1. Introduction

1.1 Purpose

The purpose of this Standard is to promote the portability of Forth programs for use on a wide variety of computing systems, to facilitate the communication of programs, programming techniques, and ideas among Forth programmers, and to serve as a basis for the future evolution of the Forth language.

1.2 Scope

This Standard specifies an interface between a Forth System and a Forth Program by defining the words provided by a Standard System.

1.2.1 Inclusions

This Standard specifies:

- the forms that a program written in the Forth language may take;
- the rules for interpreting the meaning of a program and its data.

1.2.2 Exclusions

This Standard does not specify:

- the mechanism by which programs are transformed for use on computing systems;
- the operations required for setup and control of the use of programs on computing systems;
- the method of transcription of programs or their input or output data to or from a storage medium;
- the program and Forth system behavior when the rules of this Standard fail to establish an interpretation;
- the size or complexity of a program and its data that will exceed the capacity of any specific computing system or the capability of a particular Forth system;
- the physical properties of input/output records, files, and units;
- the physical properties and implementation of storage.

1.3 Document organization

1.3.1 Word sets

This Standard groups Forth words and capabilities into *word sets* under a name indicating some shared aspect, typically their common functional area. Each word set may have an extension, containing words that offer additional functionality. These words are not required in an implementation of the word set.

The “Core” word set, defined in sections 1 through 6, contains the required words and capabilities of a Standard System. The other word sets, defined in sections 7 through 17, are optional, making it possible to provide Standard Systems with tailored levels of functionality.

1.3.1.1 Text sections

Within each word set, section 1 contains introductory and explanatory material and section 2 introduces terms and notation used throughout the Standard. There are no requirements in these sections.

Sections 3 and 4 contain the usage and documentation requirements, respectively, for Standard Systems and Programs, while section 5 specifies their labeling.

1.3.1.2 Glossary sections

Section 6 of each word set specifies the required behavior of the definitions in the word set and the extensions word set.

1.3.2 Annexes

The annexes do not contain any required material.

Annex A provides some of the rationale behind the committee's decisions in creating this Standard, as well as implementation examples. It has the same section numbering as the body of the Standard to make it easy to relate each requirements section to its rationale section.

Annex B is a short bibliography on Forth.

Annex C provides an introduction to Forth.

Annex D discusses the compatibility of ANS Forth with earlier Forths, emphasizing the differences from Forth 83.

Annex E presents some techniques for writing portable programs in ANS Forth.

Annex F includes the words from all word sets in a single list, and serves as an index of ANS Forth words.

1.4 Future directions

1.4.1 New technology

This Standard adopts certain words and practices that are increasingly found in common practice. New words have also been adopted to ease creation of portable programs.

1.4.2 Obsolescent features

This Standard adopts certain words and practices that cause some previously used words to become obsolescent. Although retained here because of their widespread use, their use in new implementations or new programs is discouraged, because they may be withdrawn from future revisions of the Standard.

This Standard designates the following words as obsolescent:

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. Terms and notation

The phrase “See:” is used throughout this Standard to direct the reader to other sections of the Standard that have a direct bearing on the current section.

In this Standard, “shall” states a requirement on a system or program; conversely, “shall not” is a prohibition; “need not” means “is not required to”; “should” describes a recommendation of the Standard; and “may”, depending on context, means “is allowed to” or “might happen”.

Throughout the Standard, typefaces are used in the following manner:

– This proportional serif typeface is used for text, with *italic* used for symbols and the first appearance of new terms;
– A bold proportional sans-serif typeface is used for **headings**;
– A bold monospaced serif typeface is used for **`Forth-language`** text.

2.1 Definitions of terms

Terms defined in this section are used generally throughout this Standard. Additional terms specific to individual word sets are defined in those word sets. Other terms are defined at their first appearance, indicated by italic type. Terms not defined in this Standard are to be construed according to the *Dictionary for Information Systems*, ANSI X3.172-1990.

address unit: depending on context, either 1) the units into which a Forth address space is divided for the purposes of locating data objects such as characters and variables; 2) the physical memory storage elements corresponding to those units; 3) the contents of such a memory storage element; or 4) the units in which the length of a region of memory is expressed.

aligned address: the address of a memory location at which a character, cell, cell pair, or double-cell integer can be accessed.

ambiguous condition: a circumstance for which this Standard does not prescribe a specific behavior for Forth systems and programs.

Ambiguous conditions include such things as the absence of a needed delimiter while parsing, attempted access to a nonexistent file, or attempted use of a nonexistent word. An ambiguous condition also exists when a Standard word is passed values that are improper or out of range.

cell: the primary unit of information in the architecture of a Forth system.

cell pair: two cells that are treated as a single unit.

character: depending on context, either 1) a storage unit capable of holding a character; or 2) a member of a character set.

character-aligned address: the address of a memory location at which a character can be accessed.

character string: data space that is associated with a sequence of consecutive character-aligned addresses. Character strings usually contain text. Unless otherwise indicated, the term “string” means “character string.”

code space: the logical area of the dictionary in which word semantics are implemented.

compile: to transform source code into dictionary definitions.

compilation semantics: the behavior of a Forth definition when its name is encountered by the text interpreter in compilation state.

counted string: a data structure consisting of one character containing a length followed by zero or more contiguous data characters. Normally, counted strings contain text.

cross compiler: a system that compiles a program for later execution in an environment that may be physically and logically different from the compiling environment. In a cross compiler, the term "host" applies to the compiling environment, and the term "target" applies to the run-time environment.

current definition: the definition whose compilation has been started but not yet ended.

data field: the data space associated with a word defined via **CREATE**.

data space: the logical area of the dictionary that can be accessed.

data-space pointer: the address of the next available data space location, i.e., the value returned by **HERE**.

data stack: a stack that may be used for passing parameters between definitions. When there is no possibility of confusion, the data stack is referred to as "the stack". Contrast with **return stack**.

data type: an identifier for the set of values that a data object may have.

defining word: a Forth word that creates a new definition when executed.

definition: a Forth execution procedure compiled into the dictionary.

dictionary: an extensible structure that contains definitions and associated data space.

display: to send one or more characters to the user output device.

environmental dependencies: a program's implicit assumptions about a Forth system's implementation options or underlying hardware. For example, a program that assumes a cell size greater than 16 bits is said to have an environmental dependency.

execution semantics: the behavior of a Forth definition when it is executed.

execution token: a value that identifies the execution semantics of a definition.

find: to search the dictionary for a definition name matching a given string.

immediate word: a Forth word whose compilation semantics are to perform its execution semantics.

implementation defined: denotes system behaviors or features that must be provided and documented by a system but whose further details are not prescribed by this Standard.

implementation dependent: denotes system behaviors or features that must be provided by a system but whose further details are not prescribed by this Standard.

input buffer: a region of memory containing the sequence of characters from the input source that is currently accessible to a program.

input source: the device, file, block, or other entity that supplies characters to refill the input buffer.

input source specification: a set of information describing a particular state of the input source, input buffer, and parse area. This information is sufficient, when saved and restored properly, to enable the nesting of parsing operations on the same or different input sources.

interpretation semantics: the behavior of a Forth definition when its name is encountered by the text interpreter in interpretation state.

keyboard event: a value received by the system denoting a user action at the user input device. The term "keyboard" in this document does not exclude other types of user input devices.

line: a sequence of characters followed by an actual or implied line terminator.

name space: the logical area of the dictionary in which definition names are stored.

number: in this Standard, "number" used without other qualification means "integer." Similarly, "double number" means "double-cell integer."

parse: to select and exclude a character string from the parse area using a specified set of delimiting characters, called delimiters.

parse area: the portion of the input buffer that has not yet been parsed, and is thus available to the system for subsequent processing by the text interpreter and other parsing operations.

pictured-numeric output: a number display format in which the number is converted using Forth words that resemble a symbolic "picture" of the desired output.

program: a complete specification of execution to achieve a specific function (application task) expressed in Forth source code form.

receive: to obtain characters from the user input device.

return stack: a stack that may be used for program execution nesting, do-loop execution, temporary storage, and other purposes.

standard word: a named Forth procedure, formally specified in this Standard.

user input device: the input device currently selected as the source of received data, typically a keyboard.

user output device: the output device currently selected as the destination of display data.

variable: a named region of data space located and accessed by its memory address.

word: depending on context, either 1) the name of a Forth definition; or 2) a parsed sequence of non-space characters, which could be the name of a Forth definition.

word list: a list of associated Forth definition names that may be examined during a dictionary search.

word set: a set of Forth definitions grouped together in this Standard under a name indicating some shared aspect, typically their common functional area.

2.2 Notation

2.2.1 Numeric notation

Unless otherwise stated, all references to numbers apply to signed single-cell integers. The inclusive range of values is shown as {*from...to*}. The allowable range for the contents of an address is shown in double braces, particularly for the contents of variables, e.g., **BASE** {{2...36}}.

2.2.2 Stack notation

Stack parameters input to and output from a definition are described using the notation:

(*stack-id before* **--** *after*)

where *stack-id* specifies which stack is being described, *before* represents the stack-parameter data types before execution of the definition and *after* represents them after execution. The symbols used in *before* and *after* are shown in table 3.2.

The control-flow-stack *stack-id* is "C:", the data-stack *stack-id* is "S:", and the return-stack *stack-id* is "R:". When there is no confusion, the data-stack *stack-id* may be omitted.

When there are alternate *after* representations, they are described by "$after_1$ | $after_2$". The top of the stack is stack is to the right. Only those stack items required for or provided by execution of the definition are shown.

2.2.3 Parsed-text notation

If, in addition to using stack parameters, a definition parses text, that text is specified by an abbreviation from table 2.1 shown surrounded by double-quotes and placed between the *before* parameters and the "--" separator in the first stack described, e.g.,

(S: *before* "*parsed-text-abbreviation*" -- *after*).

Table 2.1 – Parsed text abbreviations

Abbreviation	Description
<char>	the delimiting character marking the end of the string being parsed
<chars>	zero or more consecutive occurrences of the character *char*
<space>	a delimiting space character
<spaces>	zero or more consecutive occurrences of the character *space*
<quote>	a delimiting double quote
<paren>	a delimiting right parenthesis
<eol>	an implied delimiter marking the end of a line
ccc	a parsed sequence of arbitrary characters, excluding the delimiter character
name	a token delimited by space, equivalent to *ccc<space>* or *ccc<eol>*

2.2.4 Glossary notation

The glossary entries for each word set are listed in the standard ASCII collating sequence. Each glossary entry specifies an ANS Forth word and consists of two parts: an *index line* and the *semantic description* of the definition.

2.2.4.1 Glossary index line

The index line is a single-line entry containing, from left to right:

– Section number, the last four digits of which assign a unique sequential number to all words included in this Standard;
– **DEFINITION-NAME** in upper-case, mono-spaced, bold-face letters;
– Natural-language pronunciation in quotes if it differs from English;
– Word-set designator from table 2.2. The designation for extensions word sets includes "EXT".

Table 2.2 – Word set designators

Word set	Designator
Core word set	CORE
Block word set	BLOCK
Double-Number word set	DOUBLE
Exception word set	EXCEPTION
Facility word set	FACILITY
File-Access word set	FILE
Floating-Point word set	FLOATING
Locals word set	LOCALS
Memory-Allocation word set	MEMORY
Programming-Tools word set	TOOLS
Search-Order word set	SEARCH
String-Handling word set	STRING

2.2.4.2 Glossary semantic description

The first paragraph of the semantic description contains a stack notation for each stack affected by execution of the word. The remaining paragraphs contain a text description of the semantics. See **3.4.3 Semantics**.

3. Usage requirements

A system shall provide all of the words defined in **6.1 Core Words**. It may also provide any words defined in the optional word sets and extensions word sets. No standard word provided by a system shall alter the system state in a way that changes the effect of execution of any other standard word except as provided in this Standard. A system may contain non-standard extensions, provided that they are consistent with the requirements of this Standard.

The implementation of a system may use words and techniques outside the scope of this Standard.

A system need not provide all words in executable form. The implementation may provide definitions, including definitions of words in the Core word set, in source form only. If so, the mechanism for adding the definitions to the dictionary is implementation defined.

A program that requires a system to provide words or techniques not defined in this Standard has an environmental dependency.

3.1 Data types

A data type identifies the set of permissible values for a data object. It is not a property of a particular storage location or position on a stack. Moving a data object shall not affect its type.

No data-type checking is required of a system. An ambiguous condition exists if an incorrectly typed data object is encountered.

Table 3.1 summarizes the data types used throughout this Standard. Multiple instances of the same type in the description of a definition are suffixed with a sequence digit subscript to distinguish them.

Table 3.1 – Data types

Symbol	Data type	Size on stack
flag	flag	1 cell
true	true flag	1 cell
false	false flag	1 cell
char	character	1 cell
n	signed number	1 cell
+n	non-negative number	1 cell
u	unsigned number	1 cell
n|*u*[1]	number	1 cell
x	unspecified cell	1 cell
xt	execution token	1 cell
addr	address	1 cell
a-addr	aligned address	1 cell
c-addr	character-aligned address	1 cell
d	double-cell signed number	2 cells
+d	double-cell non-negative number	2 cells
ud	double-cell unsigned number	2 cells
d|*ud*[2]	double-cell number	2 cells
xd	unspecified cell pair	2 cells
colon-sys	definition compilation	implementation dependent
do-sys	do-loop structures	implementation dependent
case-sys	**CASE** structures	implementation dependent
of-sys	**OF** structures	implementation dependent
orig	control-flow origins	implementation dependent
dest	control-flow destinations	implementation dependent
loop-sys	loop-control parameters	implementation dependent

nest-sys	definition calls	implementation dependent
*i*x, j*x*[3]	any data type	0 or more cells

[1] May be either a signed number or an unsigned number depending on context.

[2] May be either a double-cell signed number or a double-cell unsigned number depending on context.

[3] May be an undetermined number of stack entries of unspecified type. For examples of use, see **6.1.1370 `EXECUTE`**, **6.1.2050 `QUIT`**.

3.1.1 Data-type relationships

Some of the data types are subtypes of other data types. A data type ***i*** is a subtype of type ***j*** if and only if the members of ***i*** are a subset of the members of ***j***. The following list represents the subtype relationships using the phrase "***i*** => ***j***" to denote "***i*** is a subtype of ***j***." The subtype relationship is transitive; if ***i*** => ***j*** and ***j*** => ***k*** then ***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*.

Any Forth definition that accepts an argument of type ***i*** shall also accept an argument that is a subtype of ***i***.

3.1.2 Character types

Characters shall be at least one address unit wide, contain at least eight bits, and have a size less than or equal to cell size.

The characters provided by a system shall include the graphic characters {32..126}, which represent graphic forms as shown in table 3.2.

3.1.2.1 Graphic characters

A graphic character is one that is normally displayed (e.g., A, #, &, 6). These values and graphics, shown in table 3.2, are taken directly from ANS X3.4-1974 (ASCII) and ISO-646-1983, International Reference Version (IRV). The graphic forms of characters outside the hex range {20..7E} are implementation-defined. Programs that use the graphic hex 24 (the currency sign) have an environmental dependency.

The graphic representation of characters is not restricted to particular type fonts or styles. The graphics here are examples.

Table 3.2 – Standard graphic characters

Hex	IRV	ASCII	Hex	IRV	ASCII	Hex	IRV	ASCII	Hex	IRV	ASCII	Hex	IRV	ASCII	Hex	IRV	ASCII
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			

3.1.2.2 Control characters

All non-graphic characters included in the implementation-defined character set are defined in this Standard as control characters. In particular, the characters {0..31}, which could be included in the implementation-defined character set, are control characters.

Programs that require the ability to send or receive control characters have an environmental dependency.

3.1.3 Single-cell types

The implementation-defined fixed size of a cell is specified in address units and the corresponding number of bits. See **E.2 Hardware peculiarities**.

Cells shall be at least one address unit wide and contain at least sixteen bits. The size of a cell shall be an integral multiple of the size of a character. Data-stack elements, return-stack elements, addresses, execution tokens, flags, and integers are one cell wide.

3.1.3.1 Flags

Flags may have one of two logical states, *true* or *false*. Programs that use flags as arithmetic operands have an environmental dependency.

A true flag returned by a standard word shall be a single-cell value with all bits set. A false flag returned by a standard word shall be a single-cell value with all bits clear.

3.1.3.2 Integers

The implementation-defined range of signed integers shall include {-32767..+32767}.

The implementation-defined range of non-negative integers shall include {0..32767}.

The implementation-defined range of unsigned integers shall include {0..65535}.

3.1.3.3 Addresses

An address identifies a location in data space with a size of one address unit, which a program may fetch from or store into except for the restrictions established in this Standard. The size of an address unit is specified in bits. Each distinct address value identifies exactly one such storage element. See **3.3.3 Data space**.

The set of character-aligned addresses, addresses at which a character can be accessed, is an implementation-defined subset of all addresses. Adding the size of a character to a character-aligned address shall produce another character-aligned address.

The set of aligned addresses is an implementation-defined subset of character-aligned addresses. Adding the size of a cell to an aligned address shall produce another aligned address.

3.1.3.4 Counted strings

A counted string in memory is identified by the address (*c-addr*) of its length character.

The length character of a counted string shall contain a binary representation of the number of data characters, between zero and the implementation-defined maximum length for a counted string. The maximum length of a counted string shall be at least 255.

3.1.3.5 Execution tokens

Different definitions may have the same execution token if the definitions are equivalent.

3.1.4 Cell-pair types

A cell pair in memory consists of a sequence of two contiguous cells. The cell at the lower address is the first cell, and its address is used to identify the cell pair. Unless otherwise specified, a cell pair on a stack consists of the first cell immediately above the second cell.

3.1.4.1 Double-cell integers

On the stack, the cell containing the most significant part of a double-cell integer shall be above the cell containing the least significant part.

The implementation-defined range of double-cell signed integers shall include {-2147483647..+2147483647}.

The implementation-defined range of double-cell non-negative integers shall include {0..2147483647}.

The implementation-defined range of double-cell unsigned integers shall include {0..4294967295}. Placing the single-cell integer zero on the stack above a single-cell unsigned integer produces a double-cell unsigned integer with the same value. See **3.2.1.1 Internal number representation**.

3.1.4.2 Character strings

A string is specified by a cell pair (*c-addr u*) representing its starting address and length in characters.

3.1.5 System types

The system data types specify permitted word combinations during compilation and execution.

3.1.5.1 System-compilation types

These data types denote zero or more items on the control-flow stack (see 3.2.3.2). The possible presence of such items on the data stack means that any items already there shall be unavailable to a program until the control-flow-stack items are consumed.

The implementation-dependent data generated upon beginning to compile a definition and consumed at its close is represented by the symbol *colon-sys* throughout this Standard.

The implementation-dependent data generated upon beginning to compile a do-loop structure such as **`DO ... LOOP`** and consumed at its close is represented by the symbol *do-sys* throughout this Standard.

The implementation-dependent data generated upon beginning to compile a **`CASE ... ENDCASE`** structure and consumed at its close is represented by the symbol *case-sys* throughout this Standard.

The implementation-dependent data generated upon beginning to compile an **`OF ... ENDOF`** structure and consumed at its close is represented by the symbol *of-sys* throughout this Standard.

The implementation-dependent data generated and consumed by executing the other standard control-flow words is represented by the symbols *orig* and *dest* throughout this Standard.

3.1.5.2 System-execution types

These data types denote zero or more items on the return stack. Their possible presence means that any items already on the return stack shall be unavailable to a program until the system-execution items are consumed.

The implementation-dependent data generated upon beginning to execute a definition and consumed upon exiting it is represented by the symbol *nest-sys* throughout this Standard.

The implementation-dependent loop-control parameters used to control the execution of do-loops are represented by the symbol *loop-sys* throughout this Standard. Loop-control parameters shall be available inside the do-loop for words that use or change these parameters, words such as **I**, **J**, **LEAVE** and **UNLOOP**.

3.2 The implementation environment

3.2.1 Numbers

3.2.1.1 Internal number representation

This Standard allows one's complement, two's complement, or sign-magnitude number representations and arithmetic. Arithmetic zero is represented as the value of a single cell with all bits clear.

The representation of a number as a compiled literal or in memory is implementation dependent.

3.2.1.2 Digit conversion

Numbers shall be represented externally by using characters from the standard character set.

Conversion between the internal and external forms of a digit shall behave as follows:

The value in **BASE** is the radix for number conversion. A digit has a value ranging from zero to one less than the contents of **BASE**. The digit with the value zero corresponds to the character "0". This representation of digits proceeds through the character set to the decimal value nine corresponding to the character "9". For digits beginning with the decimal value ten the graphic characters beginning with the character "A" are used. This correspondence continues up to and including the digit with the decimal value thirty-five which is represented by the character "Z". The conversion of digits outside this range is implementation defined.

3.2.1.3 Free-field number display

Free-field number display uses the characters described in digit conversion, without leading zeros, in a field the exact size of the converted string plus a trailing space. If a number is zero, the least significant digit is not considered a leading zero. If the number is negative, a leading minus sign is displayed.

Number display may use the pictured numeric output string buffer to hold partially converted strings (see **3.3.3.6 Other transient regions**).

3.2.2 Arithmetic

3.2.2.1 Integer division

Division produces a quotient *q* and a remainder *r* by dividing operand *a* by operand *b*. Division operations return *q*, *r*, or both. The identity $b*q + r = a$ shall hold for all *a* and *b*.

When unsigned integers are divided and the remainder is not zero, *q* is the largest integer less than the true quotient.

When signed integers are divided, the remainder is not zero, and *a* and *b* have the same sign, *q* is the largest integer less than the true quotient. If only one operand is negative, whether *q* is rounded toward

negative infinity (floored division) or rounded towards zero (symmetric division) is implementation defined.

Floored division is integer division in which the remainder carries the sign of the divisor or is zero, and the quotient is rounded to its arithmetic floor. Symmetric division is integer division in which the remainder carries the sign of the dividend or is zero and the quotient is the mathematical quotient "rounded towards zero" or "truncated." Examples of each are shown in tables 3.3 and 3.4.

In cases where the operands differ in sign and the rounding direction matters, a program shall either include code generating the desired form of division, not relying on the implementation-defined default result, or have an environmental dependency on the desired rounding direction.

Table 3.3 – Floored Division Example

Dividend	Divisor	Remainder	Quotient
10	7	3	1
-10	7	4	-2
10	-7	-4	-2
-10	-7	-3	1

Table 3.4 – Symmetric Division Example

Dividend	Divisor	Remainder	Quotient
10	7	3	1
-10	7	-3	-1
10	-7	3	-1
-10	-7	-3	1

3.2.2.2 Other integer operations

In all integer arithmetic operations, both overflow and underflow shall be ignored. The value returned when either overflow or underflow occurs is implementation defined.

3.2.3 Stacks

3.2.3.1 Data stack

Objects on the data stack shall be one cell wide.

3.2.3.2 Control-flow stack

The control-flow stack is a last-in, first out list whose elements define the permissible matchings of control-flow words and the restrictions imposed on data-stack usage during the compilation of control structures.

The elements of the control-flow stack are system-compilation data types.

The control-flow stack may, but need not, physically exist in an implementation. If it does exist, it may be, but need not be, implemented using the data stack. The format of the control-flow stack is implementation defined. Since the control-flow stack may be implemented using the data stack, items placed on the data stack are unavailable to a program after items are placed on the control-flow stack and remain unavailable until the control-flow stack items are removed.

3.2.3.3 Return stack

Items on the return stack shall consist of one or more cells. A system may use the return stack in an implementation-dependent manner during the compilation of definitions, during the execution of do-loops, and for storing run-time nesting information.

A program may use the return stack for temporary storage during the execution of a definition subject to the following restrictions:

– A program shall not access values on the return stack (using **R@**, **R>**, **2R@** or **2R>**) that it did not place there using **>R** or **2>R**;
– A program shall not access from within a do-loop values placed on the return stack before the loop was entered;

- All values placed on the return stack within a do-loop shall be removed before **I**, **J**, **LOOP**, **+LOOP**, **UNLOOP**, or **LEAVE** is executed;
- All values placed on the return stack within a definition shall be removed before the definition is terminated or before **EXIT** is executed.

3.2.4 Operator terminal

See **1.2.2 Exclusions**.

3.2.4.1 User input device

The method of selecting the user input device is implementation defined.

The method of indicating the end of an input line of text is implementation defined.

3.2.4.2 User output device

The method of selecting the user output device is implementation defined.

3.2.5 Mass storage

A system need not provide any standard words for accessing mass storage. If a system provides any standard word for accessing mass storage, it shall also implement the Block word set.

3.2.6 Environmental queries

The name spaces for **ENVIRONMENT?** and definitions are disjoint. Names of definitions that are the same as **ENVIRONMENT?** strings shall not impair the operation of **ENVIRONMENT?**. Table 3.5 contains the valid input strings and corresponding returned value for inquiring about the programming environment with **ENVIRONMENT?**.

Table 3.5 – Environmental Query Strings

String	Value data type	Constant?	Meaning
/COUNTED-STRING	*n*	yes	maximum size of a counted string, in characters
/HOLD	*n*	yes	size of the pictured numeric output string buffer, in characters
/PAD	*n*	yes	size of the scratch area pointed to by **PAD**, in characters
ADDRESS-UNIT-BITS	*n*	yes	size of one address unit, in bits
CORE	*flag*	no	true if complete core word set present (i.e., not a subset as defined in **5.1.1**)
CORE-EXT	*flag*	no	true if core extensions word set present
FLOORED	*flag*	yes	true if floored division is the default
MAX-CHAR	*u*	yes	maximum value of any character in the implementation-defined character set
MAX-D	*d*	yes	largest usable signed double number
MAX-N	*n*	yes	largest usable signed integer
MAX-U	*u*	yes	largest usable unsigned integer
MAX-UD	*ud*	yes	largest usable unsigned double number
RETURN-STACK-CELLS	*n*	yes	maximum size of the return stack, in cells
STACK-CELLS	*n*	yes	maximum size of the data stack, in cells

If an environmental query (using **ENVIRONMENT?**) returns *false* (i.e., unknown) in response to a string, subsequent queries using the same string may return *true*. If a query returns *true* (i.e., known) in response

to a string, subsequent queries with the same string shall also return *true*. If a query designated as constant in the above table returns *true* and a value in response to a string, subsequent queries with the same string shall return *true* and the same value.

3.3 The Forth dictionary

Forth words are organized into a structure called the dictionary. While the form of this structure is not specified by the Standard, it can be described as consisting of three logical parts: a name space, a code space, and a data space. The logical separation of these parts does not require their physical separation.

A program shall not fetch from or store into locations outside data space. An ambiguous condition exists if a program addresses name space or code space.

3.3.1 Name space

The relationship between name space and data space is implementation dependent.

3.3.1.1 Word lists

The structure of a word list is implementation dependent. When duplicate names exist in a word list, the latest-defined duplicate shall be the one found during a search for the name.

3.3.1.2 Definition names

Definition names shall contain {1..31} characters. A system may allow or prohibit the creation of definition names containing non-standard characters.

Programs that use lower case for standard definition names or depend on the case-sensitivity properties of a system have an environmental dependency.

A program shall not create definition names containing non-graphic characters.

3.3.2 Code space

The relationship between code space and data space is implementation dependent.

3.3.3 Data space

Data space is the only logical area of the dictionary for which standard words are provided to allocate and access regions of memory. These regions are: contiguous regions, variables, text-literal regions, input buffers, and other transient regions, each of which is described in the following sections. A program may read from or write into these regions unless otherwise specified.

3.3.3.1 Address alignment

Most addresses used in ANS Forth are aligned addresses (indicated by *a-addr*) or character-aligned (indicated by *c-addr*). **ALIGNED**, **CHAR+**, and arithmetic operations can alter the alignment state of an address on the stack. **CHAR+** applied to an aligned address returns a character-aligned address that can only be used to access characters. Applying **CHAR+** to a character-aligned address produces the succeeding character-aligned address. Adding or subtracting an arbitrary number to an address can produce an unaligned address that shall not be used to fetch or store anything. The only way to find the next aligned address is with **ALIGNED**. An ambiguous condition exists when **@**, **!**, **,** (comma), **+!**, **2@**, or **2!** is used with an address that is not aligned, or when **C@**, **C!**, or **C,** is used with an address that is not character-aligned.

The definitions of **6.1.1000 CREATE** and **6.1.2410 VARIABLE** require that the definitions created by them return aligned addresses.

After definitions are compiled or the word **ALIGN** is executed the data-space pointer is guaranteed to be aligned.

3.3.3.2 Contiguous regions

A system guarantees that a region of data space allocated using **ALLOT**, **,** (comma), **C,** (c-comma), and **ALIGN** shall be contiguous with the last region allocated with one of the above words, unless the restrictions in the following paragraphs apply. The data-space pointer **HERE** always identifies the beginning of the next data-space region to be allocated. As successive allocations are made, the data-space pointer increases. A program may perform address arithmetic within contiguously allocated regions. The last region of data space allocated using the above operators may be released by allocating a corresponding negatively-sized region using **ALLOT**, subject to the restrictions of the following paragraphs.

CREATE establishes the beginning of a contiguous region of data space, whose starting address is returned by the **CREATE**d definition. This region is terminated by compiling the next definition.

Since an implementation is free to allocate data space for use by code, the above operators need not produce contiguous regions of data space if definitions are added to or removed from the dictionary between allocations. An ambiguous condition exists if deallocated memory contains definitions.

3.3.3.3 Variables

The region allocated for a variable may be non-contiguous with regions subsequently allocated with **,** (comma) or **ALLOT**. For example, in:

```
VARIABLE X  1 CELLS ALLOT
```

the region **X** and the region **ALLOT**ted could be non-contiguous.

Some system-provided variables, such as **STATE**, are restricted to read-only access.

3.3.3.4 Text-literal regions

The text-literal regions, specified by strings compiled with **S"** and **C"**, may be read-only.

A program shall not store into the text-literal regions created by **S"** and **C"** nor into any read-only system variable or read-only transient regions. An ambiguous condition exists when a program attempts to store into read-only regions.

3.3.3.5 Input buffers

The address, length, and content of the input buffer may be transient. A program shall not write into the input buffer. In the absence of any optional word sets providing alternative input sources, the input buffer is either the terminal-input buffer, used by **QUIT** to hold one line from the user input device, or a buffer specified by **EVALUATE**. In all cases, **SOURCE** returns the beginning address and length in characters of the current input buffer.

The minimum size of the terminal-input buffer shall be 80 characters.

The address and length returned by **SOURCE**, the string returned by **PARSE**, and directly computed input-buffer addresses are valid only until the text interpreter does I/O to refill the input buffer or the input source is changed.

A program may modify the size of the parse area by changing the contents of **>IN** within the limits imposed by this Standard. For example, if the contents of **>IN** are saved before a parsing operation and restored afterwards, the text that was parsed will be available again for subsequent parsing operations. The extent of permissible repositioning using this method depends on the input source (see **7.3.3 Block buffer regions** and **11.3.4 Input source**).

A program may directly examine the input buffer using its address and length as returned by **SOURCE**; the beginning of the parse area within the input buffer is indexed by the number in **>IN**. The values are valid for a limited time. An ambiguous condition exists if a program modifies the contents of the input buffer.

3.3.3.6 Other transient regions

The data space regions identified by **PAD**, **WORD**, and **#>** (the pictured numeric output string buffer) may be transient. Their addresses and contents may become invalid after:

– a definition is created via a defining word;
– definitions are compiled with **:** or **:NONAME**;
– data space is allocated using **ALLOT**, **,** (comma), **C,** (c-comma), or **ALIGN**.

The previous contents of the regions identified by **WORD** and **#>** may be invalid after each use of these words. Further, the regions returned by **WORD** and **#>** may overlap in memory. Consequently, use of one of these words can corrupt a region returned earlier by a different word. The other words that construct pictured numeric output strings (**<#**, **#**, **#S**, and **HOLD**) may also modify the contents of these regions. Words that display numbers may be implemented using pictured numeric output words. Consequently, **.** (dot), **.R**, **.S**, **?**, **D.**, **D.R**, **U.**, and **U.R** could also corrupt the regions.

The size of the scratch area whose address is returned by **PAD** shall be at least 84 characters. The contents of the region addressed by **PAD** are intended to be under the complete control of the user: no words defined in this Standard place anything in the region, although changing data-space allocations as described in **3.3.3.2 Contiguous regions** may change the address returned by **PAD**. Non-standard words provided by an implementation may use **PAD**, but such use shall be documented.

The size of the region identified by **WORD** shall be at least 33 characters.

The size of the pictured numeric output string buffer shall be at least (2**n*) + 2 characters, where *n* is the number of bits in a cell. Programs that consider it a fixed area with unchanging access parameters have an environmental dependency.

3.4 The Forth text interpreter

Upon start-up, a system shall be able to interpret, as described by **6.1.2050 QUIT**, Forth source code received interactively from a user input device.

Such interactive systems usually furnish a "prompt" indicating that they have accepted a user request and acted on it. The implementation-defined Forth prompt should contain the word “OK” in some combination of upper or lower case.

Text interpretation (see **6.1.1360 EVALUATE** and **6.1.2050 QUIT**) shall repeat the following steps until either the parse area is empty or an ambiguous condition exists:

a) Skip leading spaces and parse a *name* (see **3.4.1**);
b) Search the dictionary name space (see **3.4.2**). If a definition name matching the string is found:
   1) if interpreting, perform the interpretation semantics of the definition (see **3.4.3.2**), and continue at a);
   2) if compiling, perform the compilation semantics of the definition (see **3.4.3.3**), and continue at a).
c) If a definition name matching the string is not found, attempt to convert the string to a number (see **3.4.1.3**). If successful:
   1) if interpreting, place the number on the data stack, and continue at a);
   2) if compiling, compile code that when executed will place the number on the stack (see **6.1.1780 LITERAL**), and continue at a);
d) If unsuccessful, an ambiguous condition exists (see **3.4.4**).

3.4.1 Parsing

Unless otherwise noted, the number of characters parsed may be from zero to the implementation-defined maximum length of a counted string.

If the parse area is empty, i.e., when the number in **>IN** is equal to the length of the input buffer, or contains no characters other than delimiters, the selected string is empty. Otherwise, the selected string begins with the next character in the parse area, which is the character indexed by the contents of **>IN**. An ambiguous condition exists if the number in **>IN** is greater than the size of the input buffer.

If delimiter characters are present in the parse area after the beginning of the selected string, the string continues up to and including the character just before the first such delimiter, and the number in **>IN** is changed to index immediately past that delimiter, thus removing the parsed characters and the delimiter from the parse area. Otherwise, the string continues up to and including the last character in the parse area, and the number in **>IN** is changed to the length of the input buffer, thus emptying the parse area.

Parsing may change the contents of **>IN**, but shall not affect the contents of the input buffer. Specifically, if the value in **>IN** is saved before starting the parse, resetting **>IN** to that value immediately after the parse shall restore the parse area without loss of data.

3.4.1.1 Delimiters

If the delimiter is the space character, hex 20 (**BL**), control characters may be treated as delimiters. The set of conditions, if any, under which a "space" delimiter matches control characters is implementation defined.

To skip leading delimiters is to pass by zero or more contiguous delimiters in the parse area before parsing.

3.4.1.2 Syntax

Forth has a simple, operator-ordered syntax. The phrase **A B C** returns values as if **A** were executed first, then **B** and finally **C**. Words that cause deviations from this linear flow of control are called control-flow words. Combinations of control-flow words whose stack effects are compatible form control-flow structures. Examples of typical use are given for each control-flow word in Annex **A**.

Forth syntax is extensible; for example, new control-flow words can be defined in terms of existing ones.

This Standard does not require a syntax or program-construct checker.

3.4.1.3 Text interpreter input number conversion

When converting input numbers, the text interpreter shall recognize both positive and negative numbers, with a negative number represented by a single minus sign, the character "-", preceding the digits. The value in **BASE** is the radix for number conversion.

3.4.2 Finding definition names

A string matches a definition name if each character in the string matches the corresponding character in the string used as the definition name when the definition was created. The case sensitivity (whether or not the upper-case letters match the lower-case letters) is implementation defined. A system may be either case sensitive, treating upper- and lower-case letters as different and not matching, or case insensitive, ignoring differences in case while searching.

The matching of upper- and lower-case letters with alphabetic characters in character set extensions such as accented international characters is implementation defined.

A system shall be capable of finding the definition names defined by this Standard when they are spelled with upper-case letters.

3.4.3 Semantics

The semantics of a Forth definition are implemented by machine code or a sequence of execution tokens or other representations. They are largely specified by the stack notation in the glossary entries, which

shows what values shall be consumed and produced. The prose in each glossary entry further specifies the definition's behavior.

Each Forth definition may have several behaviors, described in the following sections. The terms "initiation semantics" and "run-time semantics" refer to definition fragments, and have meaning only within the individual glossary entries where they appear.

3.4.3.1 Execution semantics

The execution semantics of each Forth definition are specified in an "Execution:" section of its glossary entry. When a definition has only one specified behavior, the label is omitted.

Execution may occur implicitly, when the definition into which it has been compiled is executed, or explicitly, when its execution token is passed to **EXECUTE**. The execution semantics of a syntactically correct definition under conditions other than those specified in this Standard are implementation dependent.

Glossary entries for defining words include the execution semantics for the new definition in a "*name* Execution:" section.

3.4.3.2 Interpretation semantics

Unless otherwise specified in an "Interpretation:" section of the glossary entry, the interpretation semantics of a Forth definition are its execution semantics.

A system shall be capable of executing, in interpretation state, all of the definitions from the Core word set and any definitions included from the optional word sets or word set extensions whose interpretation semantics are defined by this Standard.

A system shall be capable of executing, in interpretation state, any new definitions created in accordance with **3. Usage requirements**.

3.4.3.3 Compilation semantics

Unless otherwise specified in a "Compilation:" section of the glossary entry, the compilation semantics of a Forth definition shall be to append its execution semantics to the execution semantics of the current definition.

3.4.4 Possible actions on an ambiguous condition

When an ambiguous condition exists, a system may take one or more of the following actions:

- ignore and continue;
- display a message;
- execute a particular word;
- set interpretation state and begin text interpretation;
- take other implementation-defined actions;
- take implementation-dependent actions.

The response to a particular ambiguous condition need not be the same under all circumstances.

3.4.5 Compilation

A program shall not attempt to nest compilation of definitions.

During the compilation of the current definition, a program shall not execute any defining word, **:NONAME**, or any definition that allocates dictionary data space. The compilation of the current definition may be suspended using **[** (left-bracket) and resumed using **]** (right-bracket). While the compilation of

the current definition is suspended, a program shall not execute any defining word, **:NONAME**, or any definition that allocates dictionary data space.

4. Documentation requirements

When it is impossible or infeasible for a system or program to define a particular behavior itself, it is permissible to state that the behavior is unspecifiable and to explain the circumstances and reasons why this is so.

4.1 System documentation

4.1.1 Implementation-defined options

The implementation-defined items in the following list represent characteristics and choices left to the discretion of the implementor, provided that the requirements of this Standard are met. A system shall document the values for, or behaviors of, each item.

- aligned address requirements (**3.1.3.3 Addresses**);
- behavior of **6.1.1320 `EMIT`** for non-graphic characters;
- character editing of **6.1.0695 `ACCEPT`** and **6.2.1390 `EXPECT`**;
- character set (**3.1.2 Character types, 6.1.1320 `EMIT`, 6.1.1750 `KEY`**);
- character-aligned address requirements (**3.1.3.3 Addresses**);
- character-set-extensions matching characteristics (**3.4.2 Finding definition names**);
- conditions under which control characters match a space delimiter (**3.4.1.1 Delimiters**);
- format of the control-flow stack (**3.2.3.2 Control-flow stack**);
- conversion of digits larger than thirty-five (**3.2.1.2 Digit conversion**);
- display after input terminates in **6.1.0695 `ACCEPT`** and **6.2.1390 `EXPECT`**;
- exception abort sequence (as in **6.1.0680 `ABORT"`**);
- input line terminator (**3.2.4.1 User input device**);
- maximum size of a counted string, in characters (**3.1.3.4 Counted strings**, **6.1.2450 `WORD`**);
- maximum size of a parsed string (**3.4.1 Parsing**);
- maximum size of a definition name, in characters (**3.3.1.2 Definition names**);
- maximum string length for **6.1.1345 `ENVIRONMENT?`**, in characters;
- method of selecting **3.2.4.1 User input device**;
- method of selecting **3.2.4.2 User output device**;
- methods of dictionary compilation (**3.3 The Forth dictionary**);
- number of bits in one address unit (**3.1.3.3 Addresses**);
- number representation and arithmetic (**3.2.1.1 Internal number representation**);
- ranges for n, +n, u, d, +d, ud (**3.1.3 Single-cell types**, **3.1.4 Cell-pair types**);
- read-only data-space regions (**3.3.3 Data space**);
- size of buffer at **6.1.2450 `WORD`** (**3.3.3.6 Other transient regions**);
- size of one cell in address units (**3.1.3 Single-cell types**);
- size of one character in address units (**3.1.2 Character types**);
- size of the keyboard terminal input buffer (**3.3.3.5 Input buffers**);
- size of the pictured numeric output string buffer (**3.3.3.6 Other transient regions**);
- size of the scratch area whose address is returned by **6.2.2000 `PAD`** (**3.3.3.6 Other transient regions**);
- system case-sensitivity characteristics (**3.4.2 Finding definition names**);
- system prompt (**3.4 The Forth text interpreter**, **6.1.2050 `QUIT`**);
- type of division rounding (**3.2.2.1 Integer division**, **6.1.0100 `*/`**, **6.1.0110 `*/MOD`**, **6.1.0230 `/`**, **6.1.0240 `/MOD`**, **6.1.1890 `MOD`**);
- values of **6.1.2250 `STATE`** when true;
- values returned after arithmetic overflow (**3.2.2.2 Other integer operations**);

– whether the current definition can be found after **6.1.1250 DOES>** (**6.1.0450 :**).

4.1.2 Ambiguous conditions

A system shall document the system action taken upon each of the general or specific ambiguous conditions identified in this Standard. See **3.4.4 Possible actions on an ambiguous condition**.

The following specific ambiguous conditions, indicated by an asterisk, are noted in the glossary entries of the relevant words. The following general ambiguous conditions could occur because of a combination of factors:

– a *name* is neither a valid definition name nor a valid number during text interpretation (**3.4 The Forth text interpreter**);
– a definition name exceeded the maximum length allowed (**3.3.1.2 Definition names**);
– addressing a region not listed in **3.3.3 Data Space**;
– argument type incompatible with specified input parameter, e.g., passing a *flag* to a word expecting an *n* (**3.1 Data types**);
– attempting to obtain the execution token, (e.g., with **6.1.0070 '**, **6.1.1550 FIND**, etc.) of a definition with undefined interpretation semantics;
– dividing by zero (**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*/**);
– insufficient data-stack space or return-stack space (stack overflow);
– insufficient space for loop-control parameters;
– insufficient space in the dictionary;
– interpretating a word with undefined interpretation semantics;
– modifying the contents of the input buffer or a string literal (**3.3.3.4 Text-literal regions**, **3.3.3.5 Input buffers**);
– overflow of a pictured numeric output string;
– parsed string overflow;
– producing a result out of range, e.g., multiplication (using *****) results in a value too big to be represented by a single-cell integer (**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*/**);
– reading from an empty data stack or return stack (stack underflow);
– unexpected end of input buffer, resulting in an attempt to use a zero-length string as a *name*;
* **>IN** greater than size of input buffer (**3.4.1 Parsing**);
* **6.1.2120 RECURSE** appears after **6.1.1250 DOES>**;
* argument input source different than current input source for **6.2.2148 RESTORE-INPUT**;
* data space containing definitions is de-allocated (**3.3.3.2 Contiguous regions**);
* data space read/write with incorrect alignment (**3.3.3.1 Address alignment**);
* data-space pointer not properly aligned (**6.1.0150 ,**, **6.1.0860 C,**);
* less than u+2 stack items (**6.2.2030 PICK**, **6.2.2150 ROLL**);
* loop-control parameters not available (**6.1.0140 +LOOP**, **6.1.1680 I**, **6.1.1730 J**, **6.1.1760 LEAVE**, **6.1.1800 LOOP**, **6.1.2380 UNLOOP**);
* most recent definition does not have a *name* (**6.1.1710 IMMEDIATE**);
* *name* not defined by **6.2.2405 VALUE** used by **6.2.2295 TO**;
* *name* not found (**6.1.0070 '**, **6.1.2033 POSTPONE**, **6.1.2510 ['']**, **6.2.2530 [COMPILE]**);
* parameters are not of the same type (**6.1.1240 DO**, **6.2.0620 ?DO**, **6.2.2440 WITHIN**);
* **6.1.2033 POSTPONE** or **6.2.2530 [COMPILE]** applied to **6.2.2295 TO**;
* string longer than a counted string returned by **6.1.2450 WORD**;
* *u* greater than or equal to the number of bits in a cell (**6.1.1805 LSHIFT**, **6.1.2162 RSHIFT**);

* word not defined via **6.1.1000 CREATE** (**6.1.0550 >BODY**, **6.1.1250 DOES>**);
* words improperly used outside **6.1.0490 <#** and **6.1.0040 #>** (**6.1.0030 #**, **6.1.0050 #S**, **6.1.1670 HOLD**, **6.1.2210 SIGN**).

4.1.3 Other system documentation

A system shall provide the following information:

– list of non-standard words using **6.2.2000 PAD** (**3.3.3.6 Other transient regions**);
– operator's terminal facilities available;
– program data space available, in address units;
– return stack space available, in cells;
– stack space available, in cells;
– system dictionary space required, in address units.

4.2 Program documentation

4.2.1 Environmental dependencies

A program shall document the following environmental dependencies, where they apply, and should document other known environmental dependencies:

– considering the pictured numeric output string buffer a fixed area with unchanging access parameters (**3.3.3.6 Other transient regions**);
– depending on the presence or absence of non-graphic characters in a received string (**6.1.0695 ACCEPT**, **6.2.1390 EXPECT**);
– relying on a particular rounding direction (**3.2.2.1 Integer division**);
– requiring a particular number representation and arithmetic (**3.2.1.1 Internal number representation**);
– requiring non-standard words or techniques (**3. Usage requirements**);
– requiring the ability to send or receive control characters (**3.1.2.2 Control characters**, **6.1.1750 KEY**);
– using control characters to perform specific functions (**6.1.1320 EMIT**, **6.1.2310 TYPE**);
– using flags as arithmetic operands (**3.1.3.1 Flags**);
– using lower case for standard definition names or depending on the case sensitivity of a system (**3.3.1.2 Definition names**);
– using the graphic character with a value of hex 24 (**3.1.2.1 Graphic characters**).

4.2.2 Other program documentation

A program shall also document:

– minimum operator's terminal facilities required;
– whether a Standard System exists after the program is loaded.

5. Compliance and labeling

5.1 ANS Forth systems

5.1.1 System compliance

A system that complies with all the system requirements given in sections **3. Usage requirements** and **4.1 System documentation** and their sub-sections is a Standard System. An otherwise Standard System that provides only a portion of the Core words is a Standard System Subset. An otherwise Standard System (Subset) that fails to comply with one or more of the minimum values or ranges specified in **3. Usage requirements** and its sub-sections has environmental restrictions.

5.1.2 System labeling

A Standard System (Subset) shall be labeled an "ANS Forth System (Subset)." That label, by itself, shall not be applied to Standard Systems or Standard System Subsets that have environmental restrictions.

The phrase "with Environmental Restrictions" shall be appended to the label of a Standard System (Subset) that has environmental restrictions.

The phrase "Providing *name*(*s*) from the Core Extensions word set" shall be appended to the label of any Standard System that provides portions of the Core Extensions word set.

The phrase "Providing the Core Extensions word set" shall be appended to the label of any Standard System that provides all of the Core Extensions word set.

5.2 ANS Forth programs

5.2.1 Program compliance

A program that complies with all the program requirements given in sections **3. Usage requirements** and **4.2 Program documentation** and their sub-sections is a Standard Program.

5.2.2 Program labeling

A Standard Program shall be labeled an "ANS Forth Program." That label, by itself, shall not be applied to Standard Programs that require the system to provide standard words outside the Core word set or that have environmental dependencies.

The phrase "with Environmental Dependencies" shall be appended to the label of Standard Programs that have environmental dependencies.

The phrase "Requiring *name*(*s*) from the Core Extensions word set" shall be appended to the label of Standard Programs that require the system to provide portions of the Core Extensions word set.

The phrase "Requiring the Core Extensions word set" shall be appended to the label of Standard Programs that require the system to provide all of the Core Extensions word set.

6. Glossary

6.1 Core words

6.1.0010 ! "store" CORE

(*x a-addr* --)

Store *x* at *a-addr*.

See: **3.3.3.1 Address alignment**.

6.1.0030 # "number-sign" CORE

(ud_1 -- ud_2)

Divide ud_1 by the number in **BASE** giving the quotient ud_2 and the remainder *n*. (*n* is the least-significant digit of ud_1.) Convert *n* to external form and add the resulting character to the beginning of the pictured numeric output string. An ambiguous condition exists if **#** executes outside of a **<# #>** delimited number conversion.

See: **6.1.0040 #>**, **6.1.0050 #S**, **6.1.0490 <#**.

6.1.0040 #> "number-sign-greater" CORE

(*xd* -- *c-addr u*)

Drop *xd*. Make the pictured numeric output string available as a character string. *c-addr* and *u* specify the resulting character string. A program may replace characters within the string.

See: **6.1.0030 #**, **6.1.0050 #S**, **6.1.0490 <#**.

6.1.0050 #S "number-sign-s" CORE

(ud_1 -- ud_2)

Convert one digit of ud_1 according to the rule for **#**. Continue conversion until the quotient is zero. ud_2 is zero. An ambiguous condition exists if **#S** executes outside of a **<# #>** delimited number conversion.

See: **6.1.0030 #**, **6.1.0040 #>**, **6.1.0490 <#**.

6.1.0070 ' "tick" CORE

("<*spaces*>*name*" -- *xt*)

Skip leading space delimiters. Parse *name* delimited by a space. Find *name* and return *xt*, the execution token for *name*. An ambiguous condition exists if *name* is not found.

When interpreting, **' name EXECUTE** is equivalent to **name**.

See: **3.4 The Forth text interpreter**, **3.4.1 Parsing**, **A.6.1.2033 POSTPONE**, **A.6.1.2510 [']**, **D.6.7 Immediacy**.

6.1.0080 ("paren" CORE

Compilation: Perform the execution semantics given below.

Execution: ("*ccc*<*paren*>" --)

Parse *ccc* delimited by a right parenthesis ")". **(** is an immediate word.

The number of characters in *ccc* may be zero to the number of characters in the parse area.

See: **3.4.1 Parsing**, **11.6.1.0080 (**.

6.1.0090 * "star" CORE

($n_1|u_1$ $n_2|u_2$ -- $n_3|u_3$)

Multiply $n_1|u_1$ by $n_2|u_2$ giving the product $n_3|u_3$.

6.1.0100 */ "star-slash" CORE

(n_1 n_2 n_3 -- n_4)

Multiply n_1 by n_2 producing the double-cell intermediate result d. Divide d by n_3 giving the single-cell quotient n_4. An ambiguous condition exists if n_3 is zero or if the quotient n_4 lies outside the range of a signed number. If d and n_3 differ in sign, the implementation-defined result returned will be the same as that returned by either the phrase **>R M* R> FM/MOD SWAP DROP** or the phrase **>R M* R> SM/REM SWAP DROP**.

See: **3.2.2.1 Integer division**.

6.1.0110 */MOD "star-slash-mod" CORE

(n_1 n_2 n_3 -- n_4 n_5)

Multiply n_1 by n_2 producing the intermediate double-cell result d. Divide d by n_3 producing the single-cell remainder n_4 and the single-cell quotient n_5. An ambiguous condition exists if n_3 is zero, or if the quotient n_5 lies outside the range of a single-cell signed integer. If d and n_3 differ in sign, the implementation-defined result returned will be the same as that returned by either the phrase **>R M* R> FM/MOD** or the phrase **>R M* R> SM/REM**.

See: **3.2.2.1 Integer division**.

6.1.0120 + "plus" CORE

($n_1|u_1$ $n_2|u_2$ -- $n_3|u_3$)

Add $n_2|u_2$ to $n_1|u_1$, giving the sum $n_3|u_3$.

See: **3.3.3.1 Address alignment**.

6.1.0130 +! "plus-store" CORE

(*n|u a-addr* --)

Add *n|u* to the single-cell number at *a-addr*.

See: **3.3.3.1 Address alignment**.

6.1.0140 +LOOP "plus-loop" CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: *do-sys* --)

Append the run-time semantics given below to the current definition. Resolve the destination of all unresolved occurrences of **LEAVE** between the location given by *do-sys* and the next location for a transfer of control, to execute the words following **+LOOP**.

Run-time: (*n* --) (R: $loop\text{-}sys_1$ -- | $loop\text{-}sys_2$)

An ambiguous condition exists if the loop control parameters are unavailable. Add *n* to the loop index. If the loop index did not cross the boundary between the loop limit minus one and the loop limit, continue execution at the beginning of the loop. Otherwise, discard the current loop control parameters and continue execution immediately following the loop.

See: **6.1.1240 DO**, **6.1.1680 I**, **6.1.1760 LEAVE**.

6.1.0150 , "comma" CORE

(*x* --)

Reserve one cell of data space and store *x* in the cell. If the data-space pointer is aligned when **,** begins execution, it will remain aligned when **,** finishes execution. An ambiguous condition exists if the data-space pointer is not aligned prior to execution of **,**.

See: **3.3.3 Data space**, **3.3.3.1 Address alignment**.

6.1.0160 – "minus" CORE

($n_1|u_1$ $n_2|u_2$ -- $n_3|u_3$)

Subtract $n_2|u_2$ from $n_1|u_1$, giving the difference $n_3|u_3$.

See: **3.3.3.1 Address alignment**.

6.1.0180 . "dot" CORE

(*n* --)

Display *n* in free field format.

See: **3.2.1.2 Digit conversion**, **3.2.1.3 Free-field number display**.

6.1.0190 ." "dot-quote" CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: ("*ccc<quote>*" --)

Parse *ccc* delimited by **"** (double-quote). Append the run-time semantics given below to the current definition.

Run-time: (--)

Display *ccc*.

See: **3.4.1 Parsing**, **6.2.0200 .(**.

6.1.0230 / "slash" CORE

(n_1 n_2 -- n_3)

Divide n_1 by n_2, giving the single-cell quotient n_3. An ambiguous condition exists if n_2 is zero. If n_1 and n_2 differ in sign, the implementation-defined result returned will be the same as that returned by either the phrase **>R S>D R> FM/MOD SWAP DROP** or the phrase **>R S>D R> SM/REM SWAP DROP**.

See: **3.2.2.1 Integer division**.

6.1.0240 /MOD "slash-mod" CORE

(n_1 n_2 -- n_3 n_4)

Divide n_1 by n_2, giving the single-cell remainder n_3 and the single-cell quotient n_4. An ambiguous condition exists if n_2 is zero. If n_1 and n_2 differ in sign, the implementation-defined result returned will be the same as that returned by either the phrase **>R S>D R> FM/MOD** or the phrase **>R S>D R> SM/REM**.

See: **3.2.2.1 Integer division**.

6.1.0250 0< "zero-less" CORE

(*n* -- *flag*)

flag is true if and only if *n* is less than zero.

6.1.0270 0= "zero-equals" CORE

(*x* -- *flag*)

flag is true if and only if *x* is equal to zero.

6.1.0290 1+ "one-plus" CORE

($n_1|u_1$ -- $n_2|u_2$)

Add 1 to $n_1|u_1$ giving the sum $n_2|u_2$.

6.1.0300 1- "one-minus" CORE

($n_1|u_1$ -- $n_2|u_2$)

Subtract 1 from $n_1|u_1$ giving the difference $n_2|u_2$.

6.1.0310 2! "two-store" CORE

(x_1 x_2 *a-addr* --)

Store the cell pair x_1 x_2 at *a-addr*, with x_2 at *a-addr* and x_1 at the next consecutive cell. It is equivalent to the sequence **SWAP OVER ! CELL+ !**.

See: **3.3.3.1 Address alignment**.

6.1.0320 2* "two-star" CORE

(x_1 -- x_2)

x_2 is the result of shifting x_1 one bit toward the most-significant bit, filling the vacated least-significant bit with zero.

6.1.0330 2/ "two-slash" CORE

(x_1 -- x_2)

x_2 is the result of shifting x_1 one bit toward the least-significant bit, leaving the most-significant bit unchanged.

6.1.0350 2@ "two-fetch" CORE

(*a-addr* -- x_1 x_2)

Fetch the cell pair x_1 x_2 stored at *a-addr*. x_2 is stored at *a-addr* and x_1 at the next consecutive cell. It is equivalent to the sequence **DUP CELL+ @ SWAP @**.

See: **3.3.3.1 Address alignment**, **6.1.0310 2!**.

6.1.0370 2DROP "two-drop" CORE

(x_1 x_2 --)

Drop cell pair x_1 *x2* from the stack.

6.1.0380 2DUP "two-dupe" CORE

(x_1 x_2 -- x_1 x_2 x_1 x_2)

Duplicate cell pair x_1 x_2.

6.1.0400 2OVER "two-over" CORE

(x_1 x_2 x_3 x_4 -- x_1 x_2 x_3 x_4 x_1 x_2)

Copy cell pair x_1 x_2 to the top of the stack.

6.1.0430 2SWAP "two-swap" CORE

(x_1 x_2 x_3 x_4 -- x_3 x_4 x_1 x_2)

Exchange the top two cell pairs.

6.1.0450 : "colon" CORE

(C: "*<spaces>name*" -- *colon-sys*)

Skip leading space delimiters. Parse *name* delimited by a space. Create a definition for *name*, called a "colon definition." Enter compilation state and start the current definition, producing *colon-sys*. Append the initiation semantics given below to the current definition.

The execution semantics of *name* will be determined by the words compiled into the body of the definition. The current definition shall not be findable in the dictionary until it is ended (or until the execution of **DOES>** in some systems).

Initiation: (*i*x* -- *i*x*) (R: -- *nest-sys*)

Save implementation-dependent information *nest-sys* about the calling definition. The stack effects *i*x* represent arguments to *name*.

name Execution: (*i*x* -- *j*x*)

Execute the definition *name*. The stack effects *i*x* and *j*x* represent arguments to and results from *name*, respectively.

See: **3.4 The Forth text interpreter**, **3.4.1 Parsing**, **3.4.5 Compilation**, **6.1.1250 DOES>**, **6.1.2500 [**, **6.1.2540]**, **15.6.2.0470 ;CODE**.

6.1.0460 ; "semicolon" CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: *colon-sys* --)

Append the run-time semantics below to the current definition. End the current definition, allow it to be found in the dictionary and enter interpretation state, consuming *colon-sys*. If the data-space pointer is not aligned, reserve enough data space to align it.

Run-time: (--) (R: *nest-sys* --)

Return to the calling definition specified by *nest-sys*.

See: **3.4 The Forth text interpreter**, **3.4.5 Compilation**.

6.1.0480 < "less-than" CORE

(n_1 n_2 -- *flag*)

flag is true if and only if n_1 is less than n_2.

See: **6.1.2340 U<**.

6.1.0490 <# "less-number-sign" CORE

(--)

Initialize the pictured numeric output conversion process.

See: **6.1.0030 #**, **6.1.0040 #>**, **6.1.0050 #S**.

6.1.0530 = "equals" CORE

(x_1 x_2 -- *flag*)

flag is true if and only if x_1 is bit-for-bit the same as x_2.

6.1.0540 > "greater-than" CORE

(n_1 n_2 -- *flag*)

flag is true if and only if n_1 is greater than n_2.

See: **6.2.2350 U>**.

6.1.0550 >BODY "to-body" CORE

(*xt* -- *a-addr*)

a-addr is the data-field address corresponding to *xt*. An ambiguous condition exists if *xt* is not for a word defined via **CREATE**.

See: **3.3.3 Data space**.

6.1.0560 >IN "to-in" CORE

(-- *a-addr*)

a-addr is the address of a cell containing the offset in characters from the start of the input buffer to the start of the parse area.

6.1.0570 >NUMBER "to-number" CORE

(ud_1 *c-addr*$_1$ u_1 -- ud_2 *c-addr*$_2$ u_2)

ud_2 is the unsigned result of converting the characters within the string specified by *c-addr*$_1$ u_1 into digits, using the number in **BASE**, and adding each into ud_1 after multiplying ud_1 by the number in **BASE**. Conversion continues left-to-right until a character that is not convertible, including any "+" or "-", is encountered or the string is entirely converted. *c-addr*$_2$ is the location of the first unconverted character or the first character past the end of the string if the string was entirely converted. u_2 is the number of unconverted characters in the string. An ambiguous condition exists if ud_2 overflows during the conversion.

See: **3.2.1.2 Digit conversion**.

6.1.0580 `>R` "to-r" CORE

Interpretation: Interpretation semantics for this word are undefined.

Execution: (*x* --) (R: -- *x*)

Move *x* to the return stack.

See: **3.2.3.3 Return stack**, **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*)

Duplicate *x* if it is non-zero.

6.1.0650 `@` "fetch" CORE

(*a-addr* -- *x*)

x is the value stored at *a-addr*.

See: **3.3.3.1 Address alignment**.

6.1.0670 `ABORT` CORE

(*i*x* --) (R: *j*x* --)

Empty the data stack and perform the function of **`QUIT`**, which includes emptying the return stack, without displaying a message.

See: **9.6.2.0670 `ABORT`**.

6.1.0680 `ABORT"` "abort-quote" CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: ("*ccc<quote>*" --)

Parse *ccc* delimited by a **`"`** (double-quote). Append the run-time semantics given below to the current definition.

Run-time: (*i*x* x_1 -- | *i*x*) (R: *j*x* -- | *j*x*)

Remove x_1 from the stack. If any bit of x_1 is not zero, display *ccc* and perform an implementation-defined abort sequence that includes the function of **`ABORT`**.

See: **3.4.1 Parsing**, **9.6.2.0680 `ABORT"`**.

6.1.0690 `ABS` "abs" CORE

(*n* -- *u*)

u is the absolute value of *n*.

6.1.0695 **ACCEPT** CORE

(*c-addr* $+n_1$ -- $+n_2$)

Receive a string of at most $+n_1$ characters. An ambiguous condition exists if $+n_1$ is zero or greater than 32,767. Display graphic characters as they are received. A program that depends on the presence or absence of non-graphic characters in the string has an environmental dependency. The editing functions, if any, that the system performs in order to construct the string are implementation-defined.

Input terminates when an implementation-defined line terminator is received. When input terminates, nothing is appended to the string, and the display is maintained in an implementation-defined way.

$+n_2$ is the length of the string stored at *c-addr*.

6.1.0705 **ALIGN** CORE

(--)

If the data-space pointer is not aligned, reserve enough space to align it.

See: **3.3.3 Data space**, **3.3.3.1 Address alignment**.

6.1.0706 **ALIGNED** CORE

(*addr* -- *a-addr*)

a-addr is the first aligned address greater than or equal to *addr*.

See: **3.3.3.1 Address alignment**.

6.1.0710 **ALLOT** CORE

(*n* --)

If *n* is greater than zero, reserve *n* address units of data space. If *n* is less than zero, release |*n*| address units of data space. If *n* is zero, leave the data-space pointer unchanged.

If the data-space pointer is aligned and *n* is a multiple of the size of a cell when **ALLOT** begins execution, it will remain aligned when **ALLOT** finishes execution.

If the data-space pointer is character aligned and *n* is a multiple of the size of a character when **ALLOT** begins execution, it will remain character aligned when **ALLOT** finishes execution.

See: **3.3.3 Data space**.

6.1.0720 **AND** CORE

(x_1 x_2 -- x_3)

x_3 is the bit-by-bit logical “and” of x_1 with x_2.

6.1.0750 **BASE** CORE

(-- *a-addr*)

a-addr is the address of a cell containing the current number-conversion radix {{2...36}}.

6.1.0760 **BEGIN** CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: -- *dest*)

Put the next location for a transfer of control, *dest*, onto the control flow stack. Append the run-time semantics given below to the current definition.

Run-time: (--)

Continue execution.

See: **3.2.3.2 Control-flow stack**, **6.1.2140 REPEAT**, **6.1.2390 UNTIL**, **6.1.2430 WHILE**.

6.1.0770 **BL** "b-l" CORE

(-- *char*)

char is the character value for a space.

6.1.0850 **C!** "c-store" CORE

(*char c-addr* --)

Store *char* at *c-addr*. When character size is smaller than cell size, only the number of low-order bits corresponding to character size are transferred.

See: **3.3.3.1 Address alignment**

6.1.0860 **C,** "c-comma" CORE

(*char* --)

Reserve space for one character in the data space and store *char* in the space. If the data-space pointer is character aligned when **C,** begins execution, it will remain character aligned when **C,** finishes execution. An ambiguous condition exists if the data-space pointer is not character-aligned prior to execution of **C,**.

See: **3.3.3 Data space**, **3.3.3.1 Address alignment**.

6.1.0870 **C@** "c-fetch" CORE

(*c-addr* -- *char*)

Fetch the character stored at *c-addr*. When the cell size is greater than character size, the unused high-order bits are all zeroes.

See: **3.3.3.1 Address alignment**.

6.1.0880 CELL+ “cell-plus” CORE

(*a-addr*$_1$ -- *a-addr*$_2$)

Add the size in address units of a cell to *a-addr*$_1$, giving *a-addr*$_2$.

See: **3.3.3.1 Address alignment**.

6.1.0890 CELLS CORE

(n_1 -- n_2)

n_2 is the size in address units of n_1 cells.

6.1.0895 CHAR “char” CORE

(“<*spaces*>*name*” -- *char*)

Skip leading space delimiters. Parse *name* delimited by a space. Put the value of its first character onto the stack.

See: **3.4.1 Parsing**, **6.1.2520 [CHAR]**.

6.1.0897 CHAR+ “char-plus” CORE

(*c-addr*$_1$ -- *c-addr*$_2$)

Add the size in address units of a character to *c-addr*$_1$, giving *c-addr*$_2$.

See: **3.3.3.1 Address alignment**.

6.1.0898 CHARS “chars” CORE

(n_1 -- n_2)

n_2 is the size in address units of n_1 characters.

6.1.0950 CONSTANT CORE

(*x* “<*spaces*>*name*” --)

Skip leading space delimiters. Parse *name* delimited by a space. Create a definition for *name* with the execution semantics defined below.

name is referred to as a “constant.”

name Execution: (-- *x*)

Place *x* on the stack.

See: **3.4.1 Parsing**.

6.1.0980 **COUNT** CORE

(*c-addr*$_1$ -- *c-addr*$_2$ *u*)

Return the character string specification for the counted string stored at *c-addr*$_1$. *c-addr*$_2$ is the address of the first character after *c-addr*$_1$. *u* is the contents of the character at *c-addr*$_1$, which is the length in characters of the string at *c-addr*$_2$.

6.1.0990 **CR** "c-r" CORE

(--)

Cause subsequent output to appear at the beginning of the next line.

6.1.1000 **CREATE** CORE

("<*spaces*>*name*" --)

Skip leading space delimiters. Parse *name* delimited by a space. Create a definition for *name* with the execution semantics defined below. If the data-space pointer is not aligned, reserve enough data space to align it. The new data-space pointer defines *name*'s data field. **CREATE** does not allocate data space in *name*'s data field.

name Execution: (-- *a-addr*)

a-addr is the address of *name*'s data field. The execution semantics of *name* may be extended by using **DOES>**.

See: **3.3.3 Data space**, **6.1.1250 DOES>**.

6.1.1170 **DECIMAL** CORE

(--)

Set the numeric conversion radix to ten (decimal).

6.1.1200 **DEPTH** CORE

(-- +*n*)

+*n* is the number of single-cell values contained in the data stack before +*n* was placed on the stack.

6.1.1240 **DO** CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: -- *do-sys*)

Place *do-sys* onto the control-flow stack. Append the run-time semantics given below to the current definition. The semantics are incomplete until resolved by a consumer of *do-sys* such as **LOOP**.

Run-time: ($n_1|u_1$ $n_2|u_2$ --) (R: -- *loop-sys*)

Set up loop control parameters with index $n_2|u_2$ and limit $n_1|u_1$. An ambiguous condition exists if $n_1|u_1$ and $n_2|u_2$ are not both the same type. Anything already on the return stack becomes unavailable until the loop-control parameters are discarded.

See: **3.2.3.2 Control-flow stack**, **6.1.0140 +LOOP**, **6.1.1800 LOOP**.

6.1.1250 **DOES>** "does" CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: $colon\text{-}sys_1$ -- $colon\text{-}sys_2$)

Append the run-time semantics below to the current definition. Whether or not the current definition is rendered findable in the dictionary by the compilation of **DOES>** is implementation defined. Consume $colon\text{-}sys_1$ and produce $colon\text{-}sys_2$. Append the initiation semantics given below to the current definition.

Run-time: (--) (R: $nest\text{-}sys_1$ --)

Replace the execution semantics of the most recent definition, referred to as *name*, with the *name* execution semantics given below. Return control to the calling definition specified by $nest\text{-}sys_1$. An ambiguous condition exists if *name* was not defined with **CREATE** or a user-defined word that calls **CREATE**.

Initiation: (*i*x a-addr* -- *i*x*) (R: -- $nest\text{-}sys_2$)

Save implementation-dependent information $nest\text{-}sys_2$ about the calling definition. Place *name*'s data field address on the stack. The stack effects *i*x* represent arguments to *name*.

name Execution: (*i*x* -- *j*x*)

Execute the portion of the definition that begins with the initiation semantics appended by the **DOES>** which modified *name*. The stack effects *i*x* and *j*x* represent arguments to and results from *name*, respectively.

See: **6.1.1000 CREATE**.

6.1.1260 **DROP** CORE

(*x* --)

Remove *x* from the stack.

6.1.1290 **DUP** "dupe" CORE

(*x* -- *x x*)

Duplicate *x*.

6.1.1310 **ELSE** CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: $orig_1$ -- $orig_2$)

Put the location of a new unresolved forward reference $orig_2$ onto the control flow stack. Append the run-time semantics given below to the current definition. The semantics will be incomplete until $orig_2$ is resolved (e.g., by **THEN**). Resolve the forward reference $orig_1$ using the location following the appended execution semantics.

Run-time: (--)

Continue execution at the location given by the resolution of $orig_2$.

See: **6.1.1700 IF**, **6.1.2270 THEN**.

6.1.1320 **EMIT** CORE

(*x* --)

If *x* is a graphic character in the implementation-defined character set, display *x*. The effect of **EMIT** for all other values of *x* is implementation-defined.

When passed a character whose character-defining bits have a value between hex 20 and 7E inclusive, the corresponding standard character, specified by **3.1.2.1 Graphic characters**, is displayed. Because different output devices can respond differently to control characters, programs that use control characters to perform specific functions have an environmental dependency. Each **EMIT** deals with only one character.

See: **6.1.2310 TYPE**.

6.1.1345 **ENVIRONMENT?** "environment-query" CORE

(*c-addr u -- false* | *i*x true*)

c-addr is the address of a character string and *u* is the string's character count. *u* may have a value in the range from zero to an implementation-defined maximum which shall not be less than 31. The character string should contain a keyword from **3.2.6 Environmental queries** or the optional word sets to be checked for correspondence with an attribute of the present environment. If the system treats the attribute as unknown, the returned flag is *false*; otherwise, the flag is *true* and the *i*x* returned is of the type specified in the table for the attribute queried.

6.1.1360 **EVALUATE** CORE

(*i*x c-addr u -- j*x*)

Save the current input source specification. Store minus-one (-1) in **SOURCE-ID** if it is present. Make the string described by *c-addr* and *u* both the input source and input buffer, set **>IN** to zero, and interpret. When the parse area is empty, restore the prior input source specification. Other stack effects are due to the words **EVALUATE**d.

6.1.1370 **EXECUTE** CORE

(*i*x xt -- j*x*)

Remove *xt* from the stack and perform the semantics identified by it. Other stack effects are due to the word **EXECUTE**d.

See: **6.1.0070 '**, **6.1.2510 [']**.

6.1.1380 **EXIT** CORE

Interpretation: Interpretation semantics for this word are undefined.

Execution: (--) (R: *nest-sys* --)

Return control to the calling definition specified by *nest-sys*. Before executing **EXIT** within a do-loop, a program shall discard the loop-control parameters by executing **UNLOOP**.

See: **3.2.3.3 Return stack**, **6.1.2380 UNLOOP**.

6.1.1540 **FILL** CORE

(*c-addr u char* --)

If *u* is greater than zero, store *char* in each of *u* consecutive characters of memory beginning at *c-addr*.

6.1.1550 **FIND** CORE

(*c-addr* -- *c-addr* 0 | *xt* 1 | *xt* -1)

Find the definition named in the counted string at *c-addr*. If the definition is not found, return *c-addr* and zero. If the definition is found, return its execution token *xt*. If the definition is immediate, also return one (1), otherwise also return minus-one (-1). For a given string, the values returned by **FIND** while compiling may differ from those returned while not compiling.

See: **3.4.2 Finding definition names**, **A.6.1.0070 '**, **A.6.1.2510 [']**, **A.6.1.2033 POSTPONE**, **D.6.7 Immediacy**.

6.1.1561 **FM/MOD** "f-m-slash-mod" CORE

(d_1 n_1 -- n_2 n_3)

Divide d_1 by n_1, giving the floored quotient n_3 and the remainder n_2. Input and output stack arguments are signed. An ambiguous condition exists if n_1 is zero or if the quotient lies outside the range of a single-cell signed integer.

See: **3.2.2.1 Integer division**, **6.1.2214 SM/REM**, **6.1.2370 UM/MOD**.

6.1.1650 **HERE** CORE

(-- *addr*)

addr is the data-space pointer.

See: **3.3.3.2 Contiguous regions**.

6.1.1670 **HOLD** CORE

(*char* --)

Add *char* to the beginning of the pictured numeric output string. An ambiguous condition exists if **HOLD** executes outside of a **<# #>** delimited number conversion.

6.1.1680 **I** CORE

Interpretation: Interpretation semantics for this word are undefined.

Execution: (-- *n*|*u*) (R: *loop-sys* -- *loop-sys*)

n|*u* is a copy of the current (innermost) loop index. An ambiguous condition exists if the loop control parameters are unavailable.

6.1.1700 **IF** CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: -- *orig*)

Put the location of a new unresolved forward reference *orig* onto the control flow stack. Append the run-time semantics given below to the current definition. The semantics are incomplete until *orig* is resolved, e.g., by **THEN** or **ELSE**.

Run-time: (*x* --)

If all bits of *x* are zero, continue execution at the location specified by the resolution of *orig*.

See: **3.2.3.2 Control flow stack**, **6.1.1310 ELSE**, **6.1.2270 THEN**.

6.1.1710 **IMMEDIATE** CORE

(--)

Make the most recent definition an immediate word. An ambiguous condition exists if the most recent definition does not have a name.

See: **D.6.7 Immediacy**.

6.1.1720 **INVERT** CORE

(x_1 -- x_2)

Invert all bits of x_1, giving its logical inverse x_2.

See: **6.1.1910 NEGATE**, **6.1.0270 0=**.

6.1.1730 **J** CORE

Interpretation: Interpretation semantics for this word are undefined.

Execution: (-- *n*|*u*) (R: *loop-sys* -- *loop-sys*)

n|*u* is a copy of the index of the next outer loop. An ambiguous condition exists if the loop control parameters of the next outer loop are unavailable.

6.1.1750 **KEY** CORE

(-- *char*)

Receive one character *char*, a member of the implementation-defined character set. Keyboard events that do not correspond to such characters are discarded until a valid character is received, and those events are subsequently unavailable.

All standard characters can be received. Characters received by **KEY** are not displayed.

Any standard character returned by **KEY** has the numeric value specified in **3.1.2.1 Graphic characters**. Programs that require the ability to receive control characters have an environmental dependency.

See: **10.6.2.1307 EKEY**, **10.6.1.1755 KEY?**.

6.1.1760 **LEAVE** CORE

Interpretation: Interpretation semantics for this word are undefined.

Execution: (--) (R: *loop-sys* --)

Discard the current loop control parameters. An ambiguous condition exists if they are unavailable. Continue execution immediately following the innermost syntactically enclosing **DO ... LOOP** or **DO ... +LOOP**.

See: **3.2.3.3 Return stack**, **6.1.0140 +LOOP**, **6.1.1800 LOOP**.

6.1.1780 **LITERAL** CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (*x* --)

Append the run-time semantics given below to the current definition.

Run-time: (-- *x*)

Place *x* on the stack.

6.1.1800 **LOOP** CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: *do-sys* --)

Append the run-time semantics given below to the current definition. Resolve the destination of all unresolved occurrences of **LEAVE** between the location given by *do-sys* and the next location for a transfer of control, to execute the words following the **LOOP**.

Run-time: (--) (R: $loop\text{-}sys_1$ -- | $loop\text{-}sys_2$)

An ambiguous condition exists if the loop control parameters are unavailable. Add one to the loop index. If the loop index is then equal to the loop limit, discard the loop parameters and continue execution immediately following the loop. Otherwise continue execution at the beginning of the loop.

See: **6.1.1240 DO**, **6.1.1680 I**, **6.1.1760 LEAVE**.

6.1.1805 **LSHIFT** "l-shift" CORE

(x_1 *u* -- x_2)

Perform a logical left shift of *u* bit-places on x_1, giving x_2. Put zero into the least significant bits vacated by the shift. An ambiguous condition exists if *u* is greater than or equal to the number of bits in a cell.

6.1.1810 **M*** "m-star" CORE

(n_1 n_2 -- *d*)

d is the signed product of n_1 times n_2.

6.1.1870 **MAX** CORE

(n_1 n_2 -- n_3)

n_3 is the greater of n_1 and n_2.

6.1.1880 **MIN** CORE

(n_1 n_2 -- n_3)

n_3 is the lesser of n_1 and n_2.

6.1.1890 **MOD** CORE

(n_1 n_2 -- n_3)

Divide n_1 by n_2, giving the single-cell remainder n_3. An ambiguous condition exists if n_2 is zero. If n_1 and n_2 differ in sign, the implementation-defined result returned will be the same as that returned by either the phrase **>R S>D R> FM/MOD DROP** or the phrase **>R S>D R> SM/REM DROP**.

See: **3.2.2.1 Integer division**.

6.1.1900 **MOVE** CORE

($addr_1$ $addr_2$ u --)

If u is greater than zero, copy the contents of u consecutive address units at $addr_1$ to the u consecutive address units at $addr_2$. After **MOVE** completes, the u consecutive address units at $addr_2$ contain exactly what the u consecutive address units at $addr_1$ contained before the move.

See: **17.6.1.0910 CMOVE**, **17.6.1.0920 CMOVE>**.

6.1.1910 **NEGATE** CORE

(n_1 -- n_2)

Negate n_1, giving its arithmetic inverse n_2.

See: **6.1.1720 INVERT**, **6.1.0270 0=**.

6.1.1980 **OR** CORE

(x_1 x_2 -- x_3)

x_3 is the bit-by-bit inclusive-or of x_1 with x_2.

6.1.1990 **OVER** CORE

(x_1 x_2 -- x_1 x_2 x_1)

Place a copy of x_1 on top of the stack.

6.1.2033 **POSTPONE** CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: ("<*spaces*>*name*" --)

Skip leading space delimiters. Parse *name* delimited by a space. Find *name*. Append the compilation semantics of *name* to the current definition. An ambiguous condition exists if *name* is not found.

See: **3.4.1 Parsing**.

6.1.2050 **QUIT** CORE

(--) (R: *i***x* --)

Empty the return stack, store zero in **SOURCE-ID** if it is present, make the user input device the input source, and enter interpretation state. Do not display a message. Repeat the following:

– Accept a line from the input source into the input buffer, set **>IN** to zero, and interpret.

– Display the implementation-defined system prompt if in interpretation state, all processing has been completed, and no ambiguous condition exists.

See: **3.4 The Forth text interpreter**.

6.1.2060 **R>** "r-from" CORE

Interpretation: Interpretation semantics for this word are undefined.

Execution: (-- *x*) (R: *x* --)

Move *x* from the return stack to the data stack.

See: **3.2.3.3 Return stack**, **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

Interpretation: Interpretation semantics for this word are undefined.

Execution: (-- *x*) (R: *x* -- *x*)

Copy *x* from the return stack to the data stack.

See: **3.2.3.3 Return stack**, **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

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (--)

Append the execution semantics of the current definition to the current definition. An ambiguous condition exists if **RECURSE** appears in a definition after **DOES>**.

See: **6.1.1250 DOES>**, **6.1.2120 RECURSE**.

6.1.2140 **REPEAT** CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: *orig dest* --)

Append the run-time semantics given below to the current definition, resolving the backward reference *dest*. Resolve the forward reference *orig* using the location following the appended execution semantics.

Run-time: (--)

Continue execution at the location given by *dest*.

See: **6.1.0760 BEGIN**, **6.1.2430 WHILE**.

6.1.2160 **ROT** "rote" CORE

(x_1 x_2 x_3 -- x_2 x_3 x_1)

Rotate the top three stack entries.

6.1.2162 **RSHIFT** "r-shift" CORE

(x_1 *u* -- x_2)

Perform a logical right shift of *u* bit-places on x_1, giving x_2. Put zero into the most significant bits vacated by the shift. An ambiguous condition exists if *u* is greater than or equal to the number of bits in a cell.

6.1.2165 **S"** "s-quote" CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: ("*ccc*<*quote*>" --)

Parse *ccc* delimited by **"** (double-quote). Append the run-time semantics given below to the current definition.

Run-time: (-- *c-addr u*)

Return *c-addr* and *u* describing a string consisting of the characters *ccc*. A program shall not alter the returned string.

See: **3.4.1 Parsing**, **6.2.0855 C"**, **11.6.1.2165 S"**.

6.1.2170 **S>D** "s-to-d" CORE

(*n* -- *d*)

Convert the number *n* to the double-cell number *d* with the same numerical value.

6.1.2210 **SIGN** CORE

(*n* --)

If *n* is negative, add a minus sign to the beginning of the pictured numeric output string. An ambiguous condition exists if **SIGN** executes outside of a **<# #>** delimited number conversion.

6.1.2214 **SM/REM** "s-m-slash-rem" CORE

(d_1 n_1 -- n_2 n_3)

Divide d_1 by n_1, giving the symmetric quotient n_3 and the remainder n_2. Input and output stack arguments are signed. An ambiguous condition exists if n_1 is zero or if the quotient lies outside the range of a single-cell signed integer.

See: **3.2.2.1 Integer division**, **6.1.1561**, **FM/MOD**, **6.1.2370**, **UM/MOD**.

6.1.2216 **SOURCE** CORE

(-- *c-addr u*)

c-addr is the address of, and *u* is the number of characters in, the input buffer.

6.1.2220 **SPACE** CORE

(--)

Display one space.

6.1.2230 **SPACES** CORE

(*n* --)

If *n* is greater than zero, display *n* spaces.

6.1.2250 **STATE** CORE

(-- *a-addr*)

a-addr is the address of a cell containing the compilation-state flag. **STATE** is true when in compilation state, false otherwise. The true value in **STATE** is non-zero, but is otherwise implementation-defined. Only the following standard words alter the value in **STATE**: **:** (colon), **;** (semicolon), **ABORT**, **QUIT**, **:NONAME**, **[** (left-bracket), and **]** (right-bracket).

Note: A program shall not directly alter the contents of **STATE**.

See: **3.4 The Forth text interpreter**, **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

(x_1 x_2 -- x_2 x_1)

Exchange the top two stack items.

6.1.2270 **THEN** CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: *orig* --)

Resolve the forward reference *orig* using the location of the execution semantics. Append the run-time semantics given below to the current definition.

Run-time: (--)

Continue execution.

See: **6.1.1310 ELSE**, **6.1.1700 IF**.

6.1.2310 **TYPE** CORE

(*c-addr u* --)

If *u* is greater than zero, display the character string specified by *c-addr* and *u*.

When passed a character in a character string whose character-defining bits have a value between hex 20 and 7E inclusive, the corresponding standard character, specified by **3.1.2.1 graphic characters**, is displayed. Because different output devices can respond differently to control characters, programs that use control characters to perform specific functions have an environmental dependency.

See: **6.1.1320 EMIT**.

6.1.2320 **U.** "u-dot" CORE

(*u* --)

Display *u* in free field format.

6.1.2340 **U<** "u-less-than" CORE

(u_1 u_2 -- *flag*)

flag is true if and only if u_1 is less than u_2.

See: **6.1.0480 <**.

6.1.2360 **UM*** "u-m-star" CORE

(u_1 u_2 -- *ud*)

Multiply u_1 by u_2, giving the unsigned double-cell product *ud*. All values and arithmetic are unsigned.

6.1.2370 **UM/MOD** "u-m-slash-mod" CORE

(*ud* u_1 -- u_2 u_3)

Divide *ud* by u_1, giving the quotient u_3 and the remainder u_2. All values and arithmetic are unsigned. An ambiguous condition exists if u_1 is zero or if the quotient lies outside the range of a single-cell unsigned integer.

See: **3.2.2.1 Integer division**, **6.1.1561 FM/MOD**, **6.1.2214 SM/REM**.

6.1.2380 **UNLOOP** CORE

Interpretation: Interpretation semantics for this word are undefined.

Execution: (--) (R: *loop-sys* --)

Discard the loop-control parameters for the current nesting level. An **UNLOOP** is required for each nesting level before the definition may be **EXIT**ed. An ambiguous condition exists if the loop-control parameters are unavailable.

See: **3.2.3.3 Return stack**.

6.1.2390 **UNTIL** CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: *dest* --)

Append the run-time semantics given below to the current definition, resolving the backward reference *dest*.

Run-time: (*x* --)

If all bits of *x* are zero, continue execution at the location specified by *dest*.

See: **6.1.0760 BEGIN**.

6.1.2410 **VARIABLE** CORE

("<*spaces*>*name*" --)

Skip leading space delimiters. Parse *name* delimited by a space. Create a definition for *name* with the execution semantics defined below. Reserve one cell of data space at an aligned address.

name is referred to as a "variable".

name Execution: (-- *a-addr*)

a-addr is the address of the reserved cell. A program is responsible for initializing the contents of the reserved cell.

See: **3.4.1 Parsing**.

6.1.2430 **WHILE** CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: *dest* -- *orig dest*)

Put the location of a new unresolved forward reference *orig* onto the control flow stack, under the existing *dest*. Append the run-time semantics given below to the current definition. The semantics are incomplete until *orig* and *dest* are resolved (e.g., by **REPEAT**).

Run-time: (*x* --)

If all bits of *x* are zero, continue execution at the location specified by the resolution of *orig*.

6.1.2450 **WORD** CORE

(*char* "<*chars*>*ccc*<*char*>" -- *c-addr*)

Skip leading delimiters. Parse characters *ccc* delimited by *char*. An ambiguous condition exists if the length of the parsed string is greater than the implementation-defined length of a counted string.

c-addr is the address of a transient region containing the parsed word as a counted string. If the parse area was empty or contained no characters other than the delimiter, the resulting string has a zero length. A space, not included in the length, follows the string. A program may replace characters within the string.

Note: The requirement to follow the string with a space is obsolescent and is included as a concession to existing programs that use **CONVERT**. A program shall not depend on the existence of the space.

See: **3.3.3.6 Other transient regions**, **3.4.1 Parsing**.

6.1.2490 **XOR** "x-or" CORE

(x_1 x_2 -- x_3)

x_3 is the bit-by-bit exclusive-or of x_1 with x_2.

6.1.2500 **[** "left-bracket" CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: Perform the execution semantics given below.

Execution: (--)

Enter interpretation state. **[** is an immediate word.

See: **3.4 The Forth text interpreter**, **3.4.5 Compilation**, **6.1.2540]**.

6.1.2510 ['] "bracket-tick" CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: ("<*spaces*>*name*" --)

Skip leading space delimiters. Parse *name* delimited by a space. Find *name*. Append the run-time semantics given below to the current definition.

An ambiguous condition exists if *name* is not found.

Run-time: (-- *xt*)

Place *name*'s execution token *xt* on the stack. The execution token returned by the compiled phrase "**['] X**" is the same value returned by "**' X**" outside of compilation state.

See: **3.4.1 Parsing**, **A.6.1.0070 '**, **A.6.1.2033 POSTPONE**, **D.6.7 Immediacy**.

6.1.2520 [CHAR] "bracket-char" CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: ("<*spaces*>*name*" --)

Skip leading space delimiters. Parse *name* delimited by a space. Append the run-time semantics given below to the current definition.

Run-time: (-- *char*)

Place *char*, the value of the first character of *name*, on the stack.

See: **3.4.1 Parsing**, **6.1.0895 CHAR**.

6.1.2540] "right-bracket" CORE

(--)

Enter compilation state.

See: **3.4 The Forth text interpreter**, **3.4.5 Compilation**, **6.1.2500 [**.

6.2 Core extension words

6.2.0060 #TIB "number-t-i-b" CORE EXT

(-- *a-addr*)

a-addr is the address of a cell containing the number of characters in the terminal input buffer.

Note: This word is obsolescent and is included as a concession to existing implementations.

6.2.0200 .("dot-paren" CORE EXT

Compilation: Perform the execution semantics given below.

Execution: ("*ccc*<*paren*>" --)

Parse and display *ccc* delimited by a right parenthesis "**)**". **.(** is an immediate word.

See: **3.4.1 Parsing**, **6.1.0190 ."**.

6.2.0210 .R "dot-r" CORE EXT

(n_1 n_2 --)

Display n_1 right aligned in a field n_2 characters wide. If the number of characters required to display n_1 is greater than n_2, all digits are displayed with no leading spaces in a field as wide as necessary.

6.2.0260 0<> "zero-not-equals" CORE EXT

(*x* -- *flag*)

flag is true if and only if *x* is not equal to zero.

6.2.0280 0> "zero-greater" CORE EXT

(*n* -- *flag*)

flag is true if and only if *n* is greater than zero.

6.2.0340 2>R "two-to-r" CORE EXT

Interpretation: Interpretation semantics for this word are undefined.

Execution: (x_1 x_2 --) (R: -- x_1 x_2)

Transfer cell pair x_1 x_2 to the return stack. Semantically equivalent to **SWAP >R >R**.

See: **3.2.3.3 Return stack**, **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

Interpretation: Interpretation semantics for this word are undefined.

Execution: (-- x_1 x_2) (R: x_1 x_2 --)

Transfer cell pair x_1 x_2 from the return stack. Semantically equivalent to **R> R> SWAP**.

See: **3.2.3.3 Return stack**, **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

Interpretation: Interpretation semantics for this word are undefined.

Execution: (-- x_1 x_2) (R: x_1 x_2 -- x_1 x_2)

Copy cell pair x_1 x_2 from the return stack. Semantically equivalent to **R> R> 2DUP >R >R SWAP**.

See: **3.2.3.3 Return stack**, **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*)

Create an execution token *xt*, enter compilation state and start the current definition, producing *colon-sys*. Append the initiation semantics given below to the current definition.

The execution semantics of *xt* will be determined by the words compiled into the body of the definition. This definition can be executed later by using *xt* **EXECUTE**.

If the control-flow stack is implemented using the data stack, *colon-sys* shall be the topmost item on the data stack. See **3.2.3.2 Control-flow stack**.

Initiation: (*i*x* -- *i*x*) (R: -- *nest-sys*)

Save implementation-dependent information *nest-sys* about the calling definition. The stack effects *i*x* represent arguments to *xt*.

xt Execution: (*i*x* -- *j*x*)

Execute the definition specified by *xt*. The stack effects *i*x* and *j*x* represent arguments to and results from *xt*, respectively.

6.2.0500 <> "not-equals" CORE EXT

(x_1 x_2 -- *flag*)

flag is true if and only if x_1 is not bit-for-bit the same as x_2.

6.2.0620 ?DO "question-do" CORE EXT

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: -- *do-sys*)

Put *do-sys* onto the control-flow stack. Append the run-time semantics given below to the current definition. The semantics are incomplete until resolved by a consumer of *do-sys* such as **LOOP**.

Run-time: ($n_1|u_1$ $n_2|u_2$ --) (R: -- | *loop-sys*)

If $n_1|u_1$ is equal to $n_2|u_2$, continue execution at the location given by the consumer of *do-sys*. Otherwise set up loop control parameters with index $n_2|u_2$ and limit $n_1|u_1$ and continue executing immediately following **?DO**. Anything already on the return stack becomes unavailable until the loop control parameters are discarded. An ambiguous condition exists if $n_1|u_1$ and $n_2|u_2$ are not both of the same type.

See: **3.2.3.2 Control-flow stack**, **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

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: *dest* --)

Append the run-time semantics given below to the current definition, resolving the backward reference *dest*.

Run-time: (--)

Continue execution at the location specified by *dest*. If no other control flow words are used, any program code after **AGAIN** will not be executed.

See: **6.1.0760 BEGIN**.

6.2.0855 C" "c-quote" CORE EXT

Interpretation: Interpretation semantics for this word are undefined.

Compilation: ("*ccc*<*quote*>" --)

Parse *ccc* delimited by **"** (double-quote) and append the run-time semantics given below to the current definition.

Run-time: (-- *c-addr*)

Return *c-addr*, a counted string consisting of the characters *ccc*. A program shall not alter the returned string.

See: **3.4.1 Parsing**, **6.1.2165 S"**, **11.6.1.2165 S"**.

6.2.0873 **CASE** CORE EXT

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: -- *case-sys*)

Mark the start of the **CASE ... OF ... ENDOF ... ENDCASE** structure. Append the run-time semantics given below to the current definition.

Run-time: (--)

Continue execution.

See: **6.2.1342 ENDCASE**, **6.2.1343 ENDOF**, **6.2.1950 OF**.

6.2.0945 **COMPILE,** "compile-comma" CORE EXT

Interpretation: Interpretation semantics for this word are undefined.

Execution: (*xt* --)

Append the execution semantics of the definition represented by *xt* to the execution semantics of the current definition.

6.2.0970 **CONVERT** CORE EXT

(ud_1 $c\text{-}addr_1$ -- ud_2 $c\text{-}addr_2$)

ud_2 is the result of converting the characters within the text beginning at the first character after $c\text{-}addr_1$ into digits, using the number in **BASE**, and adding each digit to ud_1 after multiplying ud_1 by the number in **BASE**. Conversion continues until a character that is not convertible is encountered. $c\text{-}addr_2$ is the location of the first unconverted character. An ambiguous condition exists if ud_2 overflows.

Note: This word is obsolescent and is included as a concession to existing implementations. Its function is superseded by **6.1.0570 >NUMBER**.

See: **3.2.1.2 Digit conversion**.

6.2.1342 **ENDCASE** "end-case" CORE EXT

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: *case-sys* --)

Mark the end of the **CASE ... OF ... ENDOF ... ENDCASE** structure. Use *case-sys* to resolve the entire structure. Append the run-time semantics given below to the current definition.

Run-time: (*x* --)

Discard the case selector *x* and continue execution.

See: **6.2.0873 CASE**, **6.2.1343 ENDOF**, **6.2.1950 OF**.

6.2.1343 **ENDOF** "end-of" CORE EXT

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: *case-sys*$_1$ *of-sys* -- *case-sys*$_2$)

Mark the end of the **... OF ... ENDOF ...** part of the **CASE** structure. The next location for a transfer of control resolves the reference given by *of-sys*. Append the run-time semantics given below to the current definition. Replace *case-sys*$_1$ with *case-sys*$_2$ on the control-flow stack, to be resolved by **ENDCASE**.

Run-time: (--)

Continue execution at the location specified by the consumer of *case-sys*$_2$.

See: **6.2.0873 CASE**, **6.2.1342 ENDCASE**, **6.2.1950 OF**.

6.2.1350 **ERASE** CORE EXT

(*addr u* --)

If *u* is greater than zero, clear all bits in each of *u* consecutive address units of memory beginning at *addr* .

6.2.1390 **EXPECT** CORE EXT

(*c-addr +n* --)

Receive a string of at most *+n* characters. Display graphic characters as they are received. A program that depends on the presence or absence of non-graphic characters in the string has an environmental dependency. The editing functions, if any, that the system performs in order to construct the string of characters are implementation-defined.

Input terminates when an implementation-defined line terminator is received or when the string is *+n* characters long. When input terminates, nothing is appended to the string and the display is maintained in an implementation-defined way.

Store the string at *c-addr* and its length in **SPAN**.

Note: This word is obsolescent and is included as a concession to existing implementations. Its function is superseded by **6.1.0695 ACCEPT**.

6.2.1485 **FALSE** CORE EXT

(-- *false*)

Return a *false* flag.

See: **3.1.3.1 Flags**

6.2.1660 **HEX** CORE EXT

(--)

Set contents of **BASE** to sixteen.

6.2.1850 **MARKER** CORE EXT

("<*spaces*>*name*" --)

Skip leading space delimiters. Parse *name* delimited by a space. Create a definition for *name* with the execution semantics defined below.

name Execution: (--)

Restore all dictionary allocation and search order pointers to the state they had just prior to the definition of *name*. Remove the definition of *name* and all subsequent definitions. Restoration of any structures still existing that could refer to deleted definitions or deallocated data space is not necessarily provided. No other contextual information such as numeric base is affected.

See: **3.4.1 Parsing**, **15.6.2.1580 FORGET**.

6.2.1930 **NIP** CORE EXT

(x_1 x_2 -- x_2)

Drop the first item below the top of stack.

6.2.1950 **OF** CORE EXT

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: -- *of-sys*)

Put *of-sys* onto the control flow stack. Append the run-time semantics given below to the current definition. The semantics are incomplete until resolved by a consumer of *of-sys* such as **ENDOF**.

Run-time: (x_1 x_2 -- | x_1)

If the two values on the stack are not equal, discard the top value and continue execution at the location specified by the consumer of *of-sys*, e.g., following the next **ENDOF**. Otherwise, discard both values and continue execution in line.

See: **6.2.0873 CASE**, **6.2.1342 ENDCASE**, **6.2.1343 ENDOF**.

6.2.2000 **PAD** CORE EXT

(-- *c-addr*)

c-addr is the address of a transient region that can be used to hold data for intermediate processing.

See: **3.3.3.6 Other transient regions**.

6.2.2008 **PARSE** CORE EXT

(*char* "*ccc*<*char*>" -- *c-addr u*)

Parse *ccc* delimited by the delimiter *char*.

c-addr is the address (within the input buffer) and *u* is the length of the parsed string. If the parse area was empty, the resulting string has a zero length.

See: **3.4.1 Parsing**.

6.2.2030 **PICK** CORE EXT

(x_u ... x_1 x_0 *u* -- x_u ... x_1 x_0 x_u)

Remove *u*. Copy the x_u to the top of the stack. An ambiguous condition exists if there are less than *u*+2 items on the stack before **PICK** is executed.

6.2.2040 **QUERY** CORE EXT

(--)

Make the user input device the input source. Receive input into the terminal input buffer, replacing any previous contents. Make the result, whose address is returned by **TIB**, the input buffer. Set **>IN** to zero.

Note: This word is obsolescent and is included as a concession to existing implementations.

6.2.2125 **REFILL** CORE EXT

(-- *flag*)

Attempt to fill the input buffer from the input source, returning a true flag if successful.

When the input source is the user input device, attempt to receive input into the terminal input buffer. If successful, make the result the input buffer, set **>IN** to zero, and return *true*. Receipt of a line containing no characters is considered successful. If there is no input available from the current input source, return *false*.

When the input source is a string from **EVALUATE**, return *false* and perform no other action.

See: **7.6.2.2125 REFILL**, **11.6.2.2125 REFILL**.

6.2.2148 **RESTORE-INPUT** CORE EXT

(x_n ... x_1 *n* -- *flag*)

Attempt to restore the input source specification to the state described by x_1 through x_n. *flag* is true if the input source specification cannot be so restored.

An ambiguous condition exists if the input source represented by the arguments is not the same as the current input source.

See: **A.6.2.2182 SAVE-INPUT**.

6.2.2150 **ROLL** CORE EXT

(x_u x_{u-1} ... x_0 u -- x_{u-1} ... x_0 x_u)

Remove u. Rotate u+1 items on the top of the stack. An ambiguous condition exists if there are less than u+2 items on the stack before **ROLL** is executed.

6.2.2182 **SAVE-INPUT** CORE EXT

(-- x_n ... x_1 n)

x_1 through x_n describe the current state of the input source specification for later use by **RESTORE-INPUT**.

6.2.2218 **SOURCE-ID** "source-i-d" CORE EXT

(-- 0 | -1)

Identifies the input source as follows:

SOURCE-ID	Input source
-1	String (via **EVALUATE**)
0	User input device

See: **11.6.1.2218 SOURCE-ID**.

6.2.2240 **SPAN** CORE EXT

(-- *a-addr*)

a-addr is the address of a cell containing the count of characters stored by the last execution of **EXPECT**.

Note: This word is obsolescent and is included as a concession to existing implementations.

6.2.2290 **TIB** "t-i-b" CORE EXT

(-- *c-addr*)

c-addr is the address of the terminal input buffer.

Note: This word is obsolescent and is included as a concession to existing implementations.

6.2.2295 **TO** CORE EXT

Interpretation: (*x* "<*spaces*>*name*" --)

Skip leading spaces and parse *name* delimited by a space. Store *x* in *name*. An ambiguous condition exists if *name* was not defined by **VALUE**.

Compilation: ("<*spaces*>*name*" --)

Skip leading spaces and parse *name* delimited by a space. Append the run-time semantics given below to the current definition. An ambiguous condition exists if *name* was not defined by **VALUE**.

Run-time: (*x* --)

Store *x* in *name*.

Note: An ambiguous condition exists if either **POSTPONE** or **[COMPILE]** is applied to **TO**.

See: **6.2.2405 VALUE**, **13.6.1.2295 TO**.

6.2.2298 **TRUE** CORE EXT

(-- *true*)

Return a *true* flag, a single-cell value with all bits set.

See: **3.1.3.1 Flags**.

6.2.2300 **TUCK** CORE EXT

(x_1 x_2 -- x_2 x_1 x_2)

Copy the first (top) stack item below the second stack item.

6.2.2330 **U.R** "u-dot-r" CORE EXT

(*u n* --)

Display *u* right aligned in a field *n* characters wide. If the number of characters required to display *u* is greater than *n*, all digits are displayed with no leading spaces in a field as wide as necessary.

6.2.2350 **U>** "u-greater-than" CORE EXT

(u_1 u_2 -- *flag*)

flag is true if and only if u_1 is greater than u_2.

See: **6.1.0540 >**.

6.2.2395 **UNUSED** CORE EXT

(-- *u*)

u is the amount of space remaining in the region addressed by **HERE** , in address units.

6.2.2405 VALUE CORE EXT

(*x* "<*spaces*>*name*" --)

Skip leading space delimiters. Parse *name* delimited by a space. Create a definition for *name* with the execution semantics defined below, with an initial value equal to *x*.

name is referred to as a "value".

name Execution: (-- *x*)

Place *x* on the stack. The value of *x* is that given when *name* was created, until the phrase **x TO name** is executed, causing a new value of *x* to be associated with *name*.

See: **3.4.1 Parsing**.

6.2.2440 WITHIN CORE EXT

($n_1|u_1$ $n_2|u_2$ $n_3|u_3$ -- *flag*)

Perform a comparison of a test value $n_1|u_1$ with a lower limit $n_2|u_2$ and an upper limit $n_3|u_3$, returning *true* if either ($n_2|u_2$<$n_3|u_3$ and ($n_2|u_2$<=$n_1|u_1$ and $n_1|u_1$<$n_3|u_3$)) or ($n_2|u_2$>$n_3|u_3$ and ($n_2|u_2$<=$n_1|u_1$ or $n_1|u_1$<$n_3|u_3$)) is true, returning *false* otherwise. An ambiguous condition exists if $n_1|u_1$, $n_2|u_2$, and $n_3|u_3$ are not all the same type.

6.2.2530 [COMPILE] "bracket-compile" CORE EXT

Interpretation: Interpretation semantics for this word are undefined.

Compilation: ("<*spaces*>*name*" --)

Skip leading space delimiters. Parse *name* delimited by a space. Find *name*. If *name* has other than default compilation semantics, append them to the current definition; otherwise append the execution semantics of *name*. An ambiguous condition exists if *name* is not found.

See: **3.4.1 Parsing**.

6.2.2535 \ "backslash" CORE EXT

Compilation: Perform the execution semantics given below.

Execution: ("*ccc*<*eol*>"--)

Parse and discard the remainder of the parse area. **** is an immediate word.

See: **7.6.2.2535 **.