07.2 Included changes approved at the Dagstuhl meeting, 13–14 September, 2007:

- LaTeX source file now given in page footer.
- Rename "forward" to "foreword".
- **Proposals Process**:
  Two new sentences (ed07).
- **200*x* Membership**:
  Membership Rules (ed07).

2 Terms, notation, and references:
X:number-prefix: Extended section **2.2.1 Numeric notation**.

3 Usage Requirements:

a) Added X:fp-stack, X:number-prefix, X:structures and X:throw-iors to table **3.6 Forth 200*x* Extensions**.

b) X:number-prefix: replaced section **3.4.1.3 Text interpreter input number conversion**.

4 Documentation requirements:
Added ambiguous condition for X:structures.

6 Glossary:

a) Inline text is now in sans-serif

b) Removed state-smart note from **6.2.2295** `TO`.

8 Double-Number word set:
X:number-prefix: added ", except a ⟨*cnum*⟩," to 8.3.2 Text interpreter input number conversion.

9 Exception word set:
Added X:throw-iors to table **9.2 `THROW` code assignments**.

10 Facility word set:

a) Added X:structures: **10.6.2.0** `BEGIN-STRUCTURE`, **10.6.2.0** `END-STRUCTURE`, **10.6.2.0** `+FIELD`, **10.6.2.0** `CFIELD:` and **10.6.2.0** `FIELD:`.

b) Tidied up source code of X:ekeys definitions.

c) ed07: Added **`EKEY`** ratioanle.

12 Floating word set:

a) X:structures: Added **12.6.2.0** `DFFIELD:`, **12.6.2.0** `FFIELD:`, and **12.6.2.0** `SFFIELD:`.

b) X:fp-stack: A separate floating point stack is now the default, with a combined stack being an environmental dependency.

c) X:number-prefix: Removed section 12.2.2.1 Numeric notation, now in section **2.2.1 Numeric notation**.

A Rationale:

a) Corrected a number of section numbering errors

b) X:number-prefix: Added test cases (**A.3.4.1.3 Numeric notation**).

B Bibliography (Annex B):

a) ed07: Added books by Elizabeth Rather and Stephen Pelc.

b) ed07: Added URL for Thinking Forth.

Foreword

On completion of ANS Forth (ANS X3.215-1994 *Information Systems — Programming Languages FORTH*) in 1994, the document was presented to and adopted as an international standard, by the ISO in 1997, being published as ISO/IEC 15145:1997 *Information technology, Programming languages, FORTH*.

The current project to update ANS Forth was launched at the 2004 EuroForth conference. The intention being to allow the Forth community to contribute to a rolling standard. With changes to the document being proposed and discussed in the electronic community, via the `comp.lang.forth` usenet news group, the `forth200x@yahoogroups.com` email list, and the `www.forth200x.org` web site. An ed07 open meeting to discuss proposals ~~being~~is held annually, immediately prior to the EuroForth conference.

This document is based on the first draft of the of the standard published by Technical Committee on Forth Programming Systems as part of the first review in 1999. It has been modified in accordance with the directions of the Forth 200*x* Standards Committee which first met on October 21-22, 2005 (Santander) and subsequently on September 14-15, 2006 (Cambridge), September 13–14, 2007 (Dagstuhl).

Proposals Process

In developing a standard it is necessary for the standards committee to know what the system implementors and the programmers are already doing in that area, and what they would be willing to do, or wish for.

To that end we have introduced a system of consultation with the Forth community:

a) A proponent of an extension or change to the standard writes a proposal.

b) The proponent publishes the proposal as an *RfD* (Request for Discussion) by sending a copy to the `forth200x@~~yhoo~~yahoo groups.com` email list ~~and~~*and* to the `comp.lang.forth` usenet news group where it can be discussed. The maintainers of the `www.forth200x.org` web site will then place a copy of the proposal on that web site.

   Be warned, this will generate a lot of heated discussion.

   In order for the results to be available in time for a standards meeting, an RfD should be published at least 12 weeks before the next meeting.

c) The proponent can modify the proposal, taking any comments into consideration. Where comments have been dismiss, both the comment and the reasons for its dismissal should be given. The revised proposal is published as a revised RfD.

d) Once a proposal has settled down, it is frozen, and submitted to a vote taker, who then publishes a *CfV* (Call for Votes) on the proposal. The vote taker will normally be a member of the standards committee. In the poll, system implementors can state, whether their systems implement the proposal, or what the chances are that it ever will. Similarly, programmers can state whether they have used something similar to the proposed extension and whether they would use the proposed extension once it is standardized. The results of this poll are used by the standards committee when deciding whether to accept the proposal into the standards document.

   In order for the results to be available in time for a standards meeting, the CfV should be started at least 6 weeks before that meeting.

   If a proposal does not propose extensions or changes to the Forth language, but just a rewording of the current document, there is nothing for the system implementors to implement, and for programmers to use, so a CfV poll does not make sense and is not performed. The proposal will bypass the CfV stage and will simply be frozen and go directly to the committee for consideration.

e) One to two weeks after publishing the CfV, the vote taker will publish a *Current Standings*. Note that the poll will remain open, especially for information on additional systems, and the results will be updated on the Forth200*x* web page. The results considered at a standards meeting are those from four weeks prior to that meeting. If no poll results are available by that deadline, the proposal will be considered at a later meeting.

f) A proposal will only be accepted into the new basis document by consensus of those present at an ~~open~~ standards meeting. If you can not attend a meeting, you should ask somebody who is attending to champion the proposal on your behalf.

Should a contributor consider their comments to have been dismissed without due consideration, they are encouraged to submit a counter proposal.

Proposals which have passed the poll will be integrated into the basis document in preparation for the approaching Standards Committee meeting. Proposals often require some rewording in this process, so the proponent should work with the editor to integrate the proposal into the document.

A proposal should give a rationale for the proposal, so that system implementors and programmers will see what it is good for and why they should adopt it (and vote for it).

200*x* Membership

This document is maintained by the Forth 200*x* Standards Committee. The committee meetings are open to the public, anybody is allowed to join the committee in its deliberations. Currently the committee has the following voting members:

ed07 Membership of the committee is open to anybody who can attend. On attending a meeting of any kind a non-member becomes an observing member (observer). If they attend the next voting meeting, they will become a voting member of the committee, otherwise they revert to non-member status. An observer will not normally be allowed to vote, but may be allowed at the discretion of the committee. A member will be deemed to have resigned from the committee if they fail to attend two consecutive voting meetings.

M. Anton Ertl (Chair) ..... Technische Universität Wien
anton@mips.complang.tuwien.at Wien, Austria

Dr. Peter Knaggs (Editor) ..... Bournemouth University
pknaggs@bournemouth.ac.uk Bournemouth, UK

Willem Botha ..... Construction Computer Software (Pty) Ltd
willem.botha@ccssa.com Cape Town, South Africa

Federico de Ceballos ..... Universidad de Cantabria
federico.ceballos@unican.es Santander, Spain

Stephen Pelc ..... MicroProcessor Engineering Ltd.
stephen@mpeforth.com Southampton, UK

Dr. Bill Stoddart ..... University of Teesside
bill.stoddart@ntlworld.com Middlesbrough, UK

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}}**.

x:number-prefix

The following notation is used to define the syntax of the external representation of numbers:

- Each component of a number is defined with a rule consisting of the name of the component (italicized in angle-brackets, e.g., ⟨*decdigit*⟩), the characters := and a concatenation of tokens and metacharacters;
- Tokens may be literal characters (in bold face, e.g., **E**) or rule names in angle brackets (e.g., ⟨*decdigit*⟩);

- The metacharacter * is used to specify zero or more occurrences of the preceding token (e.g., ⟨*decdigit*⟩*);
- Tokens enclosed with [and] are optional (e.g., [**-**]);
- Vertical bars separate choices from a list of tokens enclosed with braces (e.g., { **0** | **1** }).

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 3.1.

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 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:

- 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?**.

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.

3.2.7 Extension queries

X:extension-query

ed07 As part of the Forth 200*x* standards procedure, additions to the Standard ~~where~~are labelled as *extensions*. These extensions have been added to the environmental query name space. **ENVIRONMENT?** a *true* if the system has implemented the extension as documented. Table 3.6 contains the valid input strings corresponding to the documented extensions. In order to distinguish such extensions, they start with the string "X:".

The extension to the environment query table (3.2.6) is itself an extension. Known as the X:extension-query extension.

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.

Table 3.6: Forth 200*x* Extensions

String	Value data type	Constant?	Meaning
`X:deferred`	–	–	the X:deferred extension is present
`X:defined`	–	–	the X:defined extension is present
`X:ekeys`	–	–	the X:ekeys extension is present
`X:extension-query`	–	–	the X:extension-query extension is present
`X:fp-stack`	–	–	X;fp-stack extension is present
`X:number-prefix`	–	–	X:number-prefix extension is present
`X:parse-name`	–	–	the X:parse-name extension is present
`X:required`	–	–	the X:required extension is present
`X:structures`	–	–	the X:structures extension is present
`X:throw-iors`	–	–	the X:throw-iors extension is present

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.

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 A (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

x:number-prefix

~~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.~~

When converting input numbers, the text interpreter shall recognize integer numbers in the form ⟨*BASEnum*⟩; if the X:number-prefixes extension is present, the text interpreter shall recognize integer numbers in the form ⟨*anynum*⟩.

⟨*anynum*⟩ := { ⟨*BASEnum*⟩ | ⟨*decnum*⟩ | ⟨*hexnum*⟩ | ⟨*binnum*⟩ | ⟨*cnum*⟩ }
⟨*BASEnum*⟩ := [**-**]⟨*bdigit*⟩⟨*bdigit*⟩*
⟨*decnum*⟩ := **#**[**-**]⟨*decdigit*⟩⟨*decdigit*⟩*
⟨*hexnum*⟩ := **$**[**-**]⟨*hexdigit*⟩⟨*hexdigit*⟩*
⟨*binnum*⟩ := **%**[**-**]⟨*bindigit*⟩⟨*bindigit*⟩*
⟨*cnum*⟩ := **'**⟨*char*⟩**'**
⟨*bindigit*⟩ := { **0** | **1** }
⟨*decdigit*⟩ := { **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** }
⟨*hexdigit*⟩ := { ⟨*decdigit*⟩ | **a** | **b** | **c** | **d** | **e** | **f** | **A** | **B** | **C** | **D** | **E** | **F** }

⟨*bdigit*⟩ represents a digit according to the value of **BASE** (see **3.2.1.2** Digit conversion). For ⟨*hexdigit*⟩, the digits **a**…**f** have the values 10…15. ⟨*char*⟩ represents any printable character.

The radix used for number conversion is:

⟨*BASEnum*⟩	the value in **BASE**
⟨*decnum*⟩	10
⟨*hexnum*⟩	16
⟨*binnum*⟩	2
⟨*cnum*⟩	the number is the value of ⟨*char*⟩

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**.

- access to a deferred word, a word defined by **6.2.0** DEFER, which was not defined by **6.2.0** DEFER.
- **6.1.2033** POSTPONE, **6.2.2530** [COMPILE], **6.1.2510** ['] or **6.1.0070** ' applied to **6.2.0** ACTION-OF or **6.2.0** IS.

X:required

- a file is required while it is being **REQUIRED** (11.6.2.0) or **INCLUDED** (11.6.1.1718).
- a marker is defined outside and executed inside a file or vice versa, and the file is **REQUIRED** (11.6.2.0) again.
- the same file is required twice using different names (e.g., through symbolic links), or different files with the same name are provided to **11.6.2.0** REQUIRED (by doing some renaming between the invocations of **REQUIRED**).
- the stack effect of including with **11.6.2.0** REQUIRED the file is not ($i \times x - i \times x$) .

x:structures

X:structures

- A *name* defined by **10.6.2.0** BEGIN-STRUCTURE is executed before the coresponding **10.6.2.0** END-STRUCTURE has been executed.

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**);

See: **6.2.0** `ACTION-OF`, **6.2.0** `DEFER`, **6.2.0** `DEFER!`, and **6.2.0** `DEFER@`.

Implementation:
```
: IS
STATE @ IF
   POSTPONE ['] POSTPONE DEFER!
ELSE
   ' DEFER!
THEN ; IMMEDIATE
```

Testing:
```
{ DEFER defer5 -> }
{ : is-defer5 IS defer5 ; -> }
{ ' * IS defer5 -> }
{ 2 3 defer5 -> 6 }
{ ' + is-defer5 -> }
{ 1 2 defer5 -> 3 }
```

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`.

ed06 Rationale: As dictionary implementations have ~~gotten~~become more elaborate and in some cases have used multiple address spaces, **FORGET** has become prohibitively difficult or impossible to implement on many Forth systems. **MARKER** greatly eases the problem by making it possible for the system to remember "landmark information" in advance that specifically marks the spots where the dictionary may at some future time have to be rearranged.

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**.

Table 9.2: **THROW** code assignments

Code	Reserved for	Code	Reserved for
-1	**ABORT**	-39	unexpected end of file
-2	**ABORT"**	-40	invalid **BASE** for floating point conversion
-3	stack overflow	-41	loss of precision
-4	tack underflow	-42	floating-point divide by zero
-5	return stack overflow	-43	floating-point result out of range
-6	return stack underflow	-44	floating-point stack overflow
-7	do-loops nested too deeply during execution	-45	floating-point stack underflow
-8	dictionary overflow	-46	floating-point invalid argument
-9	invalid memory address	-47	compilation word list deleted
-10	division by zero	-48	invalid **POSTPONE**
-11	result out of range	-49	search-order overflow
-12	argument type mismatch	-50	search-order underflow
-13	undefined word	-51	compilation word list changed
-14	interpreting a compile-only word	-52	control-flow stack overflow
-15	invalid **FORGET**	-53	exception stack overflow
-16	attempt to use zero-length string as a name	-54	floating-point underflow
-17	pictured numeric output string overflow	-55	floating-point unidentified fault
-18	parsed string overflow	-56	**QUIT**
-19	definition name too long	-57	exception in sending or receiving a character
-20	write to a read-only location	-58	**[IF]**, **[ELSE]**, or **[THEN]** exception
-21	unsupported operation	-59	**ALLOCATE**
	(e.g., **AT-XY** on a too-dumb terminal)	-60	**FREE**
-22	control structure mismatch	-61	**RESIZE**
-23	address alignment exception	-62	**CLOSE-FILE**
-24	invalid numeric argument	-63	**CREATE-FILE**
-25	return stack imbalance	-64	**DELETE-FILE**
-26	loop parameters unavailable	-65	**FILE-POSITION**
-27	invalid recursion	-66	**FILE-SIZE**
-28	user interrupt	-67	**FILE-STATUS**
-29	compiler nesting	-68	**FLUSH-FILE**
-30	obsolescent feature	-69	**OPEN-FILE**
-31	**>BODY** used on non-**CREATE**d definition	-70	**READ-FILE**
-32	invalid name argument (e.g., **TO** xxx)	-71	**READ-LINE**
-33	block read exception	-72	**RENAME-FILE**
-34	block write exception	-73	**REPOSITION-FILE**
-35	invalid block number	-74	**RESIZE-FILE**
-36	invalid file position	-75	**WRITE-FILE**
-37	file I/O exception	-76	**WRITE-LINE**
-38	non-existent file		

9.3.6 Exception handling

There are several methods of coupling **CATCH** and **THROW** to other procedural nestings. The usual nestings are the execution of definitions, use of the return stack, use of loops, instantiation of locals and nesting of input sources (i.e., with **LOAD**, **EVALUATE**, or **INCLUDE-FILE**).

When a **THROW** returns control to a **CATCH**, the system shall un-nest not only definitions, but also, if present, locals and input source specifications, to return the system to its proper state for continued execution past the **CATCH**.

10 The optional Facility word set

10.1 Introduction

10.2 Additional terms and notation

None.

10.3 Additional usage requirements

10.3.1 Data types

X:structures

Append table 10.1 to table 3.1.

Table 10.1: Data types

Symbol	*Data type*	*Size on stack*
struct-sys	data structures	implementation dependent

10.3.1.1 Structure type

The implementation-dependent data generated and upon beginning to compile a data structure, with **BEGIN-STRUCTURE**, and consumed at its close, with **END-STRUCTURE**, is represented by the symbol *struct-sys* throughout this Standard.

10.3.1.2 Character types

Programs that use more than seven bits of a character by **EKEY** have an environmental dependency.

See: **3.1.2 Character types**.

10.3.2 Environmental queries

Append table 10.2 to table 3.5.

See: **3.2.6 Environmental queries**.

Table 10.2: Environmental Query Strings

String	Value data type	Constant?	Meaning
FACILITY	*flag*	no	facility word set present
FACILITY-EXT	*flag*	no	facility extensions word set present

10.4 Additional documentation requirements

10.4.1 System documentation

10.4.1.1 Implementation-defined options

- encoding of keyboard events **10.6.2.1305** EKEY);
- duration of a system clock tick;

– repeatability to be expected from execution of **10.6.2.1905** `MS`.

10.4.1.2 Ambiguous conditions

– **10.6.1.0742** `AT-XY` operation can't be performed on user output device.

– A *name* defined by **10.6.2.0** `BEGIN-STRUCTURE` is executed before the corresponding **10.6.2.0** `END-STRUCTURE` has been executed. X:structure

10.4.1.3 Other system documentation

– no additional requirements.

10.4.2 Program documentation

10.4.2.1 Environmental dependencies

– using more than seven bits of a character in **10.6.2.1305** `EKEY`.

10.4.2.2 Other program documentation

– no additional requirements.

10.5 Compliance and labeling

10.5.1 ANS Forth systems

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

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

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

10.5.2 ANS Forth programs

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

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

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

10.6 Glossary

10.6.1 Facility words

10.6.1.0742 `AT-XY` "at-x-y" FACILITY

(u_1 u_2 –)

Perform implementation-dependent steps so that the next character displayed will appear in column u_1, row u_2 of the user output device, the upper left corner of which is column zero, row zero. An ambiguous condition exists if the operation cannot be performed on the user output device with the specified parameters.

Rationale: Most implementors supply a method of positioning a cursor on a CRT screen, but there is great variance in names and stack arguments. This version is supported by at least one major vendor.

10.6.1.1755 KEY? "key-question" FACILITY

(– *flag*)

If a character is available, return *true*. Otherwise, return *false*. If non-character keyboard events are available before the first valid character, they are discarded and are subsequently unavailable. The character shall be returned by the next execution of **KEY**.

After **KEY?** returns with a value of *true*, subsequent executions of **KEY?** prior to the execution of **KEY** or **EKEY** also return *true*, without discarding keyboard events.

Rationale: The Technical Committee has gone around several times on the stack effects. Whatever is decided will violate somebody's practice and penalize some machine. This way doesn't interfere with type-ahead on some systems, while requiring the implementation of a single-character buffer on machines where polling the keyboard inevitably results in the destruction of the character.

Use of **KEY** or **KEY?** indicates that the application does not wish to bother with non-character events, so they are discarded, in anticipation of eventually receiving a valid character. Applications wishing to handle non-character events must use **EKEY** and **EKEY?**. It is possible to mix uses of **KEY?**/**KEY** and **EKEY?**/**EKEY** within a single application, but the application must use **KEY?** and **KEY** only when it wishes to discard non-character events until a valid character is received.

10.6.1.2005 PAGE FACILITY

(–)

Move to another page for output. Actual function depends on the output device. On a terminal, **PAGE** clears the screen and resets the cursor position to the upper left corner. On a printer, **PAGE** performs a form feed.

10.6.2 Facility extension words

10.6.2.—— +FIELD "plus-field" FACILITY EXT
X:structures

(n_1 n_2 "⟨*spaces*⟩*name*" – n_3)

Skip leading space delimiters. Parse *name* delimited by a space. Create a definition for *name* with the execution semantics defined below. Return $n_3 = n_1 + n_2$ where n_1 is the offset in the data structure before **+FIELD** executes, and n_2 is the size of the data to be added to the data structure. n_1 and n_2 are in address units.

name Execution: ($addr_1$ – $addr_2$)

Add n_1 to $addr_1$ giving $addr_2$.

See: **10.6.2.0** BEGIN-STRUCTURE, **10.6.2.0** END-STRUCTURE, **10.6.2.0** CFIELD:, **10.6.2.0** FIELD:, **12.6.2.0** FFIELD:, **12.6.2.0** SFFIELD: and **12.6.2.0** DFFIELD:

Rationale: **+FIELD** is not required to align items. This is deliberate and allows the construction of unaligned data structures for communication with external elements such as a hardware register map or protocol packet. Field alignment has been left to the appropriate *x*FIELD: definition.

Implementation: Create a new field within a structure definition of size *n* bytes.

```
:  +FIELD  \ n <"name"> - ; Exec:  addr - 'addr
  CREATE OVER , +
  DOES> @ +
;
```

10.6.2.—— BEGIN-STRUCTURE FACILITY EXT
X:structures

(*"⟨spaces⟩name" – struct-sys 0*)

Skip leading space delimiters. Parse *name* delimited by a space. Create a definition for *name* with the execution semantics defined below. Return a *struct-sys* (zero or more implementation dependent items) that will be used by **END-STRUCTURE** and an initial offset of 0.

name Execution: (– +*n*)

+*n* is the size in memory expressed in address units of the data structure. An ambiguous condition exists if *name* is executed prior to the associated **END-STRUCTURE** being executed.

See: **10.6.2.0** +FIELD and **10.6.2.0** END-STRUCTURE.

Rationale: There are two schools of though regarding named data structures: name first and name last. The name last school can define a named data structure as follows:

```
0                       \ initial total byte count
  1 CELLS +FIELD p.x    \ A single cell filed named p.x
  1 CELLS +FIELD p.y    \ A signle cell field named p.y
CONSTANT point          \ save strucutre size
```

While the name first school would define the same data structure as:

```
BEGIN-STRUCTURE point   \ create the named strucutre
  1 CELLS +FIELD p.x    \ A signle cell filed named p.x
  1 CELLS +FIELD p.y    \ A signle cell field named p.y
END-STRUCTURE
```

Although many systems provide a name first structure there is no common practice to the words used. The words **BEGIN-STRUCTURE** and **END-STRUCTURE** have been defied as a means of providing a portable notation that does not conflict with existing systems.

The field defining words (*x*FIELD: and **+FIELD**) are defined so they can be used by both schools. Compatibility between the two schools comes from defining a new stack item *struct-sys*, which is implementation dependent and can be 0 or more cells. The name first school would provide an address (*addr*) as the *struct-sys* parameter, while the name last school would defined *struct-sys* as being empty.

Executing the name of the data structure, returns the size of the data stricture. This allows the data stricture to be used within another data structure:

```
BEGIN-STRUCTURE point  \ - a-addr 0 ; - lenp
  FIELD: p.x           \ - a-addr cell
  FIELD: p.y           \ - a-addr cell*2
END-STRUCTURE

BEGIN-STRUCTURE rect   \ - a-addr 0 ; - lenr
  point +FIELD r.tlhc  \ - a-addr cell*2
  point +FIELD r.brhc  \ - a-addr cell*4
END-STRUCTURE
```

Alignment: In practice, structures are used for two different purposes with incompatible requirements:

a) For collecting related internal-use data into a convenient "package" that can be referred to by a single "handle". For this use, alignment is important, so that efficient native fetch and store instructions can be used.

b) For mapping external data structures like hardware register maps and protocol packets. For this use, automatic alignment is inappropriate, because the alignment of the external data structure often doesn't match the rules for a given processor.

Many languages cater for the first use, but ignore the second. This leads to various customized solutions, usage requirements, portability problems, bugs, etc. **+FIELD** is defined to be non-aligning, while the named field defining words (*x*`FIELD:`) are aligning. This is intentional and allows for both uses.

The standard currently defines an aligned field defining word for each of the standard data types:

CFIELD:	a character
FIELD:	a native integer (single cell)
FFIELD:	a native float
SFFIELD:	a 32 bit float
DFFIELD:	a 64 bit float

Although this is a sufficient set, most systems provide facilities to define field defining words for standard data types.

Future: The following cannot be defined until the required addressing has been defined. The names should be considered reserved until then.

`BFIELD:`	1 byte (8 bit) field
`WFIELD:`	16 bit field
`LFIELD:`	32 bit field
`XFIELD:`	64 bit field

Implementation: Begin definition of a new structure. Use in the form **BEGIN-STRUCTURE** ⟨*name*⟩. At run time ⟨*name*⟩ returns the size of the structure.

```
:  BEGIN-STRUCTURE  \ - addr 0 ; - size
  CREATE
    HERE 0 0 ,     \ mark stack, lay dummy
  DOES> @          \ - rec-len
;
```

10.6.2.—— `CFIELD:` "c-field-colon" FACILITY EXT
X:structures

(–)

The semantics of **`CFIELD:`** are identical to the execution semantics of the phrase:

```
1 CHARS +FIELD
```

See: **10.6.2.0** `+FIELD`, **10.6.2.0** `BEGIN-STRUCTURE` and **10.6.2.0** `END-STRUCTURE`.

Implementation: Create a new field within a structure definition of size a character.

```
: cfield: \ n1 <"name"> – n2 ; Exec: addr – 'addr
  1 CHARS +FIELD
;
```

10.6.2.1305 `EKEY` "e-key" FACILITY EXT

(– *u*)

Receive one keyboard event *u*. The encoding of keyboard events is implementation defined.

See: **10.6.1.1755** `KEY?`, **6.1.1750** `KEY`.

Rationale: **`EKEY`** provides a standard word to access a system-dependent set of "raw" keyboard events, including events corresponding to members of the standard character set, events corresponding to other members of the implementation-defined character set, and keystrokes that do not correspond to members of the character set.

`EKEY` assumes no particular numerical correspondence between particular event code values and the values representing standard characters. On some systems, this may allow two separate keys that correspond to the same standard character to be distinguished from one another. A standard program may only interpret the results of **`EKEY`** via the translation words provided for that purpose (**`EKEY>CHAR`** and **`EKEY>FKEY`**). ed07

In systems that combine both keyboard and mouse events into a single "event stream", the single number returned by **`EKEY`** may be inadequate to represent the full range of input possibilities. In such systems, a single "event record" may include a time stamp, the x,y coordinates of the mouse position, the keyboard state, and the state of the mouse buttons. In such systems, it might be appropriate for **`EKEY`** to return the address of an "event record" from which the other information could be extracted.

While the standard specifies a timing relationship between **`EKEY?`**, **`KEY?`**, **`EKEY`** and **`KEY`**, some programmers find that confusing. One way to avoid such confusion is to use only one pairing in a program (for each input stream). Use **`KEY?`** and **`KEY`** where the application does not require access to anything other than the standard character set. Use **`EKEY?`** and **`EKEY`** when extended functionality is required.

Also, consider a hypothetical Forth system running under MS-DOS on a PC-compatible computer. Assume that the implementation-defined character set is the "normal" 8-bit PC character set. In that character set, the codes from 0 to 127 correspond to ASCII characters. The codes from 128 to 255 represent characters from various non-English languages, mathematical symbols, and some graphical symbols used for line drawing. In addition to those characters, the keyboard can generate various other "scan codes", representing such non-character events as arrow keys and function keys.

There may be multiple keys, with different scan codes, corresponding to the same standard character. For example, the character representing the number "1" often appears both in the row of number keys above the alphabetic keys, and also in the separate numeric keypad.

10.6.2.1307 EKEY? "e-key-question" FACILITY EXT

(– *flag*)

If a keyboard event is available, return *true*. Otherwise return *false*. The event shall be returned by the next execution of **EKEY**.

After **EKEY?** returns with a value of *true*, subsequent executions of **EKEY?** prior to the execution of **KEY**, **KEY?** or **EKEY** also return *true*, referring to the same event.

10.6.2.1325 EMIT? "emit-question" FACILITY EXT

(– *flag*)

flag is true if the user output device is ready to accept data and the execution of **EMIT** in place of **EMIT?** would not have suffered an indefinite delay. If the device status is indeterminate, *flag* is true.

Rationale: An indefinite delay is a device related condition, such as printer off-line, that requires operator intervention before the device will accept new data.

10.6.2.—— END-STRUCTURE FACILITY EXT X:structures

(*struct-sys +n* –)

Terminate definition of a structure started by **BEGIN-STRUCTURE**.

See: **10.6.2.0** +FIELD and **10.6.2.0** BEGIN-STRUCTURE.

Implementation: Terminate definition of a structure.

```
:  END-STRUCTURE  \ addr n -
  SWAP ! ;        \ set len
```

10.6.2.—— FIELD: "field-colon" FACILITY EXT X:structures

(–)

The semantics of **FIELD:** are identical to the execution semantics of the phrase:

```
ALIGNED 1 CELLS +FIELD
```

See: **10.6.2.0** +FIELD, **10.6.2.0** BEGIN-STRUCTURE and **10.6.2.0** END-STRUCTURE.

Implementation: Create a new field within a structure definition of size a cell. The field is aligned.

```
:  field:  \ n1 <"name"> - n2 ; Exec:  addr - 'addr
  ALIGNED 1 CELLS +FIELD
;
```

10.6.2.—— K-ALT-MASK FACILITY EXT X:ekeys

(– *u*)

Mask for the ALT key, that can be **OR**ed with the key value to produce a value that the sequence **EKEY EKEY>FKEY** may produce when the user presses the corresponding key combination.

See: **10.6.2.0** EKEY>FKEY, **A.10.6.2.0** EKEY>FKEY.

12 The optional Floating-Point word set

12.1 Introduction

12.2 Additional terms and notation

12.2.1 Definition of terms

float-aligned address: The address of a memory location at which a floating-point number can be accessed.

double-float-aligned address: The address of a memory location at which a 64-bit IEEE double-precision floating-point number can be accessed.

single-float-aligned address: The address of a memory location at which a 32-bit IEEE single-precision floating-point number can be accessed.

IEEE floating-point number: A single- or double-precision floating-point number as defined in **ANSI/IEEE 754-1985**.

x:number-prefix

12.2.2 Notation

12.2.2.1 ~~Numeric notation~~

~~The following notation is used to define the syntax of the external representation of floating-point numbers:~~

- ~~Each component of a floating-point number is defined with a rule consisting of the name of the component (italicized in angle-brackets, e.g., ⟨*sign*⟩), the characters := and a concatenation of tokens and metacharacters;~~
- ~~Tokens may be literal characters (in bold face, e.g., **E**) or rule names in angle brackets (e.g., ⟨*digit*⟩);~~
- ~~The metacharacter * is used to specify zero or more occurrences of the preceding token (e.g., ⟨*digit*⟩*);~~
- ~~Tokens enclosed with [and] are optional (e.g., [⟨*sign*⟩]);~~
- ~~Vertical bars separate choices from a list of tokens enclosed with braces (e.g., { + | - }).~~

12.2.2.2 Stack notation

X:fp-stack

Floating-point stack notation ~~when the floating-point stack is separate from the data stack~~ is:

(F: *before* – *after*)

A unified stack notation is provided for systems with the environmental restriction that the floating-point numbers are kept on the data stack.

12.3 Additional usage requirements

12.3.1 Data types

Append table 12.1 to table 3.1.

Table 12.1: Data Types

Symbol	*Data type*	*Size on stack*
df-addr	double-float-aligned address	1 cell

12.3.1.1 Addresses

The set of float-aligned addresses is an implementation-defined subset of the set of aligned addresses. Adding the size of a floating-point number to a float-aligned address shall produce a float-aligned address.

The set of double-float-aligned addresses is an implementation-defined subset of the set of aligned addresses. Adding the size of a 64-bit IEEE double-precision floating-point number to a double-float-aligned address shall produce a double-float-aligned address.

The set of single-float-aligned addresses is an implementation-defined subset of the set of aligned addresses. Adding the size of a 32-bit IEEE single-precision floating-point number to a single-float-aligned address shall produce a single-float-aligned address.

12.3.1.2 Floating-point numbers

The internal representation of a floating-point number, including the format and precision of the significand and the format and range of the exponent, is implementation defined.

Any rounding or truncation of floating-point numbers is implementation defined.

12.3.2 Floating-point operations

“Round to nearest” means round the result of a floating-point operation to the representable value nearest the result. If the two nearest representable values are equally near the result, the one having zero as its least significant bit shall be delivered.

“Round toward negative infinity” means round the result of a floating-point operation to the representable value nearest to and no greater than the result.

12.3.3 Floating-point stack

A last in, first out list that shall be used by all floating-point operators.

The width of the floating-point stack is implementation-defined. ~~By default the~~The floating-point stack shall be separate from the data and return stacks. ~~A program may determine whether floating-point numbers are kept on the data stack by passing the string “FLOATING-STACK” to **ENVIRONMENT?**.~~ X:fp-stack

The size of a floating-point stack shall be at least 6 items.

A program that depends on the floating-point stack being larger than six items has an environmental dependency.

12.3.4 Environmental queries

Append table 12.2 to table 3.5.

See: **3.2.6 Environmental queries**.

Table 12.2: Environmental Query Strings

String	Value data type	Constant?	Meaning
FLOATING	*flag*	no	floating-point word set present
FLOATING-EXT	*flag*	no	floating-point extensions word set present
FLOATING-STACK	*n*	yes	~~If *n* = zero, floating-point numbers are kept on the data stack; otherwise~~*n* is the maximum depth of the separate floating-point stack. On systems with the environmental restriction of keeping floating-point items on the data stack, $n = 0$.
MAX-FLOAT	*r*	yes	largest usable floating-point number

12.3.5 Address alignment

Since the address returned by a **CREATE**d word is not necessarily aligned for any particular class of floating-point data, a program shall align the address (to be float aligned, single-float aligned, or double-float aligned) before accessing floating-point data at the address.

See: **3.3.3.1 Address alignment**, **12.3.1.1 Addresses**.

12.3.6 Variables

A program may address memory in data space regions made available by **FVARIABLE**. These regions may be non-contiguous with regions subsequently allocated with **,** (comma) or **ALLOT**. See: **3.3.3.3 Variables**.

12.3.7 Text interpreter input number conversion

If the Floating-Point word set is present in the dictionary and the current base is **DECIMAL**, the input number-conversion algorithm shall be extended to recognize floating-point numbers in this form:

Convertible string := ⟨*significand*⟩⟨*exponent*⟩
⟨*significand*⟩ := [⟨*sign*⟩]⟨*digits*⟩[**.**⟨*digits0*⟩]
⟨*exponent*⟩ := **E**[⟨*sign*⟩]⟨*digits0*⟩
⟨*sign*⟩ := { **+** | **-** }
⟨*digits*⟩ := ⟨*digit*⟩⟨*digits0*⟩
⟨*digits0*⟩ := ⟨*digit*⟩*
⟨*digit*⟩ := { **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** }

These are examples of valid representations of floating-point numbers in program source:

```
1E      1.E      1.E0      +1.23E-1      -1.23E+1
```

See: **3.4.1.3 Text interpreter input number conversion**, **12.6.1.0558** >FLOAT.

12.4 Additional documentation requirements

12.4.1 System documentation

12.4.1.1 Implementation-defined options

– format and range of floating-point numbers (**12.3.1 Data types**, **12.6.1.2143** REPRESENT)

- results of **12.6.1.2143** REPRESENT when *float* is out of range;
- rounding or truncation of floating-point numbers (**12.3.1.2 Floating-point numbers**);
- size of floating-point stack (**12.3.3 Floating-point stack**);
- width of floating-point stack (**12.3.3 Floating-point stack**).

12.4.1.2 Ambiguous conditions

- **DF@** or **DF!** is used with an address that is not double-float aligned;
- **F@** or **F!** is used with an address that is not float aligned;
- floating point result out of range (e.g., in **12.6.1.1430** F/);
- **SF@** or **SF!** is used with an address that is not single-float aligned;
- **BASE** is not decimal (**12.6.1.2143** REPRESENT, **12.6.2.1427** F, **12.6.2.1513** FE, **12.6.2.1613** FS);
- both arguments equal zero (**12.6.2.1489** FATAN2);
- cosine of argument is zero for **12.6.2.1625** FTAN;
- *d* can't be precisely represented as *float* in **12.6.1.1130** D>F;
- dividing by zero (**12.6.1.1430** F/);
- exponent too big for conversion (**12.6.2.1203** DF!, **12.6.2.1204** DF@, **12.6.2.2202** SF!, **12.6.2.2203** SF@);
- *float* less than one (**12.6.2.1477** FACOSH);
- *float* less than or equal to minus-one (**12.6.2.1554** FLNP1);
- *float* less than or equal to zero (**12.6.2.1553** FLN, **12.6.2.1557** FLOG);
- *float* less than zero (**12.6.2.1487** FASINH, **12.6.2.1618** FSQRT);
- *float* magnitude greater than one (**12.6.2.1476** FACOS, **12.6.2.1486** FASIN, **12.6.2.1491** FATANH);
- integer part of *float* can't be represented by *d* in **12.6.1.1470** F>D;
- string larger than pictured-numeric output area (**12.6.2.1427** F, **12.6.2.1513** FE, **12.6.2.1613** FS).

12.4.1.3 Other system documentation

- no additional requirements.

12.4.1.4 Environmental restrictions

- Keeping floating-point numbers on the data stack. X:fp-stack

12.4.2 Program documentation

12.4.2.1 Environmental dependencies

- requiring the floating-point stack to be larger than six items (**12.3.3 Floating-point stack**).
- requiring floating-point numbers to be kept on the data stack, with *n* cells per floating point number. X:fp-stack

12.6.1.1497 FDEPTH "f-depth" FLOATING

(– +*n*)

+*n* is the number of values contained on the ~~default separate~~ floating-point stack. If ~~floating-point numbers are kept~~the system has an environmental restriction of keeping the floating-point numbers on the data stack, +*n* is the current number of possible floating-point values contained on the data stack. X:fp-stack

12.6.1.1500 FDROP "f-drop" FLOATING

(F: *r* –) or (*r* –)

Remove *r* from the floating-point stack.

12.6.1.1510 FDUP "f-dupe" FLOATING

(F: *r* – *r r*) or (*r* – *r r*)

Duplicate *r*.

12.6.1.1552 FLITERAL "f-literal" FLOATING

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (F: *r* –) or (*r* –)

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

Run-time: (F: – *r*) or (– *r*)

Place *r* on the floating-point stack.

Rationale: Typical use: `: X ... [ ... ( r ) ] FLITERAL ... ;`

12.6.1.1555 FLOAT+ "float-plus" FLOATING

(*f-addr*$_1$ – *f-addr*$_2$)

Add the size in address units of a floating-point number to *f-addr*$_1$, giving *f-addr*$_2$.

12.6.1.1556 FLOATS FLOATING

(n_1 – n_2)

n_2 is the size in address units of n_1 floating-point numbers.

12.6.1.1558 FLOOR FLOATING

(F: r_1 – r_2) or (r_1 – r_2)

Round r_1 to an integral value using the "round toward negative infinity" rule, giving r_2.

more precision than the internal representation it will be rounded to the internal representation using the "round to nearest" rule. An ambiguous condition exists if the exponent of the IEEE double-precision representation is too large to be accommodated by the internal representation.

See: **12.3.1.1 Addresses**, **12.3.2 Floating-point operations**.

12.6.2.1205 `DFALIGN` "d-f-align" FLOATING EXT

(–)

If the data-space pointer is not double-float aligned, reserve enough data space to make it so.

See: **12.3.1.1 Addresses**.

12.6.2.1207 `DFALIGNED` "d-f-aligned" FLOATING EXT

(*addr* – *df-addr*)

df-addr is the first double-float-aligned address greater than or equal to *addr*.

See: **12.3.1.1 Addresses**.

12.6.2.—— `DFFIELD:` "d-f-field-colon" FLOATING EXT
X:structures

(–)

The semantics of **`DFFIELD:`** are identical to the execution semantics of the phrase:

```
DFALIGNED 1 DFLOATS +FIELD
```

See: **10.6.2.0** `+FIELD`, **10.6.2.0** `BEGIN-STRUCTURE` and **10.6.2.0** `END-STRUCTURE`.

Implementation: Create a new field within a structure definition of size a double-float. The field is aligned.

```
:  DFFIELD:  \ n1 <"name"> – n2 ; Exec:  addr – 'addr
  DFALIGNED 1 DFLOATS +FIELD
;
```

12.6.2.1208 `DFLOAT+` "d-float-plus" FLOATING EXT

($df\text{-}addr_1$ – $df\text{-}addr_2$)

Add the size in address units of a 64-bit IEEE double-precision number to $df\text{-}addr_1$, giving $df\text{-}addr_2$.

See: **12.3.1.1 Addresses**.

12.6.2.1209 `DFLOATS` "d-floats" FLOATING EXT

(n_1 – n_2)

n_2 is the size in address units of n_1 64-bit IEEE double-precision numbers.

```
: FSINH ( r1 - r2 )
  FEXPM1 FDUP FDUP 1.0E0 F+ F/ F+ 2.0E0 F/ ;
```

12.6.2.—— FFIELD: "f-field-colon" FLOATING EXT X:structures

(-)

The semantics of **FFIELD:** are identical to the execution semantics of the phrase:

```
FALIGNED 1 FLOATS +FIELD
```

See: **10.6.2.0** +FIELD, **10.6.2.0** BEGIN-STRUCTURE and **10.6.2.0** END-STRUCTURE.

Implementation: Create a new field within a structure definition of size a float. The field is aligned.

```
: FFIELD: \ n1 <"name"> - n2 ; Exec: addr - 'addr
  FALIGNED 1 FLOATS +FIELD
;
```

12.6.2.1553 FLN "f-l-n" FLOATING EXT

(F: r_1 - r_2) or (r_1 - r_2)

r_2 is the natural logarithm of r_1. An ambiguous condition exists if r_1 is less than or equal to zero.

12.6.2.1554 FLNP1 "f-l-n-p-one" FLOATING EXT

(F: r_1 - r_2) or (r_1 - r_2)

r_2 is the natural logarithm of the quantity r_1 plus one. An ambiguous condition exists if r_1 is less than or equal to negative one.

Rationale: This function allows accurate compilation when its arguments are close to zero, and provides a useful base for the standard logarithmic functions. For example, **FLN** can be implemented as:

```
: FLN 1.0E0 F- FLNP1 ;
```

See: **A.12.6.2.1516** FEXPM1.

12.6.2.1557 FLOG "f-log" FLOATING EXT

(F: r_1 - r_2) or (r_1 - r_2)

r_2 is the base-ten logarithm of r_1. An ambiguous condition exists if r_1 is less than or equal to zero.

12.6.2.2204 SFALIGN "s-f-align" FLOATING EXT

(–)

If the data-space pointer is not single-float aligned, reserve enough data space to make it so.

See: **12.3.1.1 Addresses**.

12.6.2.2206 SFALIGNED "s-f-aligned" FLOATING EXT

(*addr* – *sf-addr*)

sf-addr is the first single-float-aligned address greater than or equal to *addr*.

See: **12.3.1.1 Addresses**.

12.6.2.—— SFFIELD: "s-f-field-colon" FLOATING EXT
X:structures

(–)

The semantics of **SFFIELD:** are identical to the execution semantics of the phrase:

```
SFALIGNED 1 SFLOATS +FIELD
```

See: **10.6.2.0** +FIELD, **10.6.2.0** BEGIN-STRUCTURE and **10.6.2.0** END-STRUCTURE.

Implementation: Create a new field within a structure definition of size a single-float. The field is aligned.

```
:  SFFIELD:  \ n1 <"name"> – n2 ; Exec:  addr – 'addr
  SFALIGNED 1 SFLOATS +FIELD
;
```

12.6.2.2207 SFLOAT+ "s-float-plus" FLOATING EXT

($sf\text{-}addr_1$ – $sf\text{-}addr_2$)

Add the size in address units of a 32-bit IEEE single-precision number to $sf\text{-}addr_1$, giving $sf\text{-}addr_2$.

See: **12.3.1.1 Addresses**.

12.6.2.2208 SFLOATS "s-floats" FLOATING EXT

(n_1 – n_2)

n_2 is the size in address units of n_1 32-bit IEEE single-precision numbers.

See: **12.3.1.1 Addresses**.

```
TABLE C@                          will return 1
TABLE CHAR+ C@                    will return 2
TABLE 2 CHARS + ALIGNED @         will return 1000
TABLE 2 CHARS + ALIGNED CELL+ @   will return 2000.
```

Similarly,

```
CREATE DATA   1000 ALLOT
```

makes an array 1000 address units in size. A more portable strategy would define the array in application units, such as:

```
500 CONSTANT NCELLS
CREATE CELL-DATA NCELLS CELLS ALLOT
```

This array can be indexed like this:

```
:  LOOK   NCELLS 0 DO CELL-DATA I CELLS + ?  LOOP ;
```

A.3.3.3.6 Other transient regions

In many existing Forth systems, these areas are at **HERE** or just beyond it, hence the many restrictions.

$(2*n)+2$ is the size of a character string containing the unpunctuated binary representation of the maximum double number with a leading minus sign and a trailing space.

Implementation note: Since the minimum value of n is 16, the absolute minimum size of the pictured numeric output string is 34 characters. But if your implementation has a larger n, you must also increase the size of the pictured numeric output string.

A.3.4 The Forth text interpreter

x:number-prefix

A.3.4.1.3 Numeric notation

The numeric representation provided by the X:number-prefix extension can be tested with the following test cases:

```
DECIMAL
{ #1289 -> 1289 }
{ #12346789. -> 12346789. }
{ #-1289 -> -1289 }
{ #-12346789. -> -12346789. }
{ $12eF -> 4847 }
{ $12aBcDeF. -> 313249263. }
{ $-12eF -> -4847 }
{ $-12AbCdEf. -> -313249263. }
{ %10010110 -> 150 }
{ %10010110. -> 150. }
{ %-10010110 -> -150 }
{ %-10010110. -> -150. }
{ 'z' -> 122 }
```

Annex B
(informative)
Bibliography

Industry standards

Forth-77 Standard, Forth Users Group, FST-780314.
Forth-78 Standard, Forth International Standards Team.
Forth-79 Standard, Forth Standards Team.
Forth-83 Standard and Appendices, Forth Standards Team.

The standards referenced in this section were developed by the Forth Standards Team, a volunteer group which included both implementors and users. This was a volunteer organization operating under its own charter and without any formal ties to ANSI, IEEE or any similar standards body.

The following standards where developed under the auspices of ANSI. The committee drawing up the ANSI standard included several members of the Forth Standards Team.

ANSI X3.215-1994 Information Systems — Programming Language FORTH
ISO/IEC 15145:1997 Information technology. Programming languages. FORTH

Books

Brodie, L. *Thinking FORTH*. Englewood Cliffs, NJ: Prentice Hall, 1984. Now available from http://thinking-forth.sourceforge.net/ ed07

Brodie, L. *Starting FORTH* (2nd edition). Englewood Cliffs, NJ: Prentice Hall, 1987.

Feierbach, G. and Thomas, P. *Forth Tools & Applications*. Reston, VA: Reston Computer Books, 1985.

Haydon, Dr. Glen B. *All About FORTH* (3rd edition). La Honda, CA: 1990.

Kelly, Mahlon G. and Spies, N. *FORTH: A Text and Reference*. Englewood Cliffs, NJ: Prentice Hall, 1986.

Knecht, K. *Introduction to Forth*. Indiana: Howard Sams & Co., 1982.

Koopman, P. *Stack Computers, The New Wave*. Chichester, West Sussex, England: Ellis Horwood Ltd. 1989.

Martin, Thea, editor. *A Bibliography of Forth References* (3rd edition). Rochester, New York: Institute of Applied Forth Research, 1987.

McCabe, C. K. *Forth Fundamentals* (2 volumes). Oregon: Dilithium Press, 1983.

Ouverson, Marlin, editor. *Dr. Dobbs Toolbook of Forth*. Redwood City, CA: M&T Press, Vol. 1, 1986; Vol. 2, 1987.

Pelc, Stephen. *Programming Forth*. Southampton, England: MicroProcessor Engineering Limited, 2005. http://www.mpeforth.com/arena/ProgramForth.pdf.

Pountain, R. *Object Oriented Forth*. London, England: Academic Press, 1987.

Rather, Elizabeth D. *Forth Application Techniques*. FORTH, Inc., 2006. ISBN: 978-0966215618.

Rather, Elizabeth D. and Conklin, Edward K. *Forth Programmer's Handbook* (3rd edition). BookSurge Publishing, 2007. ISBN: 978-1419675492.

Terry, J. D. *Library of Forth Routines and Utilities*. New York: Shadow Lawn Press, 1986.

Tracy, M. and Anderson, A. *Mastering FORTH* (revised edition). New York: Brady Books, 1989.

Winfield, A. *The Complete Forth*. New York: Wiley Books, 1983.