPS ! LIGHT ;
15.
```

Рисунок С.1 - блок источника Forth, содержащий слова, которые управляют набором светодиодов.

Строки 4 - 7, содержат простую диагностику типа, которая может быть введена с терминала для подтверждения, что все работает. Переменная DELAY содержит время задержки в миллисекундах - выполнение слова DELAY возвращает адрес этой переменной. Два значения DELAY устанавливаются определениями SLOW и FAST, используя оператор Forth ! (произносится "store") который берет значение и адрес, и сохраняет значение по адресу. Определение COUNTS выполняет цикл от 0 до 255 (циклы Forth этого типа ограничены верхним пределом диапазона), посылая каждое значение к огням и затем ожидание в течение периода, определенного DELAY. Слово @ (произносится "fetch") выбирает значение по адресу, в этом случае адрес поставляется DELAY. Это значение посылается к MS, которое ждет определенного числа миллисекунд. Результат выполнения COUNTS такой, что огни будут считать от 0 до 255 с желательной скоростью. Чтобы выполнить это, введите одно слово:

```
SLOW COUNTS   or   FAST COUNTS
```

с терминала.

Строка 9 предоставляет возможность обозначения индивидуальных ламп. В этом приложении они используются как огни индикатора. Слово LAMP является определяющим словом, которое берет как параметр маску, которая представляет специфическую лампу, и компилирует ее как именованный объект. Строки 10, и 11 содержат пять использований LAMP для наименования специфических индикаторов.

Когда одно из этих слов типа POWER выполнено, маска возвращается на стеке. Фактически, поведение определенного значения такое, что когда слово вызвано, возвращенное значение идентично поведению Forth слова CONSTANT. Мы создавали здесь новое определяющее слово, однако, чтобы иллюстрировать, как это было бы сделано.

Наконец, в строках 13 и 14, мы имеем слова, которые будут управлять световой панелью. LAMPS является переменной, которая содержит текущее состояние ламп. Слово TOGGLE берет маску (которая может быть получена от одного из слов LAMP) и изменяет состояние этой специфической лампы, сохраняя результат в LAMPS.

В конце приложения, имена ламп и TOGGLE - вероятно единственные слова, которые будут выполнены непосредственно. Использование там будет, например:

```
POWER TOGGLE   or   SAMPLING TOGGLE
```

как соответствующее, всякий раз, когда должны быть изменены системные индикаторы.

Время для компиляции этого блока кода на той системе было приблизительно половина секунды, включая время выбора его с диска. Так что это весьма практично (и нормальная практика) для программиста, чтобы просто напечатать определение и пробовать его немедленно.

Кроме того, каждый всегда имеет возможность связи с внешними устройствами непосредственно. Первая вещь, которую можно было бы делать, когда говорится о лампах, будет состоять в том, чтобы напечатать:

```
HEX FF 40 OUTPUT
```

и смотреть, включаются ли все лампы. Если нет - предположение, что кое-что неправильно с аппаратными средствами, так как эта фраза непосредственно передает "полную" маску на устройство. Этот тип прямого взаимодействия полезен в приложениях, вовлекающих заказные аппаратные средства, так как он уменьшает время отладки аппаратных средств.

C.6 Мультипрограммные системы

Мультипрограммные системы Forth существовали приблизительно с 1970. Самые ранние доступные Forth системы распространяли "перехватчики" для этой возможности, несмотря на то, что многие не использовали их. Однако основные предположения были общеизвестны в сообществе, и существуют значительные точки соприкосновения среди этих мультипрограммных систем. Эти системы не только языковые процессоры, но содержат также характеристики операционной системы. Многие из этих интегрированных систем работают полностью автономно, выполняя все необходимые функции операционной системы.

Некоторые системы Forth очень быстры, и могут поддерживать, многозадачный режим и многопользовательские операции даже на компьютерах, чьи аппаратные средства, обычно полагается, неспособны к таким продвинутым операциям. Например, один производитель телефонных коммутаторов выполняет 50 задач на Z80. Есть несколько мультипрограммных изделий для PC, некоторые из которых даже поддерживают много пользователей. Даже на компьютерах, которые обычно используются в многопользовательских операциях, число пользователей, которые могут быть поддержаны, может быть намного больше, чем ожидается. Одно большое приложение базы данных выполняющееся на единственном 68000 имеет более чем 100 терминалов модифицирующих и запрашивающих эту базу данных, без существенного ухудшения свойств.

Многопользовательские системы могут также поддерживать множество программистов, каждый из которых имеет частный словарь, стеки, и набор переменных, управляющих

этой задачей. Частный словарь связан с общедоступным, реентерабельным словарем, содержащим все стандартные функции Forth. Частный словарь может использоваться для разработки прикладного кода, который может позже быть интегрирован в общедоступный словарь. Он может также использоваться для выполнения функций требующих текстовый интерпретатор, включая компиляцию и выполнение исходного текста.

C.7 Конструкционные и организационные соображения

Также, как выбор строительных материалов имеет сильный эффект на дизайн и конструкцию постройки, выбор языка и операционной системы затронет конструкцию приложения и решения руководства проектом.

Традиционно, программные проекты проходят через четыре стадии: анализ, конструкция, кодирование, и испытание. Проект Forth также обязательно включает эти действия. Forth оптимизирован для методологии управления проектом, свойственной маленьким группам квалифицированных профессионалов. Forth поощряет итерационный процесс "последовательного макетирования" в котором Forth высокого уровня используется как выполняемая утилита конструкции, с "заглушками", заменяющими подпрограммы низшего уровня по мере необходимости (например, для аппаратных средств, которые все еще не сформированы).

Во многих случаях последовательное макетирование может производить доброкачественное, более полезное изделие. Как проектные решения, разработчики изучают вещи, которые могут вести к лучшей конструкции. Более мудрые решения могут быть сделаны, если точные связанные затраты известны, и часто это невозможно пока код прототипа не может быть написан и испытан.

Использование Forth может сокращать время, требуемое для программной разработки, и так же уменьшать уровень усилий, требуемых для обслуживания и модификаций в течение жизни изделия.

C.8 Заключение

Forth произвел некоторые замечательные достижения в разнообразии прикладных областей. В течение нескольких лет, его принятие выросло быстро, особенно среди программистов ищущих способов улучшить их производительность, и менеджеров ищущих способы упростить новые проекты программной разработки.

D. Анализ Совместимости ANS Forth (информационное приложение)

До ANS Forth, было несколько промышленных стандартов для Forth. Наиболее влиятельные перечислены здесь в хронологическом порядке, наряду с главными различиями между ANS Forth и самым современным, Forth 83.

D.1 FIG Forth (приблизительно 1978)

FIG Forth был "образцовой" реализацией языка Forth, разработанного Forth Interest Group (FIG). В FIG Forth, сравнительно мало слов было реализовано на процессорно-зависимом машинном языке, остальные слова были реализованы в Forth. Модель FIG была размещена на всеобщее достояние, и была перенесена на широкое разнообразие компьютерных систем. Поскольку большая часть реализаций FIG Forth была одна и та же на всех машинах, программы, написанные в FIG Forth, обладали существенной степенью переносимости, даже для программ "системного уровня", которые непосредственно управляют внутренней организацией системной реализации Forth.

FIG Forth реализации имели решающее влияние в увеличении числа людей,

заинтересованных в использовании Forth. Много людей связывают методы реализации, воплощенные в FIG Forth модели с "сущностью Forth".

Однако FIG Forth не обязательно был представителем коммерческой реализации Forth того же самого периода. Часть наиболее успешных коммерческих систем Forth использовали методы реализации отличные от "модели" FIG Forth.

D.2 Forth 79

Forth-79 Стандарт - результат ряда встреч с 1978 по 1980, Forth Standards Team, международной группы пользователей Forth и продавцов (промежуточные версии известные как Forth 77 и Forth 78 были также выпущены группой).

Forth 79 описал набор слов, определенных на 16-бит виртуальной машине, с дополнительным кодом (дополнение до двух), не выровненной, линейной адресацией байта. Это предписало методику реализации известную как "косвенный шитый код", и использование ASCII набора символов.

Forth-79 Стандарт служил как основание для нескольких общедоступных и коммерческих реализаций, некоторые из которых все еще доступны и поддерживаются сегодня.

D.3 Forth 83

Forth-83 Стандарт, также Forth Standards Team, был выпущен в 1983. Forth 83 предпринял попытку, устранить некоторые из неточностей Forth 79.

Forth 83 в большинстве отношений был подобен Forth 79. Однако Forth 83 изменил определение нескольких вполне определенных особенностей Forth 79. Например, поведение округления целочисленного деления, основное значение операндов PICK и ROLL, значение адреса, возвращаемого ', поведение компиляции ', значение "true" флага, значение NOT, и "последовательное" поведение слов, определенных VOCABULARY были полностью изменены. Forth 83 ослабил ограничения реализации Forth 79, чтобы позволить любой вид шитого кода, но не полностью позволял компиляцию чистого машинного кода (это не было специально запрещено, а скорее было косвенное последствие другого условия).

Много новых реализаций Forth были основаны на Forth-83 Стандарте, но немного существует "строго совместимых" реализаций Forth-83.

Хотя несовместимость, следующую из изменений между Forth 79 и Forth 83, было обычно относительно просто устранить, множество преуспевающих продавцов Forth не преобразовало свои реализации, чтобы быть совместимыми с Forth 83. Например, наиболее преуспевающий коммерческий Forth для компьютеров Apple Macintosh основан на Forth 79.

D.4 Недавние события

С тех пор как был издан Forth-83 Стандарт, компьютерная промышленность подверглась быстрым и глубоким изменениям. Скорость, емкость памяти, и объем диска доступных персональных компьютеров увеличились больше чем в 100 раз. 8-бит процессоры уступили 16-бит процессорам, и теперь 32-бит процессоры обычное явление.

Операционные системы и среды языков программирования маленьких систем - гораздо более мощные, чем они были в начале 80-ых.

Рынок персонального компьютера изменился от преобладающего рынка "хобби" до зрелого делового и коммерческого рынка.

Улучшенная технология для проектирования заказных микропроцессоров привела к

разработке многочисленных "чипов Forth", компьютеров, оптимизированных для выполнения языка Forth.

Рынок для ROM-базирующихся внедренных управляющих компьютеров существенно вырос.

Чтобы получать полное преимущество этой развивающейся технологии, и лучше конкурировать с другими языками программирования, много недавних реализаций Forth игнорировали некоторые "правила" предыдущих стандартов Forth. В особенности:

- 32-разрядные реализации Forth теперь обычны.
- Некоторые системы Forth принимают ограничения выравнивания адреса аппаратных средств, на которых они работают.
- Некоторые системы Forth используют генерацию чистого кода, генерацию микропрограмм, и методы оптимизации, скорее, чем традиционный "шитый код".
- Некоторые системы Forth используют архитектуру сегментированной адресации, размещения частей Forth "словаря" в различных сегментах.
- Все более систем Forth теперь работают в среде другой "стандартной" операционной системы, используя текстовые файлы OS для исходного текста, скорее, чем традиционные Forth "блоки".
- Некоторые системы Forth допускают программное обеспечение внешней операционной системы, программное обеспечение работы с окнами, концентраторы терминала, или каналы связи, для обработки или препроцессорной обработки пользовательского ввода, приводя к отступлениям от редактирования ввода, пригодности набора символов, и управление поведением экрана, предписанное Forth 83.

Конкурентное давление из других языков программирования (преимущественно "C") и от других продавцов Forth вели продавцов Forth к оптимизации, которая не соответствует хорошо "модели виртуальной машины" подразумеваемой по существующим стандартам Forth.

D.5 ANS Forth подход

ANS Forth комитет обратился к серьезной фрагментации семейства Forth вызванной различиями между Forth 79 и Forth 83, и отклонениями от любого из этих двух промышленных стандартов, вызванными давлением рынка.

Следовательно, комитет хотел основывать свои решения совместимости не на строгом сравнении со Стандартом Forth-83, но вместо этого на рассмотрении разнообразия существующих реализаций, особенно тех, что с существенными пользовательскими базами и/или значительным успехом на рынке.

Комитет чувствует что, если ANS Forth предпишет строгие требования к виртуальной машинной модели, так же как и предыдущие стандарты, тогда много разработчиков не выберет исполнение ANS Forth. Комитет надеется, что ANS Forth будет служить для объединения скорее, чем для дальнейшего деления семейства Forth, и таким образом выбрал обобщение скорее, чем лишение законной силы популярных методов реализации.

Многие из изменений в Forth 83 оправданы этим объяснением. Больше всего попадает в категорию, что "ANS Forth Стандартная Программа не может принимать x", где "x" - наименование, вытекающее из виртуальной машинной модели, предписанной Forth-83 Стандартом. Комитет чувствует, что эти ограничения разумны, особенно полагая, что существенное число существующих реализаций Forth не реализует правильно Forth-83 виртуальную модель, таким образом, Forth-83 наименование существует "теоретически" но не "практически".

Другой взгляд на это - это то, что в то время как ANS Forth признает

разнообразие текущей практики Forth, он пытается документировать в этом аналогию. В некотором смысле, ANS Forth - таким образом "описание действительности" скорее чем "предписание для специфической виртуальной машины".

Так как нет никакого предыдущего American National Standard для Forth, требования действий, предписанные разделом 3.4 X3/SD-9, "Политика и Рекомендации", относительно предыдущих стандартов не применяются.

Следующее обсуждение описывает различия между ANS Forth и Forth 83. В большинстве случаев, Forth 83 образец Forth 79 и FIG Forth для целей этого обсуждения. Во многих из этих случаев, однако, ANS Forth более типичный для существующего состояния индустрии Forth, чем предварительно изданные стандарты.

D.6.1 Ширина стека

Forth 83 определяет, что элементы стека занимают 16 бит. Это включает адреса, флаги, и числа. ANS Forth определяет, что элементы стека - по крайней мере 16 бит; фактический размер должен быть документирован реализацией.

Затрагиваемые слова: вся арифметика, логические и адресующие операторы

Причина: 32-бит машины становятся банальными. 16-бит система Forth на 32-бит машине - не конкурентоспособна.

Воздействие: Программы, которые принимают 16-бит ширину стека, продолжат работать на 16-бит машинах; ANS Forth не требует другой ширины стека, но просто позволяет это. Много программ не будут затронуты (но см. "адресуемый элемент").

Переход/Преобразование: Программы, которые используют разрядные маски с набором старших бит, вероятно, придется изменить, заменяя определенную реализацией константу битовой маски, или процедуру вычисляющую битовую маску независимым от ширины стека способом. Вот - некоторые процедуры для построения независимых от ширины битовых масок:

```
1  CONSTANT LO-BIT

TRUE 1 RSHIFT  INVERT  CONSTANT HI-BIT

: LO-BITS  ( n -- mask )  0 SWAP  0 ?DO  1 LSHIFT  LO-BIT OR  LOOP ;
: HI-BITS  ( n -- mask )  0 SWAP  0 ?DO  1 RSHIFT  HI-BIT OR  LOOP ;
```

Программы, которые зависят от поведения "по модулю '553' ", подразумеваемого в операциях 16-бит арифметики, должны быть переписаны, чтобы явно выполнять операцию модуля в соответствующих местах. Комитет верит, что такие предположения происходят нечасто.
Примеры: некоторая контрольная сумма или вычисление CRC, некоторые генераторы случайного числа и большая часть дробной математики с фиксированной точкой.

D.6.2 Представление числа

Forth 83 определяет представление числа и арифметики дополнительным кодом (дополнение до двух). ANS Forth также позволяет обратный код (дополнение до единицы) и величину со знаком.

Затрагиваемые слова: вся арифметика и логические операторы, LOOP, +LOOP

Причина: Некоторые компьютеры используют обратный код (дополнение до единицы) или величину со знаком. Комитет не желал принуждать реализации Forth для этих машин эмулировать арифметику с дополнительным кодом (дополнение до двух), и таким образом подвергаться серьезному ухудшению производительности. Опыт

некоторых членов комитета с такими машинами указывает, что ограничения использования необходимые для поддержки этих представлений чисел не обременительны чрезмерно.

Воздействие: ANS Forth Стандартная Программа может объявлять "зависимость от окружения для арифметики с дополнительным кодом (дополнение до двух)". Это означает иначе, что стандартная программа гарантированно будет работать только на машинах с дополнительным кодом (дополнение до двух). Действительно, это - не серьезное ограничение, потому что подавляющее большинство современных компьютеров использует дополнительный код (дополнение до двух). Комитет знает не совместимые с Forth-83 реализации для машин с не дополнительным кодом (дополнение до двух) в настоящее время, таким образом, существующие Forth-83 программы будут все еще работать на том же классе машин, на котором они в настоящее время работают.

Переход/Преобразование: Существующие программы, желающие воспользоваться преимуществом возможности ANS Forth Стандартной Системы на машинах не с дополнительным кодом (дополнение до двух) могут делать так, устраняя использование арифметических операторов для выполнения логических функций, получая константы битовой маски от битовых операций как описано в разделе о ширине стека, ограничивая диапазон использования чисел без знака до диапазона положительных чисел, и используя предусмотренные операторы для преобразования из одинарных чисел в двойные числа.

D.6.3 Адресуемые элементы

Forth 83 определяет, что каждый уникальный адрес относится к 8-бит байту в памяти. ANS Forth определяет, что размер элемента, адресуемого каждым уникальным адресом - определенное реализацией, но, по умолчанию, является размером одного символа. Forth 83 описывает много операций памяти в терминах множества байт. ANS Forth описывает эти операции в терминах множества символов или адресуемых элементов.

Затрагиваемые слова: те, что с параметром "адресуемого элемента"

Причина: Некоторые машины, включая наиболее популярные чипы Forth, адресуют 16-бит адреса ячеек памяти вместо 8-бит байтов.

Воздействие: Программы могут выбирать объявление зависимости от окружения для байтовой адресации, и продолжать работать на классе машин, для которых они теперь работают. Для реализации Forth на адресуемой словом машине, для Forth 83 совместимости, она должна была бы моделировать байтовую адресацию за значительную цену в скорости и эффективности памяти. Комитет не знает ни о какой такой Forth-83 реализации для таких машин, таким образом, зависимость от окружения для байтовой адресации не ограничивает Стандартную Программу сверх ее текущих фактических ограничений.

Переход/Преобразование: Новые операторы адресной арифметики CHARS и CHAR+ должны быть использованы для программ, которые требуют переносимости на машины с не байтовой адресацией. Места, где такое преобразование необходимо, могут быть идентифицированы, поиском местонахождения слов которые принимают множество адресуемых элементов как параметр (например, MOVE , ALLOT).

D.6.4 Приращение адреса для ячейки больше не два

Как последствие Forth-83 одновременной спецификации 16-бит ширины стека и байтовой адресации, число два могло надежно использоваться в вычислениях адреса затрагивающих массивы памяти, содержащие элементы со стека. Так как ANS Forth не требует ни 16-бит ширины стека ни адресации байта, число два больше не обязательно соответствует таким вычислениям.

Затрагиваемые слова: @ ! +! 2+ 2* 2- +LOOP

Причина: См. причины для "Адресуемые элементы" и "Ширина стека"

Воздействие: В этом отношении, существующие программы продолжат работать на машинах, где ячейка стека занимает два адресуемых элемента, при сохранении в памяти. Это включает большинство машин, для которых в настоящее время существуют Forth 83 совместимые реализации. В принципе, это также включило бы адресуемые 16-бит словами машины с 32-бит шириной стека, но комитет не знает о примерах таких машин.

Переход/Преобразование: Новые операторы адресной арифметики CELLS и CELL+ должны быть использованы для переносимых программ. Места, где такое преобразование необходимо, могут быть идентифицированы, поиском символа "2" и определением, действительно ли оно используется как часть вычисления адреса. Следующие замены уместны при вычислении адреса:

Старое	Новое
2+ or 2 +	CELL+
2* or 2 *	CELLS
2- or 2 -	1 CELLS -
2/ or 2 /	1 CELLS /
2	1 CELLS

Число "2" отдельно иногда используется для вычислений адреса как параметр к +LOOP, когда индекс цикла - адрес. При преобразовании слова 2/, которое работает на отрицательных делимых, нужно быть осведомленным об используемом методе округления.

D.6.5 Выравнивание адреса

Forth 83 не налагает никаких ограничений на выравнивание адресов к какой либо границе. ANS Forth определяет, что Стандартная Система может требовать выравнивание адресов для использования с различными операторами "@" и "!".

Затрагиваемые слова: ! +! 2! 2@ @ ? ,

Причина: Много компьютеров имеют аппаратные ограничения, которые поддерживают использование выровненных адресов. На некоторых машинах, чистые команды доступа к памяти вызовут прерывание по исключительной ситуации, если используются с не выровненным адресом. Даже на машинах где не выровненные обращения не вызывают прерывание по исключительной ситуации, выровненные обращения обычно быстрее.

Воздействие: Все ANS Forth слова, что возвращают адреса, применимые для использования с выровненными словами "@" и "!" должны возвращать выровненные адреса. В большинстве случаев, это не будет проблемой. Проблемы могут являться результатом использования определяемых пользователем структур данных содержащих смесь символьных данных и данных размера ячейки.

Много существующих систем Forth, особенно используемых в настоящее время на компьютерах с жесткими требованиями выравнивания, уже требуют выравнивание. Много существующего кода Forth, который в настоящее время используется на таких машинах, уже было преобразовано для использования в выровненных средах.

Переход/Преобразование: Есть два возможных подхода к преобразованию программ для использования на системе, требующей выравнивание адреса.

Самый простой подход состоит в том, чтобы переопределить систему выровненных операторов "@" и "!" так, чтобы они не требуовали выравнивания. Например, на 16-бит машине с прямым порядком байт и байтовой адресацией, не выровненные "@"

и "!" могли быть определены:

```
: @  ( addr -- x )  DUP C@ SWAP CHAR+ C@ 8 LSHIFT OR  ;

: !  ( x addr -- )  OVER 8 RSHIFT OVER CHAR+ C! C!  ;
```

Эти определения, и подобные для "+!", "2@", "2!", ",", и "?" при необходимости, могут быть откомпилированы перед не выровненным приложением, которое будет тогда работать как ожидается.

Этот подход может оберегать память, если приложение использует существенное количество структур данных, содержащих не выровненные поля.

Другой подход состоит в том, чтобы изменить исходный текст приложения для устранения не выровненных полей данных. ANS Forth слова ALIGN и ALIGNED могут использоваться для вынужденного выравнивания полей данных. Места, где такое выравнивание необходимо, могут быть определены обследованием частей приложения, где определены структуры данных (отличные от простых переменных), или методами "интеллектуального компилятора" (см. обсуждение "интеллектуального компилятора" ниже).

Этот подход, вероятно, приведет к более быстрой скорости выполнения приложения, при возможном увеличении расхода использования памяти для структур данных.

Наконец, возможно комбинировать предшествующие методы, идентифицируя точно те поля данных, которые не выровнены, и используя "не выровненные" версии операторов доступа к памяти только для этих полей. Этот "гибридный" подход определяет компромисс между скоростью выполнения и использованием памяти.

D.'.' Направление округления Division/modulus

Forth 79 определяет, что деление округляется к 0 и остаток несет знак делимого. Forth 83 определяет, что деление округляется к отрицательной бесконечности и остаток несет знак делителя. ANS Forth позволяет любое поведение для операторов деления перечисленных ниже на усмотрение разработчика, и предоставляет пару примитивов деления, чтобы позволить пользователю синтезировать любое определенное поведение.

Затрагиваемые слова: / MOD /MOD */MOD */

Причина: Различие между поведением деления в Forth 79 и Forth 83 было объектом большого спора, и много реализаций Forth не переключались на поведение Forth 83. Оба варианта имеют шумных сторонников, цитирующих требования к приложениям и аргументы эффективности выполнения с обеих сторон. После обширных дебатов охватывающих много встреч, комитет был не способен достигнуть согласия в выборе того или иного поведения, и выбрал разрешение любого поведения как значения по умолчанию, в тоже время предусматривая пользовательские средства для явного использования любого поведения по необходимости. Так как разработчикам разрешен выбор любого поведения, не требуется изменять поведение, показанное их текущими системами, таким образом, сохраняя правильное функционирование существующих программ, которые работают на этих системах и зависят от специфического поведения. Новые реализации могли выбирать поведение, которое поддерживается родной системой команд CPU, таким образом максимизируя скорость выполнения, или могли выбирать поведение, которое больше соответствует предназначенной прикладной области системы.

Воздействие: Проблема затрагивает только программы, которые используют отрицательное делимое с положительным делителем, или положительное делимое с отрицательным делителем. Огромное большинство использования деления происходят, с положительным делимым и положительным делителем; в этом случае, результат один и тот же для обоих разрешенных поведений деления.

Переход/Преобразование: Для программ, которые требуют специфического поведения округления с операндами деления смешанного знака, операторы деления, используемые в программе, могут быть переопределены в терминах одного из новых примитивов деления ANS Forth SM/REM (симметричное деление, то есть округление к нулю) или FM/MOD (минимальное деление, то есть, округление к отрицательной бесконечности). Тогда программа может быть перетранслирована без изменения. Например, операторы деления стиля Forth 83 могут быть определены:


```
: /MOD  ( n1 n2 -- n3 n4 )  >R S>D R> FM/MOD  ;
: MOD   ( n1 n2 -- n3 )  /MOD DROP  ;
: /     ( n1 n2 -- n3 )  /MOD SWAP DROP   ;
: */MOD ( n1 n2 n3 -- n4 n5 )  >R M* R> FM/MOD  ;
: */    ( n1 n2 n3 -- n4 n5 )  */MOD SWAP DROP  ;
```

D.6.7 Immediate-ности

Forth 83 определил, что множество "компилирующих слов" являются "немедленного исполнения", означая что они выполняются вместо компилирования в течение компиляции. ANS Forth менее конкретен относительно большинства этих слов, заявляя, что их поведение определено только в течение компиляции, и определяя их результат скорее, чем их специфические действия во время компиляции.

Чтобы заставить компилироваться слово, которое обычно выполнилось бы, Forth 83, предусматривал слова COMPILE используемое со словами не немедленного исполнения, и [COMPILE] используемое со словами немедленного исполнения. ANS Forth предусматривает единственное слово POSTPONE , которое используется и с немедленного исполнения и с не немедленного исполнения словами, автоматически выбирая соответствующее поведение.

Затрагиваемые слова: COMPILE [COMPILE] ['] '

Причина: Обозначение специфических слов как немедленного исполнения или нет, зависит от методики реализации, выбранной для системы Forth. В традиционных реализациях "шитого кода", выбор был вообще довольно ясный (с единственным исключением слова LEAVE), и стандарт мог определить какие слова должны быть немедленного исполнения. Однако, некоторые в настоящее время популярные методы реализации, типа генерация чистого кода с оптимизацией, требуют атрибута немедленного исполнения на наборе слов отличном от набора слов немедленного исполнения реализации шитого кода. ANS Forth, подтверждая законность этих других методов реализации, определяет атрибут немедленного исполнения в таких немногих случаях, насколько это возможно.

Когда состав набора слов немедленного исполнения неясен, решение использовать ли COMPILE или [COMPILE], становится неясным. Следовательно, ANS Forth предоставляет "универсальное" слово замены POSTPONE, которое обслуживает цели огромного большинства применений COMPILE и [COMPILE] без требования знания пользователем, действительно ли "отложенное" слово немедленного исполнения.

Точно так же использование ' и ['] с компилирующими словами не ясно, если точное поведение компиляции этих слов не определено, так что ANS Forth не разрешает Стандартной Программе использовать ' или ['] с компилирующими словами.

Традиционное (не немедленного исполнения) определение слова COMPILE имеет дополнительную проблему. Его традиционное определение принимает методику реализации шитого кода, и его поведение может быть должным образом описано только в этом контексте. В контексте ANS Forth, который разрешает другие методы реализации в дополнение к шитому коду, очень трудно, если не невозможно описать поведение традиционного COMPILE. Скорее чем изменить его поведение, и таким образом нарушить существующий код, ANS Forth не включает слово COMPILE. Это

позволяет существующим реализациям продолжить снабжать слово COMPILE его традиционным поведением, если это соответствует реализации.

Воздействие: [COMPILE] остается в ANS Forth, так как его надлежащее использование не зависит от знания того, действительно ли слово является немедленного исполнения (Использование [COMPILE] со словом не немедленного исполнения является, и всегда было пустой командой). Действительно ли Вы должны использовать [COMPILE] требует знания того, действительно ли его целевое слово является словом немедленного исполнения, но использовать [COMPILE] всегда безопасно. [COMPILE] - больше не в (требуемом) основном наборе слов, будучи перемещенное в Расширение Основного набора слов, но комитет предупреждает, что большинство продавцов поставят его, так или иначе.

Почти во всех случаях, правильно заменить [COMPILE] и COMPILE на POSTPONE. Использования [COMPILE] и COMPILE, которые - "бессмысленно" менять на POSTPONE , являются весьма редкими, и относятся следующим к двум категориям:

a) Использование [COMPILE] с не немедленного исполнения словами. Это иногда делается со словами ' (tick) (которое было немедленного исполнения в Forth 79, но не в Forth 83) и LEAVE (которое было немедленного исполнения в Forth 83 но не в Forth 79), чтобы вынуждать компиляцию этих слов без относительно к тому, используете ли Вы Forth 79 или Forth 83 систему.

b) Использование фразы COMPILE [COMPILE] <слово немедленного исполнения> для "двойного postpone" слова немедленного исполнения.

Переход/Преобразование: Много ANS Forth реализаций продолжит реализовывать оба [COMPILE] и COMPILE в формах, совместимых с существующим использованием. В этих средах, нет необходимости в преобразовании.

Для полной переносимости, использование COMPILE и [COMPILE] должно быть заменено на POSTPONE, кроме редких случаев обозначенных выше. Использование [COMPILE] со словами не немедленного исполнения, может быть оставлено как есть, и программа может объявить, что требуется слово [COMPILE] из Расширения Основного набора слов, или [COMPILE] перед словом не немедленного исполнения может быть просто удалено, если целевое слово, как известно, является не немедленного исполнения.

Использование фразы COMPILE [COMPILE] <слово немедленного исполнения> может быть обработано, представляя "промежуточное слово" (XX в примере ниже) и затем выполняя POSTPONE с этим словом. Например:

```
: ABC  COMPILE [COMPILE] IF  ;
```

изменения:

```
: XX  POSTPONE IF  ;
: ABC  POSTPONE XX  ;
```

Нестандартный случай может происходить в программах, которые "выключают состояние компиляции" для явной компиляции потока в словарь после COMPILE. Например:

```
: XYZ  COMPILE  [ ' ABC , ]  ;
```

Это сильно зависит от точного знания, как работают COMPILE и реализация шитого кода. Случаи подобно этому не могут быть обработаны механически; они должны быть транслированы с точным пониманием того, что делает код, и переписыванием этого раздела согласно ANS Forth ограничениям.

Используйте фразу POSTPONE [COMPILE] для замены [COMPILE] [COMPILE].

D.6.8 Входной набор символов

Forth 83 определяет, что полный 7-бит набор символов ASCII доступен через KEY. ANS Forth ограничивает его графическими символами набора ASCII, с кодами от hex 20 до hex 7E включительно.

Затрагиваемые слова: KEY

Причина: Много системных сред "потребляют" некоторые управляющие символы для таких целей как редактирование ввода, управление заданиями, или управление потоком данных. Реализация Forth не всегда может управлять этим системным поведением.

Воздействие: Стандартные Программы, которые требуют способности получить специфические управляющие символы через KEY должны объявить зависимость от окружения для входного набора символов.

Переход/Преобразование: Для максимальной переносимости, программы должны ограничить их требуемый входной набор символов только графическими символами. Управляющие символы могут быть обработаны, если доступны, но полные функциональные возможности программы должны быть доступны с использованием только графических символов.

Как определено выше, может быть объявлена зависимость от окружения на входном наборе символов. Даже в этом случае, рекомендуется, чтобы программа избегала требования особенно неприятных управляющих символов, типа control-S и control-Q (часто используемых для управления потоком, иногда аппаратными средствами связи, чье присутствие может быть трудно обнаруживаемо), ASCII NUL (трудно напечатать на многих клавиатурах), и различий между возвратом каретки и переводом строки (некоторые системы транслируют возвраты каретки в переводы строки, или наоборот).

D.6.9 Сдвиги с UM/MOD

Данная Forth-83 природа дополнительного кода (дополнение до двух), и его требование минимального деления (округление к минус бесконечности), сдвиг эквивалентен делению. Также, представление дополнительного кода (дополнение до двух) подразумевает, что деление без знака величиной два эквивалентно логическому правому сдвигу, так UM/MOD могло использоваться для исполнения логического правого сдвига.

Затрагиваемые слова: UM/MOD

Причина: Проблема с UM/MOD - это результат разрешения представления числа не дополнительным кодом (дополнение до двух), как уже описано.

ANS Forth предоставляет слова LSHIFT и RSHIFT для выполнения логических сдвигов. Это - обычно более эффективно, и конечно более наглядно, чем использование UM/MOD для логического сдвига.

Воздействие: Программы, выполняющиеся на ANS Forth системах с арифметикой дополнительного кода (дополнение до двух) (большинство машин), не будут испытывать никакой несовместимости с UM/MOD. Существующие Forth-83 Стандартные программы, предназначенные для работы на машинах с не дополнительным кодом (дополнение до двух), не смогут использовать UM/MOD для смещения на ANS Forth системе с не дополнительным кодом (дополнение до двух). Это не должно затронуть значительное число существующих программ (возможно ни какие), так как комитет не знает ни о каких существующих Forth-83 реализациях на машинах с не дополнительным кодом (дополнение до двух).

Переход/Преобразование: Программа, которая требует, чтобы UM/MOD вел себя как операция сдвига, может объявить зависимость от окружения на арифметике дополнительного кода (дополнение до двух).

Программа, которая не может объявить зависимость от окружения на арифметике дополнительного кода (дополнение до двух) может требовать редактирования для замены несовместимого использования UM/MOD другими операторами, определенными в пределах приложения.

D.6.10 Словари / списки слов

ANS Forth не определяет слова VOCABULARY, CONTEXT, и CURRENT, которые существовали в Forth 83. Вместо этого, ANS Forth определяет примитивный набор слов для спецификации порядка поиска и управления порядком поиска, включая слова, которые не существовали ни в каком предыдущем стандарте.

Forth-83 "ALSO/ONLY" экспериментальный набор слов порядка поиска определен главным образом как часть расширения ANS Forth набора слов Порядка Поиска.

Затрагиваемые слова: VOCABULARY CONTEXT CURRENT

Причина: Словари - область больших расхождений среди существующих систем. Рассматривая системы главных продавцов и предыдущие стандарты, есть, по крайней мере, 5 различных и взаимно несовместимых поведений слов, определенных VOCABULARY. Forth 83 предпринял шаг в направлении "спецификации порядка поиска времени-выполнения", опускаясь к определению специфических отношений между иерархией откомпилированных словарей и порядком поиска времени-выполнения. Forth 83 также определил экспериментальный механизм для спецификации порядка поиска времени-выполнения, ALSO/ONLY схема. ALSO/ONLY были реализованы в многочисленных системах, и достигли некоторой меры популярности в семействе Forth.

Однако, несколько продавцов отказываются реализовать это, цитируя технические ограничения. В попытке адресовать эти ограничения и таким образом надеясь сделать ALSO/ONLY более приемлемыми для этой критики, комитет определил простой "примитивный набор слов", который не только устраняет некоторые из возражений по ALSO/ONLY, но также и предусматривает достаточную возможность реализации ALSO/ONLY и всех других наборов слов порядка поиска, которые в настоящее время популярны.

Forth 83 ALSO/ONLY набор слов предусмотрен как дополнительное расширение к набору слов порядка поиска. Это предоставляет разработчикам, которые так склонны предоставлять этот набор слов, четкое стандартное поведение, но не заставляет разработчиков так делать. Некоторые продавцы публично заявили, что они не будут реализовывать ALSO/ONLY, независимо ни от чего, и один главный продавец заявил нежелание реализовать ANS Forth вообще, если ALSO/ONLY получит мандат. Комитет чувствует, что его действия благоразумны, определяя ALSO/ONLY по возможности без наказа включения его во все системы, и также предусматривая примитивный набор слов порядка поиска, который продавцы может быть, более вероятно реализуют, и который может использоваться для синтеза ALSO/ONLY.

Переход/Преобразование: Так как Forth 83 не указывал точную семантику для VOCABULARY, существующие Forth-83 Стандартные программы не могут использовать это кроме как тривиальным способом. Программы могут объявить зависимость от существования набора слов Порядка Поиска, и могут реализовать любую семантику, требующуюся для использования этих примитивов набора слов. Forth 83 программы которые требуют ALSO/ONLY могут объявлять зависимость от Расширения набора слов Порядка Поиска, или могут реализовать расширения в терминах набора слов Порядка Поиска непосредственно.

D.6.11 Воздействие мультипрограммирования

Forth 83 отметил слова с "воздействием мультипрограммирования" символом "M" в первых строках их описаний. ANS Forth удалил обозначение "M" из описания слова, перемещая обсуждение воздействия мультипрограммирования в это ненормативное приложение.

Затрагиваемые слова: Ни одно

Причина: Значение "воздействие мультипрограммирования" является точным только в контексте специфической модели для мультипрограммирования. Хотя много систем Forth предусматривают свойства мультипрограммирования, использующие специфическую карусель, кооперативность, модель совместного использования блочных буферов, эта модель - не универсальна. Даже принимая классическую модель, "M" обозначения не содержали достаточно информации, чтобы дать возможность написания приложений которые взаимодействуют в мультипрограммной системе.

Практически говоря, "M" обозначения в Forth 83 служат для документирования правил использования адресов блочных буферов в мультипрограммных системах. Эти адреса, часто становятся бессмысленными после того, как задача оставляет CPU по какой либо причине, наиболее часто с целью выполнения ввода-вывода, ожидая события, или добровольного деления ресурсов CPU используя слово PAUSE. Было существенно, что переносимые приложения уважают эти правила использования, делая возможным выполнение их на мультипрограммных системах; отказ твердо придерживаться правил, мог легко поставить под угрозу целостность других приложений, выполняющихся на этих системах так же как приложений фактически с ошибкой. Таким образом, "M" появилось на всех словах, которые созданы, чтобы покинуть CPU, с пониманием, что другие слова НИКОГДА не оставляют его.

Эти правила использования были явно зарегистрированы в Блочном наборе слов, где они уместны. "M" обозначения были полностью удалены.

Воздействие: Практически, нет.
В смысле что любое приложение, которое зависит от мультипрограммирования, должно состоять из, по крайней мере, двух задач, которые совместно используют некоторый ресурс и связываются между собой, Forth 83 не содержал достаточно информации, чтобы дать возможность написания стандартной программы, которая ЗАВИСИЛА от мультипрограммирования. Это - также истинно для ANS Forth.

Не мультипрограммные приложения в Forth 83 требовались для признания правил использования для BLOCK так, чтобы они могли быть выполнены должным образом на мультипрограммных системах. Тот же самое истинно для ANS Forth.

Единственное различие - метод документирования, используемый для определения правил использования BLOCK. Технический Комитет верит, что текущий метод более ясен, чем концепция "воздействия мультипрограммирования".

Переход/Преобразование: Нет необходимости.

D.6.12 Слова, не представленные в исполняемой форме

ANS Forth позволяет реализациям поставлять некоторые слова в исходном коде или форме "загрузки при необходимости", скорее чем требует, чтобы все снабжаемые слова были доступны без дополнительных действий программиста.

Затрагиваемые слова: Все

Причина: Системы Forth часто используются в средах, где пространство памяти

пользуется большим спросом. Каждое слово, включенное в систему в выполнимой форме, потребляет пространство памяти. Комитет верит, что разрешение стандартным словам быть предоставленными в исходной форме, увеличит вероятность того, что разработчики предоставят законченные ANS Forth реализации даже в системах, разработанных для использования в ограниченных средах.

Воздействие: Чтобы использовать Стандартную Программу с заданной ANS Forth реализацией, может быть необходимо предшествующее программе зависящее от реализации "предисловие", чтобы делать слова "в исходной форме" выполняемыми. Это подобно методам, как другие машинные языки требуют выбрать библиотечные подпрограммы, необходимые для специфического приложения.

На языках подобных C, цель устранения ненужных подпрограмм из отображения памяти приложения обычно выполняется предоставлением библиотек подпрограмм, и использованием программы "компоновщика", чтобы включить только необходимые подпрограммы в выполнимое приложение. Метод вызова и управления компоновщиком - вне контекста определения языка.

Переход/Преобразование: Перед компилированием программы, программист может быть должен выполнить некоторое действие, чтобы сделать слова, требуемые этой программе, доступными для выполнения.

E. Руководство по переносимости ANS Forth (информационное приложение)

E.1 Введение

Наиболее популярные архитектуры, используемые для реализации Forth, имели память с байтовой адресацией, 16-бит операции, и представление числа с дополнительным кодом (дополнение до двух). Forth-83 Стандарт диктует, что эти специфические особенности должны присутствовать в Forth-83 Стандартной системе, и что Forth-83 программы могут эксплуатировать эти возможности свободно.

Однако есть много животных в архитектурных джунглях, которые являются бит адресуемыми или ячейкой адресуемыми, или предпочитают 32-бит операции, или представляют числа в обратном коде (дополнение до единицы). Так как одна из сильных сторон Forth - это его пригодность в "странных" средах на "необычных" аппаратных средствах со "специфическими" возможностями, важно, что Стандартный Forth также работает на этих машинах.

Первичная цель ANS Forth Стандарта состоит в том, чтобы увеличить типы машин, которые могут поддерживать Стандартный Forth. Это выполнено, допущением некоторых ключевых терминов Forth быть определенными реализацией (например, насколько большая - ячейка?) и, предусмотрением Forth операторов (слов) которые скрывают реализацию. Это освобождает разработчика от необходимости создавать систему Forth, которая наиболее эффективно использует родные аппаратные средства. Машинно-независимые операторы, вместе с некоторой дисциплиной программирования, дают возможность программисту писать программы Forth, которые работают на широком разнообразии машин.

Остаток от этого Приложения предоставляет рекомендации для записи переносимых ANS Forth программ. Первый раздел описывает способы делать программы аппаратно независимыми. Это трудно для кого-то знакомого с только одной машинной архитектурой, чтобы вообразить проблемы, вызванные транспортировкой программ между несходными машинами. Поэтому даются примеры специфических архитектур с их соответствующими проблемами. Второй раздел описывает предположения о реализациях Forth, которые делают много программистов, но на которые нельзя полагаться в переносимой программе.

E.2 Аппаратные особенности

E.2.1 Абстракция данных/памяти

Данные и память - камни и строительный раствор конструкции программы. К сожалению, каждый компьютер обрабатывает данные и память различно. Стандарт ANS Forth Систем дает определения данных и памяти, которые применяются к широкому разнообразию компьютеров. Эти определения дают нам способ говорить об общих элементах данных и памяти, в то же время игнорируя подробности специфических аппаратных средств. Точно так же ANS Forth программы, которые используют данные и память способами соответствующими этим определениям, могут также игнорировать аппаратные подробности. Следующие разделы обсуждают определения и описывают, как записать программы, которые не зависят от особенностей данных/памяти различных компьютеров.

E.2.2 Определения

Три термина, определены ANS Forth - адресуемый элемент, ячейка, и символ. Адресное пространство из ANS Forth системы разделено на массив адресуемых элементов; адресуемый элемент- это самая маленькая коллекция битов, которые могут быть адресованы. Другими словами, адресуемый элемент- это число битов, охватываемых адресами addr и addr+1. Наиболее распространенные машины используют 8-бит адресуемые элементы. Такие "байт адресуемые" машины включают Intel 8086 и Motorola 68000 семейства.

Однако существуют другие размеры адресуемых элементов. Есть машины, которые являются бит адресуемыми и машины, которые являются адресуемыми 4-бит полубайтом. Есть также машины с адресуемыми элементами больше 8-бит. Например, есть несколько Forth-in-hardware компьютеров - ячейкой адресуемых.

Ячейка - фундаментальный тип данных системы Forth. Ячейка может быть целым числом одна-ячейка или адресом памяти. Параметры Forth и стеки возвратов - это стеки ячеек. Forth 83 определяет, что ячейка - 16- бит. В ANS Forth размер ячейки - это число определенных реализацией адресуемых элементов. Таким образом, ANS Forth реализованный на 16-бит микропроцессоре может использовать 16-бит ячейку, и реализация на 32-бит машине может использовать 32-бит ячейку. Также 18-бит машины, 36-бит машины, и т.д., могли поддерживать ANS Forth системы с 18 или 36-бит ячейками соответственно. Во всех этих системах, DUP делает одну и ту же вещь: оно дублирует вершину стека данных. ! (память) ведет себя соответственно также: учитывая две ячейки данных на стеке, оно сохраняет вторую ячейку по адресу ячейки памяти обозначенному верхней ячейкой.

Точно так же было обобщено определение символа, чтобы быть определенным реализацией числом адресуемых элементов (но, по крайней мере, восемь бит). Это устраняет потребность для разработчика Forth предусматривать 8-бит символы на процессорах, где это не так. Например, на 18-бит машине с 9-бит адресуемым элементом, 9-бит символ был бы наиболее удобен. С тех пор, по определению, Вы не можете адресовать что - ни будь меньшее чем адресуемый элемент, символ должен быть, по крайней мере, столь же большой как адресуемый элемент. Это приведет к большим символам на машинах с большими адресуемыми элементами. Как пример - машина с 16-бит адресуемой ячейкой, где 16-бит символ имеет больше смысла.

E.2.3 Адресация памяти

ANS Forth устраняет много проблем переносимости, используя вышеупомянутые определения. Одна из наиболее обычных проблем переносимости - это адресация последовательных ячеек в памяти. Имея адрес ячейки памяти, как Вы находите адрес следующей ячейки? В Forth 83 это просто: 2 + . Этот код предполагает, что память адресуется элементами по 8-бит (байт) и ячейка - 16-бит шириной. На машине с байтовой адресацией и с 32-бит ячейками, код для нахождения следующей ячейки был бы 4 + . Код был бы 1+ на ячейкой адресуемом процессоре, и 16 + на бит адресуемом процессоре с 16-бит ячейкой. ANS Forth предоставляет оператор следующей ячейки именуемый CELL+ , который может использоваться во всех этих

случаях. Задаваясь адресом, CELL+ корректирует адрес размером ячейки (измеряемым в адресуемых элементах). Связанная проблема - это адресация массива ячеек в произвольном порядке. Определяющее слово для создания массива ячеек, используя Forth 83 было бы:

```
: ARRAY   CREATE  2* ALLOT  DOES> SWAP 2* + ;
```

Использование 2* для масштабирования индекса массива предполагает байт адресацию и 16-бит ячейки. Как в примере выше, для различных машин были бы необходимы различные версии кода. ANS Forth предоставляет переносимый оператор масштабирования именуемый CELLS. Учитывая число n, CELLS возвращает число адресуемых элементов, необходимых для получения n ячеек. Переносимое определение массива таково:

```
: ARRAY   CREATE  CELLS ALLOT
  DOES> SWAP CELLS + ;
```


Есть также проблемы переносимости с адресацией массивов символов. В Forth 83 (и в наиболее обычных ANS Forth реализациях), размер символа будет равняться размеру адресуемого элемента. Следовательно, адреса последовательных символов в памяти могут быть найдены, используя 1+ и масштабирование индексов в символьном массиве - это пустая команда (то есть, 1 *). Однако есть случаи, где символ больше чем адресуемый элемент. Примеры включают (1) системы с маленькими адресуемыми элементами (например, бит- и полубайт- адресуемые системы), и (2) системы с большими наборами символов (например, 16-бит символы на байт адресуемой машине). CHAR+ и CHARS операторы, аналогичные CELL+ и CELLS доступны для разрешения максимальной переносимости.

ANS Forth обобщает определение некоторых слов Forth, которые работают на участках памяти с использованием адресуемых элементов. Один пример - ALLOT. Приписыванием перед ALLOT соответствующего масштабирующего оператора (CELLS, CHARS, и т.д.), может быть распределено пространство для любой желательной структуры данных (см. определение массива выше). Например:

```
CREATE ABUFFER 5 CHARS ALLOT (назначает 5 символьный буфер)
```

Слово блочного перемещения памяти также использует адресуемые элементы:

```
source destination 8 CELLS MOVE (перемещает 8 ячеек)
```

E.2.4 Проблемы выравнивания

Не все адреса созданы равными. Много процессоров имеют ограничения на адреса, которые могут использоваться командами доступа к памяти. Этот Стандарт не требует от разработчика ANS Forth делать выравнивание прозрачным; напротив, он требует (в Разделе 3.3.3.1 Выравнивание адреса), что ANS Forth программа принимает, что может требоваться выравнивание символа и ячейки.

Одна из наиболее обычных проблем, вызванных ограничениями выравнивания, состоит в создании таблиц содержащих и символы и ячейки. Когда , (запятая) или C, используется для инициализации таблицы, данные сохраняются по указателю области данных. Следовательно, они должны быть соответственно выровнены. Например, непереносимое определение таблицы было бы:

```
CREATE ATABLE   1 C,  X ,  2 C,  Y ,
```

На машине, которая ограничивает 16-бит выборки по четным адресам, CREATE оставил бы указатель области данных на четном адресе, 1 C, делало бы указатель области данных нечетным, и , (запятая) нарушила бы ограничение адреса, сохраняя X по нечетному адресу. Переносимый способ создания таблицы:

```
CREATE ATABLE   1 C,  ALIGN X ,  2 C,  ALIGN Y ,
```

ALIGN корректирует указатель области данных на первый выровненный адрес больший чем или равный его текущему адресу. Выровненный адрес подходит для сохранения или выборки символов, ячеек, пар-ячеек, или чисел две-ячейки.

После инициализации таблицы, мы также хотели бы читать значения из таблицы. Например, предположите, что мы хотим выбрать первую ячейку X , из таблицы. ATABLE CHAR+ дает адрес первой вещи после символа. Однако это не может быть адрес из X , так как мы выровняли указатель словаря между C, и , . Переносимый путь получения адреса X такой:


```
ATABLE CHAR+ ALIGNED
```

ALIGNED корректирует адрес на вершине стека к первому выровненному адресу большему чем или равному его текущему значению.

E.3 Представление числа

Различные компьютеры представляют числа различными способами. Понимание этих различий может помочь программисту избежать написания программы, которая зависит от частного представления.

E.3.1 Обратный порядок байт против прямого порядка байт

Битовые составляющие числа в памяти сохраняются в различном порядке на различных машинах. Некоторые машины помещают самую старшую часть числа по адресу в памяти, и младшие части после ней в более высоких адресах. Другие машины делают напротив - самая младшая часть сохраняется по самому меньшему адресу. Например, следующий код для 16-бит 8086 Forth "прямой порядок байт" произвел бы ответ 34 (hex):

```
VARIABLE FOO   HEX 1234 FOO !   FOO C@
```

Тот же самый код на 16-бит 68000 Forth "обратный порядок байт" произвел бы ответ 12 (hex). Переносимая программа не может использовать представление числа в памяти.

Связанная проблема - это представление в памяти пары-ячеек и чисел две-ячейки. Когда пара-ячеек перемещается со стека в память с помощью 2! , ячейка, которая была на вершине стека, помещается по меньшему адресу памяти. Это полезно и разумно для манипулирования отдельными ячейками, когда они находятся в памяти.

E.3.2 Организация ALU

Различные компьютеры используют различный набор двоичных разрядов для представления целых чисел. Варианты включают двоичные представления (дополнительный код (дополнение до двух), обратный код (дополнение до единицы), величина со знаком, и т.д.), и десятичные представления (BCD, и т.д.). Каждый из этих форматов создает преимущества и недостатки в конструкции арифметико-логического элемента компьютера (ALU). Наиболее часто используемое представление, дополнительный код (дополнение до двух), является популярным из-за простоты его алгоритмов суммирования и вычитания.

Программисты, которые выросли на машинах с дополнительным кодом (дополнение до двух), имеют тенденцию стать зависимыми от их представления чисел и брать для предоставления некоторые свойства этого представления. Например, хитрость для нахождения остатка от числа, разделенного величиной два - это маскировать некоторые биты с помощью AND. Обычное приложение этой хитрости - это проверить число на нечетность, используя 1 AND. Однако это не будет работать на машине с обратным кодом (дополнение до единицы), если число отрицательное (переносимая

методика - 2 MOD).

Остаток от этого раздела - (не исчерпывающий) список вещей для наблюдения за тем, когда желательна переносимость между машинами с двоичными представлениями отличными от дополнительного кода (дополнение до двух).

Преобразовать число одна-ячейка в число две-ячейки, ANS Forth предусматривает оператор S>D. Чтобы преобразовать число две-ячейки в одну-ячейку, программисты Forth имеют традиционное использование DROP. Однако эта уловка не работает на машинах величины со знаком. Для переносимости доступен оператор D>S. Преобразование числа одна-ячейка без знака в число две-ячейки может быть сделано переносимо, помещением нуля на стек.

E.4 Реализация Forth системы

В продолжение истории Forth, было разработано удивительное разнообразие методов реализации. ANS Forth Стандарт поощряет это разнообразие и следовательно ограничивает предположения пользователя, которые он может делать об основной реализации ANS Forth системы. Пользователи конкретной реализации Forth часто привыкают к аспектам реализации и предполагают, что они обычны для всех Forth. Этот раздел указывает многие из этих неправильных предположений.

E.4.1 Определения

Традиционно, определения Forth состояли из имени слова Forth, связи поиска в словаре, данных, описывающих, как выполнять определение, и параметров, непосредственно описывающих определение. Эти компоненты названы именем, связью, кодом, и полем параметров. Не было найдено никакого метода для доступа к этим полям, который работал бы поперек всех используемых в настоящее время реализаций Forth. Поэтому, ANS Forth строго ограничивает то, как могут использоваться поля. Определенно, переносимая ANS Forth программа не может использовать имя, связь, или поле кода произвольно. Использование поля параметров (для ясности переименованное в поле данных) ограничено операциями, описанными ниже.

Только слова, определенные CREATE или другими определяющими словами, которые вызывают CREATE, имеют поля данных. Другие определяющие слова в Стандарте (VARIABLE, CONSTANT, :, и т.д.) не могут быть реализованы с помощью CREATE. Следовательно, Стандартная Программа должна принять что слова определенные через VARIABLE, CONSTANT, :, и т.д., не могут иметь никаких полей данных. Нет никакого способа для Стандартной Программы изменить значение константы или заменить значение определения через двоеточие. DOES> часть определяющего слово работает на поле данных. Так как только CREATE слова имеют поля данных, DOES> может быть только объединено с CREATE или словом которое вызывает CREATE.

В ANS Forth, FIND, ['] и ' (tick) возвращают неопределенный объект называемый "идентификатор исполнения". Есть только несколько вещей, которые могут быть сделаны с идентификатором исполнения. Идентификатор может быть передан в EXECUTE для выполнения слова, идентификатор исполнения которого получен с помощью ' (tick) или скомпилирован в текущее определение с помощью COMPILE, . Идентификатор может также быть сохранен в переменной и использован позже. Наконец, если слово, идентификатор исполнения которого получен с помощью ' (tick), было определено через CREATE , >BODY преобразует идентификатор исполнения в адрес области данных слова.

Одна вещь, которая определенно не может быть сделана с идентификатором исполнения - это использование ! или , для записи его в объектный код определения Forth. Эта методика иногда используется в реализациях, где объектный код - это список адресов (шитый код) и идентификатор исполнения - также адрес. Однако ANS Forth разрешает реализации с чистым кодом, где это не будет работать.

E.4.2 Стеки

В некоторых реализациях Forth, возможно найти адрес стека в памяти и манипулировать стеком как массивом ячеек. Эта методика, однако, не переносимая. В некоторых системах, особенно в аппаратно реализованных системах Forth, стеки могли бы быть в части памяти, которая не может быть адресована с помощью программы или не могла бы быть в памяти вообще. Forth стеки параметров и возвратов должны обрабатываться как стеки.

Стандартная Программа может использовать стек возвратов непосредственно только для временного хранения значений. Каждое значение, полученное или удаленное со стека возвратов с использованием R@, R>, или 2R>, должно было быть положенным на стек явным использованием >R или 2>R. Даже это должно быть сделано осторожно, так как система может использовать стек возвратов для хранения адресов возврата и параметров управления циклом. Раздел 3.2.3.3 Стек возвратов Стандарта имеет список ограничений.

E.5 Дисциплины соглашения ROM приложений

Когда Стандартная Система предоставляет область данных, которая является однородно читаемой и записываемой, мы можем называть эту среду "RAM-only".

Программы, предназначенные для ROM приложений должны делить область данных, по крайней мере, на две части: записываемую и читаемую не инициализируемую часть называемую "RAM", и инициализируемую часть только для чтения называемую "ROM". Третья возможность, записываемая и читаемая инициализируемая часть, обычно называемая "инициализируемая RAM", не относится к этой дисциплине. Стандартная Программа должна явно инициализировать область данных RAM как необходимо.

Разделение области данных в RAM и ROM значимо только в течение жизни ROM программы. Если ROM программа - стандартно разработанная система, она имеет такую же самую систематику как обычная RAM-only система.

Слова, затрагиваемые преобразованием из RAM-only в смешанную RAM и ROM среду:

```
, (comma)  ALIGN  ALIGNED  ALLOT  C,  CREATE  HERE  UNUSED
```

(VARIABLE всегда обращается к области данных RAM.)

За исключением , (запятая) и C, эти слова значимы в ROM и RAM областях данных.

Для выбора области данных, этим словам могли бы предшествовать селекторы RAM и ROM. Например:

```
ROM  CREATE ONES  32 ALLOT  ONES 32 1 FILL  RAM
```

создало бы таблицу в области данных ROM. Сохранение данных в RAM области данных при производстве программы для ROM было бы неопределенной ситуацией.

Прямая реализация этих селекторов поддерживала бы отдельные счетчики адреса для каждой области. Значение счетчика было бы возвращено HERE и изменено , (запятая), C, , ALIGN, и ALLOT, в RAM и ROM, просто выбором соответствующего счетчика адреса. Эта методика могла бы быть расширена на дополнительное разделение области данных.

E.6 Резюме

ANS Forth, стандарт не может и не должен вынуждать кого либо записывать переносимую программу. В ситуации, где первостепенна производительность, программист, поощряется использовать каждую уловку в книге. С другой стороны, если необходима переносимость на широкое разнообразие систем, ANS Forth предусматривает для выполнения этого инструментальные средства. Вероятно, нет никакой такой вещи как полностью переносная программа. Программист, используя это руководство, должен разумно взвешивать компромиссы обеспечения переносимости на специфические машины. Например, машины использующие числа величин со знаком редки и вероятно не заслуживают, многих размышлений. Но, системы с различными размерами ячеек будут, конечно, встречаться и должны быть предусмотрены. В общем, создание переносимой программы проясняет, и процесс размышления программиста и заключительную программу.

F. Алфавитный список слов (информационное приложение)

В следующем списке, последняя, четырехразрядная, часть номера ссылки устанавливает последовательность соответствующую алфавитному порядку всех стандартных слов. Первые две или три части указывают набор слов и раздел словаря, в котором слово определено.

```
  .6.1.0010   !  ..................."store".......................CORE......41
  .6.1.0030   #  ..................."number-sign".................CORE......41
  .6.1.0040   #>  .................."number-sign-greater".........CORE......41
  .6.1.0050   #S  .................."number-sign-s"...............CORE......41
  .6.2.0060   #TIB  ................."number-t-i-b"............CORE EXT......64
  .6.1.0070   '  ..................."tick"........................CORE......41
  .6.1.0080   (  ..................."paren"......................CORE......41
11.6.1.0080   (  ..................."paren"......................FILE......97
13.6.1.0086   (LOCAL)  ............."paren-local-paren"..........LOCAL.....122
  .6.1.0090   *  ..................."star"........................CORE......42
  .6.1.0100   */ ..................."star-slash".................CORE......42
  .6.1.0110   */MOD  ..............."star-slash-mod"..............CORE......42
  .6.1.0120   +  ..................."plus"........................CORE......42
  .6.1.0130   +!  .................."plus-store".................CORE......42
  .6.1.0140   +LOOP  ..............."plus-loop"..................CORE......42
  .6.1.0150   ,  ..................."comma"......................CORE......43
  .6.1.0160   -  ..................."minus"......................CORE......43
17.6.1.0170   -TRAILING  ..........."dash-trailing"............STRING.....140
  .6.1.0180   .  ..................."dot"........................CORE......43
  .6.1.0190   ."  .................."dot-quote".................CORE......43
  .6.2.0200   .(  .................."dot-paren"..............CORE EXT......64
  .6.2.0210   .R  .................."dot-r"..................CORE EXT......64
15.6.1.0220   .S  .................."dot-s"....................TOOLS.....129
  .6.1.0230   /  ..................."slash"......................CORE......44
  .6.1.0240   /MOD  ................."slash-mod"..................CORE......44
17.6.1.0245   /STRING  ............."slash-string"............STRING.....140
  .6.1.0250   0<  .................."zero-less".................CORE......44
  .6.2.0260   0<>  ................."zero-not-equals".........CORE EXT......64
  .6.1.0270   0=  .................."zero-equals"...............CORE......44
  .6.2.0280   0>  .................."zero-greater"............CORE EXT......64
  .6.1.0290   1+  .................."one-plus"..................CORE......44
  .6.1.0300   1-  .................."one-minus".................CORE......44
  .6.1.0310   2!  .................."two-store".................CORE......44
  .6.1.0320   2*  .................."two-star"..................CORE......44
  .6.1.0330   2/  .................."two-slash".................CORE......44
  .6.2.0340   2>R  ................."two-to-r"..............CORE EXT......64
  .6.1.0350   2@  .................."two-fetch".................CORE......45
 8.6.1.0360   2CONSTANT  ..........."two-constant"............DOUBLE......80
  .6.1.0370   2DROP  ............."two-drop"..................CORE......45
  .6.1.0380   2DUP  ............."two-dupe"..................CORE......45
 8.6.1.0390   2LITERAL  ..........."two-literal"............DOUBLE......80
```

```
  .6.1.0890   CELLS  ..........................................CORE......50
  .6.1.0895   CHAR  ..............."char".......................CORE......50
  .6.1.0897   CHAR+ ..............."char-plus"..................CORE......50
  .6.1.0898   CHARS  ..............."chars"......................CORE......50
11.6.1.0900   CLOSE-FILE  ......................................FILE......97
17.6.1.0910   CMOVE  ..............."c-move"....................STRING.....140
17.6.1.0920   CMOVE>  .............."c-move-up".................STRING.....140
15.6.2.0930   CODE  ........................................TOOLS EXT.....130
17.6.1.0935   COMPARE  .....................................STRING.....140
  .6.2.0945   COMPILE,  ............"compile-comma"............CORE EXT......66
  .6.1.0950   CONSTANT  ....................................CORE......50
  .6.2.0970   CONVERT  ...................................CORE EXT......67
  .6.1.0980   COUNT  ......................................CORE......50
  .6.1.0990   CR  ................."c-r"......................CORE......50
  .6.1.1000   CREATE  .....................................CORE......50
11.6.1.1010   CREATE-FILE  ....................................FILE......97
15.6.2.1015   CS-PICK  ............."c-s-pick"................TOOLS EXT.....131
15.6.2.1020   CS-ROLL  ............."c-s-roll"................TOOLS EXT.....131
 8.6.1.1040   D+  ................."d-plus"....................DOUBLE......81
 8.6.1.1050   D-  ................."d-minus"...................DOUBLE......81
 8.6.1.1060   D.  ................."d-dot"....................DOUBLE......81
 8.6.1.1070   D.R  ................"d-dot-r"...................DOUBLE......81
 8.6.1.1075   D0<  ................"d-zero-less"..............DOUBLE......81
 8.6.1.1080   D0=  ................"d-zero-equals"............DOUBLE......81
 8.6.1.1090   D2*  ................"d-two-star"..............DOUBLE......81
 8.6.1.1100   D2/  ................"d-two-slash"..............DOUBLE......82
 8.6.1.1110   D<  .................."d-less-than"..............DOUBLE......82
 8.6.1.1120   D=  .................."d-equals"................DOUBLE......82
12.6.1.1130   D>F  ................."d-to-f"..................FLOATING.....109
 8.6.1.1140   D>S  ................."d-to-s"..................DOUBLE......82
 8.6.1.1160   DABS  ..............."d-abs"....................DOUBLE......82
  .6.1.1170   DECIMAL  ...................................CORE......51
16.6.1.1180   DEFINITIONS  ...................................SEARCH.....136
11.6.1.1190   DELETE-FILE  ....................................FILE......97
  .6.1.1200   DEPTH  .....................................CORE......51
12.6.2.1203   DF!  ................"d-f-store"............FLOATING EXT.....112
12.6.2.1204   DF@  ................"d-f-fetch"............FLOATING EXT.....113
12.6.2.1205   DFALIGN  ............."d-f-align"............FLOATING EXT.....113
12.6.2.1207   DFALIGNED  ..........."d-f-aligned"..........FLOATING EXT.....113
12.6.2.1208   DFLOAT+  ............"d-float-plus".........FLOATING EXT.....113
12.6.2.1209   DFLOATS  ............."d-floats"............FLOATING EXT.....113
 8.6.1.1210   DMAX  ..............."d-max"....................DOUBLE......82
 8.6.1.1220   DMIN  ..............."d-min"....................DOUBLE......82
 8.6.1.1230   DNEGATE  ............"d-negate"..................DOUBLE......82
  .6.1.1240   DO  ........................................CORE......51
  .6.1.1250   DOES>  .............."does"......................CORE......51
  .6.1.1260   DROP  ......................................CORE......52
 8.6.2.1270   DU<  ................"d-u-less"..............DOUBLE EXT......83
15.6.1.1280   DUMP  ..........................................TOOLS.....129
  .6.1.1290   DUP  ................"dupe"......................CORE......52
```



```
15.6.2.1300   EDITOR  ........................................TOOLS EXT.....132
10.6.2.1305   EKEY  ..............."e-key"..............FACILITY EXT......91
10.6.2.1306   EKEY>CHAR  ..........."e-key-to-char"........FACILITY EXT......91
10.6.2.1307   EKEY?  ..............."e-key-question".......FACILITY EXT......91
  .6.1.1310   ELSE  ......................................CORE......52
  .6.1.1320   EMIT  ......................................CORE......52
10.6.2.1325   EMIT?  ..............."emit-question"........FACILITY EXT......91
 7.6.2.1330   EMPTY-BUFFERS  ..............................BLOCK EXT......77
  .6.2.1342   ENDCASE  ............."end-case"..............CORE EXT......67
  .6.2.1343   ENDOF  ..............."end-of"................CORE EXT......67
```

```
  .6.2.2298   TRUE  ..........................................CORE EXT......71
  .6.2.2300   TUCK  .........................................CORE EXT......71
  .6.1.2310   TYPE  ...............................................CORE......60
  .6.1.2320   U.  ..................."u-dot".....................CORE......60
  .6.2.2330   U.R  .................."u-dot-r"..................CORE EXT......71
  .6.1.2340   U<  ..................."u-less-than"..............CORE......60
  .6.2.2350   U>  ..................."u-greater-than"..........CORE EXT......71
  .6.1.2360   UM*  .................."u-m-star"..................CORE......60
  .6.1.2370   UM/MOD  ..............."u-m-slash-mod"............CORE......61
  .6.1.2380   UNLOOP  ..............................................CORE......61
  .6.1.2390   UNTIL  ...............................................CORE......61
  .6.2.2395   UNUSED  ..........................................CORE EXT......71
 7.6.1.2400   UPDATE  ..............................................BLOCK......77
  .6.2.2405   VALUE  ...........................................CORE EXT......72

                                                                        250


  .6.1.2410   VARIABLE  ...........................................CORE......61
11.6.1.2425   W/O  ................."w-o"........................FILE.....101
  .6.1.2430   WHILE  ...............................................CORE......62
  .6.2.2440   WITHIN  ..........................................CORE EXT......72
  .6.1.2450   WORD  ................................................CORE......62
16.6.1.2460   WORDLIST  .........................................SEARCH.....137
15.6.1.2465   WORDS  ..............................................TOOLS.....129
11.6.1.2480   WRITE-FILE  .........................................FILE.....101
11.6.1.2485   WRITE-LINE  .........................................FILE.....102
  .6.1.2490   XOR  ................."x-or"........................CORE......62
  .6.1.2500   [  ..................."left-bracket"..............CORE......62
  .6.1.2510   [']  ................."bracket-tick"..............CORE......62
  .6.1.2520   [CHAR]  ............."bracket-char"..............CORE......63
  .6.2.2530   [COMPILE]  ..........."bracket-compile"........CORE EXT......72
15.6.2.2531   [ELSE]  ............."bracket-else"...........TOOLS EXT.....132
15.6.2.2532   [IF]  ..............."bracket-if"............TOOLS EXT.....132
15.6.2.2533   [THEN]  ............."bracket-then"...........TOOLS EXT.....133
  .6.2.2535   \  ..................."backslash"..............CORE EXT......72
 7.6.2.2535   \  ..................."backslash"............BLOCK EXT......78
  .6.1.2540   ]  ..................."right-bracket"..............CORE......63

                                                                        251
```