D. Compatibility analysis of ANS Forth (informative annex)

Prior to ANS Forth, there were several industry standards for Forth. The most influential are listed here in chronological order, along with the major differences between ANS Forth and the most recent, Forth-83.

D.1 FIG Forth (circa 1978)

FIG Forth was a **model** implementation of the Forth language developed by the Forth Interest Group (FIG). In FIG Forth, a relatively small number of words were implemented in processor-dependent machine language and the rest of the words were implemented in Forth. The FIG model was placed in the public domain, and was ported to a wide variety of computer systems. Because the bulk of the FIG Forth implementation was the same across all machines, programs written in FIG Forth enjoyed a substantial degree of portability, even for **system-level** programs that directly manipulate the internals of the Forth system implementation.

FIG Forth implementations were influential in increasing the number of people interested in using Forth. Many people associate the implementation techniques embodied in the FIG Forth model with **the nature of Forth**.

However, FIG Forth was not necessarily representative of commercial Forth implementations of the same era. Some of the most successful commercial Forth systems used implementation techniques different from the FIG Forth **model**.

D.2 Forth-79

The Forth-79 Standard resulted from a series of meetings from 1978 to 1980, by the Forth Standards Team, an international group of Forth users and vendors (interim versions known as Forth 77 and Forth 78 were also released by the group).

Forth-79 described a set of words defined on a 16-bit, twos-complement, unaligned, linear byte-addressing virtual machine. It prescribed an implementation technique known as **indirect threaded code**, and used the ASCII character set.

The Forth-79 Standard served as the basis for several public domain and commercial implementations, some of which are still available and supported today.

D.3 Forth-83

The Forth-83 Standard, also by the Forth Standards Team, was released in 1983. Forth-83 attempted to fix some of the deficiencies of Forth-79.

Forth-83 was similar to Forth-79 in most respects. However, Forth-83 changed the definition of several well-defined features of Forth-79. For example, the rounding behavior of integer division, the base value of the operands of PICK and ROLL, the meaning of the address returned by ', the compilation behavior of ', the value of a **true** flag, the meaning of NOT, and the **chaining** behavior of words defined by VOCABULARY were all changed. Forth-83 relaxed the implementation restrictions of Forth-79 to allow any kind of threaded code, but it did not fully allow compilation to native machine code (this was not specifically prohibited, but rather was an indirect consequence of another provision).

Many new Forth implementations were based on the Forth-83 Standard, but few **strictly compliant** Forth-83 implementations exist.

Although the incompatibilities resulting from the changes between Forth-79 and Forth-83 were usually relatively easy to fix, a number of successful Forth vendors did not convert their implementations to be Forth-83 compliant. For example, the most successful commercial Forth for Apple Macintosh computers is based on Forth-79.

D.4 Recent developments

Since the Forth-83 Standard was published, the computer industry has undergone rapid and profound changes. The speed, memory capacity, and disk capacity of affordable personal computers have increased by factors of more than 100. 8-bit processors have given way to 16-bit processors, and now 32-bit processors are commonplace.

The operating systems and programming-language environments of small systems are much more powerful than they were in the early 80's.

The personal-computer marketplace has changed from a predominantly **hobbyist** market to a mature business and commercial market.

Improved technology for designing custom microprocessors has resulted in the design of numerous **Forth chips**, computers optimized for the execution of the Forth language.

The market for ROM-based embedded control computers has grown substantially.

In order to take full advantage of this evolving technology, and to better compete with other programming languages, many recent Forth implementations have ignored some of the **rules** of previous Forth standards. In particular:

- 32-bit Forth implementations are now common.
- Some Forth systems adopt the address-alignment restrictions of the hardware on which they run.
- Some Forth systems use native-code generation, microcode generation, and optimization techniques, rather than the traditional **threaded code**.
- Some Forth systems exploit segmented addressing architectures, placing portions of the Forth **dictionary** in different segments.
- More and more Forth systems now run in the environment of another **standard** operating system, using OS text files for source code, rather than the traditional Forth **blocks**.
- Some Forth systems allow external operating system software, windowing software, terminal concentrators, or communications channels to handle or preprocess user input, resulting in deviations from the input editing, character set availability, and screen management behavior prescribed by Forth-83.

Competitive pressure from other programming languages (predominantly **C**) and from other Forth vendors have led Forth vendors to optimizations that do not fit in well with the **virtual machine model** implied by existing Forth standards.

D.5 ANS Forth approach

The ANS Forth committee addressed the serious fragmentation of the Forth community caused by the differences between Forth-79 and Forth-83, and the divergence from either of these two industry standards caused by marketplace pressures.

Consequently, the committee has chosen to base its compatibility decisions not upon a strict comparison with the Forth-83 Standard, but instead upon consideration of the variety of existing implementations, especially those with substantial user bases and/or considerable success in the marketplace.

The committee feels that, if ANS Forth prescribes stringent requirements upon the virtual machine model, as did the previous standards, then many implementors will chose not to comply with ANS Forth. The committee hopes that ANS Forth will serve to unify rather than to further divide the Forth community, and thus has chosen to encompass rather than invalidate popular implementation techniques.

Many of the changes from Forth-83 are justified by this rationale. Most fall into the category that **an ANS Forth Standard Program may not assume x**, where **x** is an entitlement resulting from the virtual machine model prescribed by the Forth-83 Standard. The committee feels that these restrictions are reasonable, especially considering that a substantial number of existing Forth implementations do not correctly implement the Forth-83 virtual model, thus the Forth-83 entitlements exist **in theory** but not **in practice**.

Another way of looking at this is that while ANS Forth acknowledges the diversity of current Forth practice, it attempts to document the similarity therein. In some sense, ANS Forth is thus a **description of reality** rather than a **prescription for a particular virtual machine**.

Since there is no previous American National Standard for Forth, the action requirements prescribed by section 3.4 of X3/SD-9, **Policy and Guidelines**, regarding previous standards do not apply.

The following discussion describes differences between ANS Forth and Forth-83. In most cases, Forth-83 is representative of Forth-79 and FIG Forth for the purposes of this discussion. In many of these cases, however, ANS Forth is more representative of the existing state of the Forth industry than the previously-published standards.

D.6 Differences from Forth-83

D.6.1 Stack width

Forth-83 specifies that stack items occupy 16 bits. This includes addresses, flags, and numbers. ANS Forth specifies that stack items are at least 16 bits; the actual size must be documented by the implementation.

Words affected: all arithmetic, logical and addressing operators

Reason: 32-bit machines are becoming commonplace. A 16-bit Forth system on a 32-bit machine is not competitive.

Impact: Programs that assume 16-bit stack width will continue to run on 16-bit machines; ANS Forth does not require a different stack width, but simply allows it. Many programs will be unaffected (but see **address unit**).

Transition/Conversion: Programs which use bit masks with the high bits set may have to be changed, substituting either an implementation-defined bit-mask constant, or a procedure to calculate a bit mask in a stack-width-independent way. Here are some procedures for constructing width-independent bit masks:

```
1  CONSTANT LO-BIT
TRUE 1 RSHIFT  INVERT  CONSTANT HI-BIT

: LO-BITS  ( n -- mask )
    0 SWAP  0 ?DO  1 LSHIFT  LO-BIT OR  LOOP ;

: HI-BITS  ( n -- mask )
    0 SWAP  0 ?DO  1 RSHIFT  HI-BIT OR  LOOP ;
```

Programs that depend upon the **modulo 65536** behavior implicit in 16-bit arithmetic operations will need to be rewritten to explicitly perform the modulus operation in the appropriate places. The committee believes that such assumptions occur infrequently. Examples: some checksum or CRC calculations, some random number generators and most fixed-point fractional math.

D.6.2 Number representation

Forth-83 specifies two's-complement number representation and arithmetic. ANS Forth also allows one's-complement and signed-magnitude.

Words affected: all arithmetic and logical operators, LOOP, +LOOP

Reason: Some computers use one's-complement or signed-magnitude. The committee did not wish to force Forth implementations for those machines to emulate two's-complement arithmetic, and thus incur severe performance penalties. The experience of some committee members with such machines indicates that the usage restrictions necessary to support their number representations are not overly burdensome.

Impact: An ANS Forth Standard Program may declare an **environmental dependency on two's-complement arithmetic**. This means that the otherwise-Standard Program is only guaranteed to work on two's-complement machines. Effectively, this is not a severe restriction, because the overwhelming majority of current computers use two's-complement. The committee knows of no Forth-83 compliant implementations for non-two's-complement machines at present, so existing Forth-83 programs will still work on the same class of machines on which they currently work.

Transition/Conversion: Existing programs wishing to take advantage of the possibility of ANS Forth Standard Systems on non-two's-complement machines may do so by eliminating the use of arithmetic operators to perform logical functions, by deriving bit-mask constants from bit operations as described in the section about stack width, by restricting the usage range of unsigned numbers to the range of positive numbers, and by using the provided operators for conversion from single numbers to double numbers.

D.6.3 Address units

Forth-83 specifies that each unique address refers to an 8-bit byte in memory. ANS Forth specifies that the size of the item referred to by each unique address is implementation-defined, but, by default, is the size of one character. Forth-83 describes many memory operations in terms of a number of bytes. ANS Forth describes those operations in terms of a number of either characters or address units.

Words affected: those with **address unit** arguments

Reason: Some machines, including the most popular Forth chip, address 16-bit memory locations instead of 8-bit bytes.

Impact: Programs may choose to declare an environmental dependency on byte addressing, and will continue to work on the class of machines for which they now work. In order for a Forth implementation on a word-addressed machine to be Forth-83 compliant, it would have to simulate byte addressing at considerable cost in speed and memory efficiency. The committee knows of no such Forth-83 implementations for such machines, thus an environmental dependency on byte addressing does not restrict a Standard Program beyond its current de facto restrictions.

Transition/Conversion: The new CHARS and CHAR+ address arithmetic operators should be used for programs that require portability to non-byte-addressed machines. The places where such conversion is necessary may be identified by searching for occurrences of words that accept a number of address units as an argument (e.g., MOVE, ALLOT).

D.6.4 Address increment for a cell is no longer two

As a consequence of Forth-83's simultaneous specification of 16-bit stack width and byte addressing, the number two could reliably be used in address calculations involving memory arrays containing items from the stack. Since ANS Forth requires neither 16-bit stack width nor byte addressing, the number two is no longer necessarily appropriate for such calculations.

Words affected: @ ! +! 2+ 2* 2- +LOOP

Reason: See reasons for **Address Units** and **Stack Width**

Impact: In this respect, existing programs will continue to work on machines where a stack cell occupies two address units when stored in memory. This includes most machines for which Forth-83 compliant implementations currently exist. In principle, it would also include 16-bit-word-addressed machines with 32-bit stack width, but the committee knows of no examples of such machines.

Transition/Conversion: The new CELLS and CELL+ address arithmetic operators should be used for portable programs. The places where such conversion is necessary may be identified by searching for the character **2** and determining whether or not it is used as part of an address calculation. The following substitutions are appropriate within address calculations:

```
Old                     New
---                     ---
2+  or  2 +             CELL+
2*  or  2 *             CELLS
2-  or  2 -             1 CELLS -
2/  or  2 /             1 CELLS /
2                       1 CELLS
```

The number **2** by itself is sometimes used for address calculations as an argument to +LOOP, when the loop index is an address. When converting the word 2/ which operates on negative dividends, one should be cognizant of the rounding method used.

D.6.5 Address alignment

Forth-83 imposes no restriction upon the alignment of addresses to any boundary. ANS Forth specifies that a Standard System may require alignment of addresses for use with various **@** and **!** operators.

Words Affected: ! +! 2! 2@ @ ? ,

Reason: Many computers have hardware restrictions that favor the use of aligned addresses. On some machines, the native memory-access instructions will cause an exception trap if used with an unaligned address. Even on machines where unaligned accesses do not cause exception traps, aligned accesses are usually faster.

Impact: All of the ANS Forth words that return addresses suitable for use with aligned **@** and **!** words must return aligned addresses. In most cases, there will be no problem. Problems can arise from the use of user-defined data structures containing a mixture of character data and cell-sized data.

Many existing Forth systems, especially those currently in use on computers with strong alignment requirements, already require alignment. Much existing Forth code that is currently in use on such machines has already been converted for use in an aligned environment.

Transition/Conversion: There are two possible approaches to conversion of programs for use on a system requiring address alignment.

The easiest approach is to redefine the system's aligned **@** and **!** operators so that they do not require alignment. For example, on a 16-bit little-endian byte-addressed machine, unaligned **@** and **!** could be defined:

```
        : @  ( addr -- x )  DUP C@ SWAP CHAR+ C@ 8 LSHIFT OR  ;
        : !  ( x addr -- )  OVER 8 RSHIFT OVER CHAR+ C! C!  ;
```

These definitions, and similar ones for **+!**, **2@**, **2!**, **,**, and **?** as needed, can be compiled before an unaligned application, which will then work as expected.

This approach may conserve memory if the application uses substantial numbers of data structures containing unaligned fields.

Another approach is to modify the application's source code to eliminate unaligned data fields. The ANS Forth words ALIGN and ALIGNED may be used to force alignment of data fields. The places where such alignment is needed may be determined by inspecting the parts of the application where data structures (other than simple variables) are defined, or by **smart compiler** techniques (see the **Smart Compiler** discussion below).

This approach will probably result in faster application execution speed, at the possible expense of increased memory utilization for data structures.

Finally, it is possible to combine the preceding techniques by identifying exactly those data fields that are unaligned, and using **unaligned** versions of the memory access operators for only those fields. This **hybrid** approach affects a compromise between execution speed and memory utilization.

D.6.6 Division/modulus rounding direction

Forth-79 specifies that division rounds toward 0 and the remainder carries the sign of the dividend. Forth-83 specifies that division rounds toward negative infinity and the remainder carries the sign of the divisor. ANS Forth allows either behavior for the division operators listed below, at the discretion of the implementor, and provides a pair of division primitives to allow the user to synthesize either explicit behavior.

Words Affected: / MOD /MOD */MOD */

Reason: The difference between the division behaviors in Forth-79 and Forth-83 was a point of much contention, and many Forth implementations did not switch to the Forth-83 behavior. Both variants have vocal proponents, citing both application requirements and execution efficiency arguments on both sides. After extensive debate spanning many meetings, the committee was unable to reach a consensus for choosing one behavior over the other, and chose to allow either behavior as the default, while providing a means for the user to explicitly use both behaviors as needed. Since implementors are allowed to choose either behavior, they are not required to change the behavior exhibited by their current systems, thus preserving correct functioning of existing programs that run on those systems and depend on a particular behavior. New implementations could choose to supply the behavior that is supported by the native CPU instruction set, thus maximizing execution speed, or could choose the behavior that is most appropriate for the intended application domain of the system.

Impact: The issue only affects programs that use a negative dividend with a positive divisor, or a positive dividend with a negative divisor. The vast majority of uses of division occur with both a positive dividend and a positive divisor; in that case, the results are the same for both allowed division behaviors.

Transition/Conversion: For programs that require a specific rounding behavior with division operands of mixed sign, the division operators used by the program may be redefined in terms of one of the new ANS Forth division primitives SM/REM (symmetrical division, i.e., round toward zero) or FM/MOD (floored division, i.e., round toward negative infinity). Then the program may be recompiled without change. For example, the Forth-83 style division operators may be defined by:

```
: /MOD  ( n1 n2 -- n3 n4 )     >R S>D R> FM/MOD  ;
: MOD   ( n1 n2 -- n3 )        /MOD DROP  ;
: /     ( n1 n2 -- n3 )        /MOD SWAP DROP   ;
: */MOD ( n1 n2 n3 -- n4 n5 )  >R M* R> FM/MOD  ;
: */    ( n1 n2 n3 -- n4 n5 )  */MOD SWAP DROP  ;
```

D.6.7 Immediacy

Forth-83 specified that a number of **compiling words** are **immediate**, meaning that they are executed instead of compiled during compilation. ANS Forth is less specific about most of these words, stating that their behavior is only defined during compilation, and specifying their results rather than their specific compile-time actions.

To force the compilation of a word that would normally be executed, Forth-83 provided the words COMPILE , used with non-immediate words, and [COMPILE] , used with immediate words. ANS Forth provides the single word POSTPONE , which is used with both immediate and non-immediate words, automatically selecting the appropriate behavior.

Words Affected: **COMPILE [COMPILE] ['] '**

Reason: The designation of particular words as either immediate or not depends upon the implementation technique chosen for the Forth system. With traditional **threaded code** implementations, the choice was generally quite clear (with the single exception of the word LEAVE), and the standard could specify which words should be immediate. However, some of the currently popular implementation techniques, such as native-code generation with optimization, require the immediacy attribute on a different set of words than the set of immediate words of a threaded code implementation. ANS Forth, acknowledging the validity of these other implementation techniques, specifies the immediacy attribute in as few cases as possible.

When the membership of the set of immediate words is unclear, the decision about whether to use COMPILE or [COMPILE] becomes unclear. Consequently, ANS Forth provides a **general purpose** replacement word POSTPONE that serves the purpose of the vast majority of uses of both COMPILE and [COMPILE], without requiring that the user know whether or not the **postponed** word is immediate.

Similarly, the use of ' and ['] with compiling words is unclear if the precise compilation behavior of those words is not specified, so ANS Forth does not permit a Standard Program to use ' or ['] with compiling words.

The traditional (non-immediate) definition of the word COMPILE has an additional problem. Its traditional definition assumes a threaded code implementation technique, and its behavior can only be properly described in that context. In the context of ANS Forth, which permits other implementation techniques in addition to threaded code, it is very difficult, if not impossible, to describe the behavior of the traditional COMPILE. Rather than changing its behavior, and thus breaking existing code, ANS Forth does not include the word COMPILE. This allows existing implementations to continue to supply the word COMPILE with its traditional behavior, if that is appropriate for the implementation.

Impact: [COMPILE] remains in ANS Forth, since its proper use does not depend on knowledge of whether or not a word is immediate (Use of [COMPILE] with a non-immediate word is and has always been a no-op). Whether or not you need to use [COMPILE] requires knowledge of whether or not its target word is immediate, but it is always safe to use [COMPILE]. [COMPILE] is no longer in the (required) core word set, having been moved to the Core Extensions word set, but the committee anticipates that most vendors will supply it anyway.

In nearly all cases, it is correct to replace both [COMPILE] and COMPILE with POSTPONE. Uses of [COMPILE] and COMPILE that are not suitable for **mindless** replacement by POSTPONE are quite infrequent, and fall into the following two categories:

a) Use of [COMPILE] with non-immediate words. This is sometimes done with the words '-(tick, which was immediate in Forth-79 but not in Forth-83) and LEAVE (which was immediate in Forth-83 but not in Forth-79), in order to force the compilation of those words without regard to whether you are using a Forth-79 or Forth-83 system.

b) Use of the phrase COMPILE [COMPILE] <immediate word> to **doubly postpone** an immediate word.

Transition/Conversion: Many ANS Forth implementations will continue to implement both [COMPILE] and COMPILE in forms compatible with existing usage. In those environments, no conversion is necessary.

For complete portability, uses of COMPILE and [COMPILE] should be changed to POSTPONE , except in the rare cases indicated above. Uses of [COMPILE] with non-immediate words may be left as-is, and the program may declare a requirement for the word [COMPILE] from the Core Extensions word set, or the [COMPILE] before the non-immediate word may be simply deleted if the target word is known to be non-immediate.

Uses of the phrase COMPILE [COMPILE] <immediate-word> may be handled by introducing an **intermediate word** (XX in the example below) and then postponing that word. For example:

```
        : ABC  COMPILE [COMPILE] IF  ;
```

changes to:

```
        : XX  POSTPONE IF  ;
        : ABC  POSTPONE XX  ;
```

A non-standard case can occur with programs that **switch out of compilation state** to explicitly compile a thread in the dictionary following a COMPILE . For example:

```
        : XYZ  COMPILE  [ ' ABC , ]  ;
```

This depends heavily on knowledge of exactly how COMPILE and the threaded-code implementation works. Cases like this cannot be handled mechanically; they must be translated by understanding exactly what the code is doing, and rewriting that section according to ANS Forth restrictions.

Use the phrase POSTPONE [COMPILE] to replace [COMPILE] [COMPILE].

D.6.8 Input character set

Forth-83 specifies that the full 7-bit ASCII character set is available through KEY. ANS Forth restricts it to the graphic characters of the ASCII set, with codes from hex 20 to hex 7E inclusive.

Words Affected: KEY

Reason: Many system environments **consume** certain control characters for such purposes as input editing, job control, or flow control. A Forth implementation cannot always control this system behavior.

Impact: Standard Programs which require the ability to receive particular control characters through KEY must declare an environmental dependency on the input character set.

Transition/Conversion: For maximum portability, programs should restrict their required input character set to only the graphic characters. Control characters may be handled if available, but complete program functionality should be accessible using only graphic characters.

As stated above, an environmental dependency on the input character set may be declared. Even so, it is recommended that the program should avoid the requirement for particularly-troublesome control characters, such as control-S and control-Q (often used for flow control, sometimes by communication hardware whose presence may be difficult to detect), ASCII NUL (difficult to type on many keyboards), and the distinction between carriage return and line feed (some systems translate carriage returns into line feeds, or vice versa).

D.6.9 Shifting with UM/MOD

Given Forth-83's two's-complement nature, and its requirement for floored (round toward minus infinity) division, shifting is equivalent to division. Also, two's-complement representation implies that unsigned division by a power of two is equivalent to logical right-shifting, so UM/MOD could be used to perform a logical right-shift.

Words Affected: UM/MOD

Reason: The problem with UM/MOD is a result of allowing non-two's-complement number representations, as already described.

ANS Forth provides the words LSHIFT and RSHIFT to perform logical shifts. This is usually more efficient, and certainly more descriptive, than the use of UM/MOD for logical shifting.

Impact: Programs running on ANS Forth systems with two's-complement arithmetic (the majority of machines), will not experience any incompatibility with UM/MOD . Existing Forth-83 Standard programs intended to run on non-two's-complement machines will not be able to use UM/MOD for shifting on a non-two's-complement ANS Forth system. This should not affect a significant number of existing programs (perhaps none at all), since the committee knows of no existing Forth-83 implementations on non-two's-complement machines.

Transition/Conversion: A program that requires UM/MOD to behave as a shift operation may declare an environmental dependency on two's-complement arithmetic.

A program that cannot declare an environmental dependency on two's-complement arithmetic may require editing to replace incompatible uses of UM/MOD with other operators defined within the application.

D.6.10 Vocabularies / wordlists

ANS Forth does not define the words VOCABULARY, CONTEXT, and CURRENT , which were present in Forth-83. Instead, ANS Forth defines a primitive word set for search order specification and control, including words which have not existed in any previous standard.

Forth-83's ALSO/ONLY experimental search order word set is specified for the most part as the extension portion of the ANS Forth Search Order word set.

Words Affected: **VOCABULARY CONTEXT CURRENT**

Reason: Vocabularies are an area of much divergence among existing systems. Considering major vendors' systems and previous standards, there are at least 5 different and mutually incompatible behaviors of words defined by VOCABULARY. Forth-83 took a step in the direction of **run-time search-order specification** by declining to specify a specific relationship between the hierarchy of compiled vocabularies and the run-time search order. Forth-83 also specified an experimental mechanism for run-time search-order specification, the ALSO/ONLY scheme. ALSO/ONLY was implemented in numerous systems, and has achieved some measure of popularity in the Forth community.

However, several vendors refuse to implement it, citing technical limitations. In an effort to address those limitations and thus hopefully make ALSO/ONLY more palatable to its critics, the committee specified a simple **primitive word set** that not only fixes some of the objections to ALSO/ONLY, but also provides sufficient power to implement ALSO/ONLY and all of the other search-order word sets that are currently popular.

The Forth-83 ALSO/ONLY word set is provided as an optional extension to the search-order word set. This allows implementors that are so inclined to provide this word set, with well-defined standard behavior, but does not compel implementors to do so. Some vendors have publicly stated that they will not implement ALSO/ONLY, no matter what, and one major vendor stated an unwillingness to implement ANS Forth at all if ALSO/ONLY is mandated. The committee feels that its actions are prudent, specifying ALSO/ONLY to the extent possible without mandating its inclusion in all systems, and also providing a primitive search-order word set that vendors may be more likely to implement, and which can be used to synthesize ALSO/ONLY.

Transition/Conversion: Since Forth-83 did not mandate precise semantics for VOCABULARY, existing Forth-83 Standard programs cannot use it except in a trivial way. Programs can declare a dependency on the existence of the Search Order word set, and can implement whatever semantics are required using that word set's primitives. Forth-83 programs that need ALSO/ONLY can declare a dependency on the Search Order Extensions word set, or can implement the extensions in terms of the Search Order word set itself.

D.6.11 Multiprogramming impact

Forth-83 marked words with **multiprogramming impact** by the letter **M** in the first lines of their descriptions. ANS Forth has removed the **M** designation from the word descriptions, moving the discussion of multiprogramming impact to this non-normative annex.

Words affected: none

Reason: The meaning of **multiprogramming impact** is precise only in the context of a specific model for multiprogramming. Although many Forth systems do provide multiprogramming capabilities using a particular round-robin, cooperative, block-buffer sharing model, that model is not universal. Even assuming the classical model, the **M** designations did not contain enough information to enable writing of applications that interacted in a multiprogrammed system.

Practically speaking, the **M** designations in Forth-83 served to document usage rules for block buffer addresses in multiprogrammed systems. These addresses often become meaningless after a task has relinquished the CPU for any reason, most often for the purposes of performing I/O, awaiting an event, or voluntarily sharing CPU resources using the word PAUSE. It was essential that portable applications respect those usage rules to make it practical to run them on multiprogrammed systems; failure to adhere to the rules could easily compromise the integrity of other applications running on those systems as well as the applications actually in error. Thus, **M** appeared on all words that by design gave up the CPU, with the understanding that other words NEVER gave it up.

These usage rules have been explicitly documented in the Block word set where they are relevant. The **M** designations have been removed entirely.

Impact: In practice, none.

In the sense that any application that depends on multiprogramming must consist of at least two tasks that share some resource(s) and communicate between themselves, Forth-83 did not contain enough information to enable writing of a standard program that DEPENDED on multiprogramming. This is also true of ANS Forth.

Non-multiprogrammed applications in Forth-83 were required to respect usage rules for BLOCK so that they could be run properly on multiprogrammed systems. The same is true of ANS Forth.

The only difference is the documentation method used to define the BLOCK usage rules. The Technical Committee believes that the current method is clearer than the concept of **multiprogramming impact**.

Transition/Conversion: none needed.

D.6.12 Words not provided in executable form

ANS Forth allows an implementation to supply some words in source code or **load as needed** form, rather than requiring all supplied words to be available with no additional programmer action.

Words affected: all

Reason: Forth systems are often used in environments where memory space is at a premium. Every word included in the system in executable form consumes memory space. The committee believes that allowing standard words to be provided in source form will increase the probability that implementors will provide complete ANS Forth implementations even in systems designed for use in constrained environments.

Impact: In order to use a Standard Program with a given ANS Forth implementation, it may be necessary to precede the program with an implementation-dependent **preface** to make **source form** words executable. This is similar to the methods that other computer languages require for selecting the library routines needed by a particular application.

In languages like C, the goal of eliminating unnecessary routines from the memory image of an application is usually accomplished by providing libraries of routines, using a **linker** program to incorporate only the necessary routines into an executable application. The method of invoking and controlling the linker is outside the scope of the language definition.

Transition/Conversion: Before compiling a program, the programmer may need to perform some action to make the words required by that program available for execution.

Table of Contents

Next Section