Some Problems in Implementations of the FORTH Standards

Mahlon G. Kelly,
268 Turkey Ridge Rd.,
Charlottesville, VA 22901

Nicholas Spies,
313 Grace St.,
Pittsburgh, PA 15211-1503

There have been many complaints about the FORTH standards. These have been both from users and from those responsible for implementing the standards in commercial FORTHs. Complaints have even appeared in letters in major magazines, and have included such adjectives as "unuseable". But the reasons for the complaints have been, in large part, unfocused. Many users' complaints seem to stem from a wish that standard FORTH "did more", which reveals a misunderstanding of the purpose of the standards: to provide a minimal word set so as allow writing of transportable code (although one may argue what "minimal" means -- for example standard floating-point words were called for even before FORTH-79 was formulated). Perhaps a more basic user's complaint is that small changes in FORTH-83 (e.g. redefinition of ' and the termination of **LOOP**) make it difficult to translate code and can lead to subtle bugs (e.g. with floored division) if the differences are not understood. The failure of the FORTH-83 standard document to explicitly spell out differences aggravates this problem.

More fundamental complaints come from the implementors of FORTH systems (which, for lack of a better term, we will call "system-authors"). In some cases they have seen the differences between FORTH-79 and FORTH-83 as so small that they have not converted to the more recent standard (e.g. MMSFORTH). In other cases system-authors have provided both standards by using overlays (e.g. HS/FORTH). The most damning complaint, however, is that the standards documents are sufficiently unclear and/or ambiguous that implementation is subjective. For example, is it required that setting **BLK** to zero will force interpretation to the input device? Are state-smart words prohibited, e.g. can ." only work in compile mode and ' only work in execute mode? And what does "an error condition results" mean; should an attempt to divide by zero result in a reported error in FORTH-83, with the required sacrifice in speed? All of these questions (and more) have been answered differently by different system-authors, with the result that there are many different "standard" FORTHs on the market.

The standards must be viewed in light of their primary goal: to enable the writing of transportable code. While writing a text on both FORTH-79 and FORTH-83 for Prentice-Hall we have had to examine closely the differences in the standards. We have examined 12 different implementations of "standard" FORTH, and we have discussed the standards with the system-authors. This has led us to define more precisely the problems in relation to the goal of the standards. We have defined six problem areas and also feel we can define some solutions. The problem areas (with some typical examples) are as follows:

1) Standard words are so few that most implementations have double or treble the number of words required in the standards. And naturaly most programs call on those dialect-specific words. For example, FORTH-79 has 171 reserved words while FORTH-83 has 158. On the other hand MMSFORTH has 426 words, MVPFORTH has 396, and PC/FORTH has 472 (all without extensions). If a programmer working with MMSFORTH has to input a number during program execution she will use **#IN** rather than create a separate word using **EXPECT** and **CONVERT** . The program will not be translatable to another dialect, and only because of a very basic lack, a single word to allow numeric input during execution of a program.

2) Differences between the standards are small enough that many system-authors have not implemented FORTH-83. Others have implemented both standards using overlays. Of 11 standard FORTHs examined, 6 were FORTH-79, 3 were FORTH-83, and 2 allowed either standard. Thus the existance of FORTH-83 has in effect increased the number of incompatible dialects.

3) The lack of explicit documentation for subtle changes in FORTH-83 has caused confusion for both users and system-authors. The article by C. Kevin McCabe (FORTH-83: The evolution continues. Byte, August, 1984) clarifies many of the differences, but is not generally available. Many users seem to be confused about such things as loop termination, floored division, the lack of state-smart words such as ' and ." , and so on. At least one commercial FORTH-83 dialect has continued to treat ' as state-smart and has not implemented ['] . While this confusion is not a fault of the new standard per se, it does result in resistance to its implementation, and when the confusion is that of the system-authors it results in non-transportable and confusing code.

4) Some specifications are ignored as unrealistic. One of these is the implication that words such as ' and ." should not be state smart. At least two dialects continue to treat them as state smart, presumably with the reasoning "if it works, why change it?". We know of no dialect that limits **EMIT** to 7 bits, although that is explicitly required by the FORTH-83 document. Such deliberate but reasonable departures from the standards make for non-transportable source code.

5) Errors and ambiguities in the standards documents can lead to confusion. Some of the ambiguities are at worst humorous, since common sense gives the correct interpretation. For example "The minimum capacity of **TIB** is 80 characters" obviously means that the minimum capacity of the text input buffer should be 80 characters, not that **TIB** should have a parameter field 80 bytes long. And in the definition of **ROLL** "moving the remaining values into the vacated position" cannot mean that two 32-bit numbers are somehow moved into a 16-bit space. But there are more damaging errors. Probably the most blatant error is the specification of **BLANKS** as a reference word in FORTH-79 (to fill a specified number of bytes with ASCII 32). In all previous dialects and in FORTH-83 the word is **BLANK** , and presumably the "S" is a typographical error, yet some FORTH-79 implementations use **BLANKS** while others use **BLANK** . Other ambiguities have led to differences in implementations of FORTH-83. Does the standard intend that ' and ." can no longer be state smart? That is, can ." only be used within a colon definition? And is the action of ' to be deferred when it is compiled in

a colon definition? These are open to interpretation and have been handled differently in different implementations. And is it required that setting **BLK** to zero will immediately return interpretation to the keyboard? The standards are unclear and some dialects return to the keyboard while others do not. These ambiguities and resulting differences in FORTH dialects can make the language confusing, they decrease transportability of programs, and they can earn FORTH a bad name.

6) Most users and system-authors perceive a need for the standards to specify words for a wider range of uses. Thus nearly all dialects add floating-point arithmetic, most add some level of graphics, and many add string manipulation words (to sort, concatenate, extract, and search for strings, for example). And most dialects now work from within an operating system, with words for such things as loading files of blocks, making a specific file the one "seen" by FORTH as containing blocks, displaying the operating system directory, and exiting FORTH to the operating system. Some of these added words are fairly uniform between dialects simply because of common sense. Thus **F+** is used to add two floating point numbers, **DIR** displays the operating system directory, and **BYE** exits from FORTH. Since virtually all programs written in these dialects will use some of the dialect-dependant words, virtually none of the programs are transportable. While it is understandable that the standards define a minimal set of words with the idea that the standards should specify what FORTH should do rather than how it should be done, and while a goal of the standards is to allow the user or system-author to develop FORTH as desired, it can be argued that unless extensions are added to the standards, "modern" FORTHs will largely result in non-transportable programs and the goal of the standards will be defeated. We found that in order to write a usable textbook we had to describe not only FORTH-79 and FORTH-83 but also a typical enhanced dialect (we somewhat arbitrarily chose MMSFORTH with examples from other dialects as well). Even then the usefulness of the text (and any text) is limited by the variety of different dialects that may be used by students. No single book can describe them all.

We feel that the solutions to these problems are fairly straight forward, although they will require an effort of the standards team. First, we do not feel that a new standard is needed. That would only decrease the transportability of code by increasing the number of dialects. We feel that nearly all of the problems can be eliminated by a clarification of the FORTH-83 standard document and by the addition of extensions, which a particular dialect may or may not include while still meeting the standard.

The standard document should be clarified in three ways: 1) Ambiguities (such as mentioned in (5) above) should be removed from definitions by careful editing. 2) Terminology must be carefully defined; for example what does "an error condition results" imply, what does it mean that "sys is balanced", what is a "compilation address", and so on. (For that matter, name-field, code-field, and so on should be defined, since they are used in the document.) 3) The differences between FORTH-79 and FORTH-83 should be clearly and unambiguously stated in the standard document. We recognize that it is nearly impossible to write a completely unambiguous standard document, and for that reason we

suggest in a separate paper that a "test-suite" should be created that will test whether a dialect meets the standard.

Finally we feel that extensions should be added to the standard for such things as floating-point arithmetic, graphics, strings, and use within an operating system. It would then be possible for a dialect to be specified as standard with the exception that certain extensions are not included, just as Microsoft FORTRAN is specified as standard FORTRAN with the exclusion of imaginary numbers. It is clear, for example, that many floating-point words are used in common by nearly all dialects and could easily be standardized. Dialects would still be free to add to the extensions, and for that matter, to ignore an extension and still be standard.

If these clarifications and extensions of FORTH-83 were made FORTH programs would be much more easily transported between systems, FORTH implementations would not interpret the standards differently, FORTH would be much easier to teach and learn, and programmers would have much less trouble when switching between dialects. But the changes must be made quickly, for the acceptance of certain widespread implementations otherwise may force a de-facto but less than optimal standard. Finally, extending FORTH-83 would not remove any of the flexibility and extensibility so prized in FORTH; users would be free to ignore or modify the standard extensions for their own purposes or even to use only the required words, and system-authors could add to or replace the standard extensions if needed.