Special Words in Forth

Stephen Pelc
MicroProcessor Engineering
133 Hill Lane
Southampton SO15 5AF
England
t: +44 (0)23 8063 1441
e: sfp@mpeforth.com
w: www.mpeforth.com

Abstract

Over the last few years, I have become convinced that I do not understand the ANS Forth description of compilation and how this situation came about. The Forth 2012 description of compilation is the same as that of ANS. This paper describes the process of understanding that leads to being able to make a few proposals to make use of a new description of compilation. In essence, we are going to have to regard IMMEDIATE as a special case of our new situation. The model also allows us to build words that would previously have had to be state-smart.

Introduction

The Forth94 (ANS) and Forth 2012 standards talk about execution of a word in terms of semantics. In the Oxford dictionary, we find the definition of semantics to be:

The branch of linguistics and logic concerned with meaning. The two main areas are logical semantics, concerned with matters such as sense and reference and presupposition and implication, and lexical semantics, concerned with the analysis of word meanings and relations between them.

Wikipaedia says:

In **programming** language theory, **semantics** is the field concerned with the rigorous mathematical study of the **meaning** of **programming** languages. It does so by evaluating the **meaning** of syntactically legal strings **defined** by a specific **programming** language, showing the computation involved.

In terms of undestanding Forth standards, these do not help much. In practice semantics means action or behaviour. From the Forth 2012 standard:

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

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

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

In this paper we use semantics, behaviour and action interchangeably.

MPE's VFX code generator was written in the late 1990s just as the Forth94 standard was being adopted by most vendors. In particular, VFX took advantage of the then new word **`COMPILE,`** to attach code generators for a range of words. It did this while preserving the classic Forth interpreter loop, or so we thought.

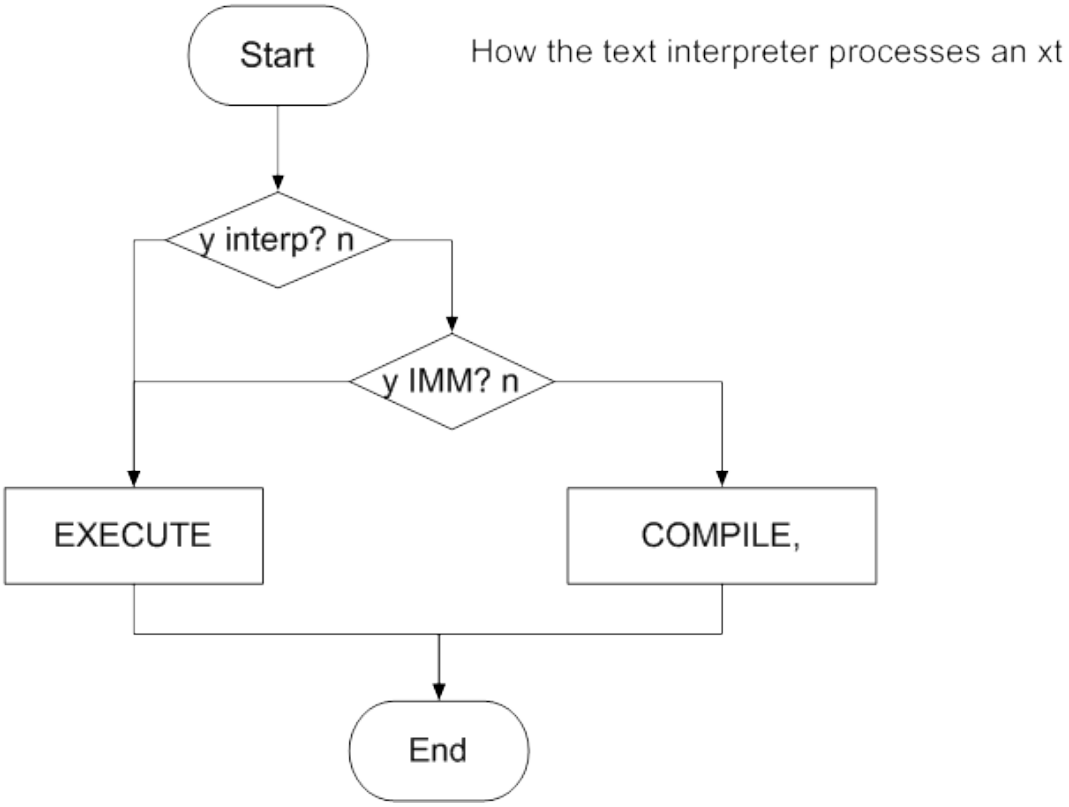

Illustration 1: Classical Forth interpreter loop

```
: process-xt    \ i*x xt -- j*x
  state @ 0 = if
    execute
  else
    dup immediate?
    if  execute  else  compile,  then
  then
;
```

The classical Forth interpreter loop has been used to describe the operation of Forth for over three decades now. It has been a useful model for many people. People regularly claim that they need to write a custom interpreter and that not all Forth systems permit this in a portable manner. We will see that a minor change to the loop and its associated structures brings it in line with Forth 2012 and expands the interpreter's facilities to take advantage of the Forth 2012 description of Forth words' action or behaviour or semantics.

Although this paper describes the interpreter in terms of the classic Forth interpreter loop, it should not be assumed that other techniques for writing interpreters are excluded. Exactly the same problems and solutions are present in techniques with different organisations including recognisers.

Smart COMPILE,

VFX Forth and other Forths take advantage of **`COMPILE, ( xt -- )`** by attaching optimisers to the words that they generate code for. For example, the word **`DUP`** has a word **`C_DUP`** that generates code for **`DUP`**. The xt for **`C_DUP`** is attached to **`DUP`**. Then when **`COMPILE,`** looks at **`DUP`** it executes **`C_DUP`** to generate the code for **`DUP`**.

The smart **COMPILE,** introduces the idea that a word (identified by one primary xt) may require one or more secondary xts. It has become common practice in desktop Forths for dictionary headers to contain more than just link, name and flags. This trend is particularly true in Forth systems that perform native code compilation (NCC).

The smart **COMPILE,** can completely separate the interpretation (execution of **DUP**) and compilation actions of a word. This technique can also be used for other words such as **IF**, with the deliberate intention that the interpretation and compilation actions of a word can be separated. However, **COMPILE,** is then broken as far as current standards are concerned because structure words such as **IF** produce or consume stack items, and string words parse the input stream. There may/will also be corner cases to do with **POSTPONE**.

Standards issues

The use of the smart **COMPILE,** for optimisation is not contentious. However, it opens a box that cannot and should not be closed. The Forth94 standard introduced a new way of talking about Forth words. Words have a number of actions, including interpretation and compilation actions. The only standard way to separate interpretation and compilation actions is, paradoxically, to define them as being the same and then to use **STATE** to separate them within the word. This is the state-smart nightmare that leads to bugs which are hard to find.

In the Forth94 and Forth 2012 world, very few words are defined as **IMMEDIATE** and there is no standard way to ask the system if the xt of a word is of an **IMMEDIATE** word.

In terms of the classical loop shown above, the only place at which non-default compilation semantics can be attached is **COMPILE,** and the system immediately becomes contentious, not least because some people insist that **IF** must be **IMMEDIATE** without stating any evidence for this. Another way to look at the problem is to state that the language of the standard does not match any Forth implementations except cmForth and Gforth. Chuck Moore's cmForth is as idiosyncratic as all Chuck Moore's other tools and was obsolete at the time of the ANS standard. Gforth's original design target was to be a model implementation of the Forth94 standard, i.e. the standard is correct. In my opinion this design target has lead to complexity. Correcting the disconnect between the current standard and real Forths while maintaining simplicity is the function of this paper.

A way forward

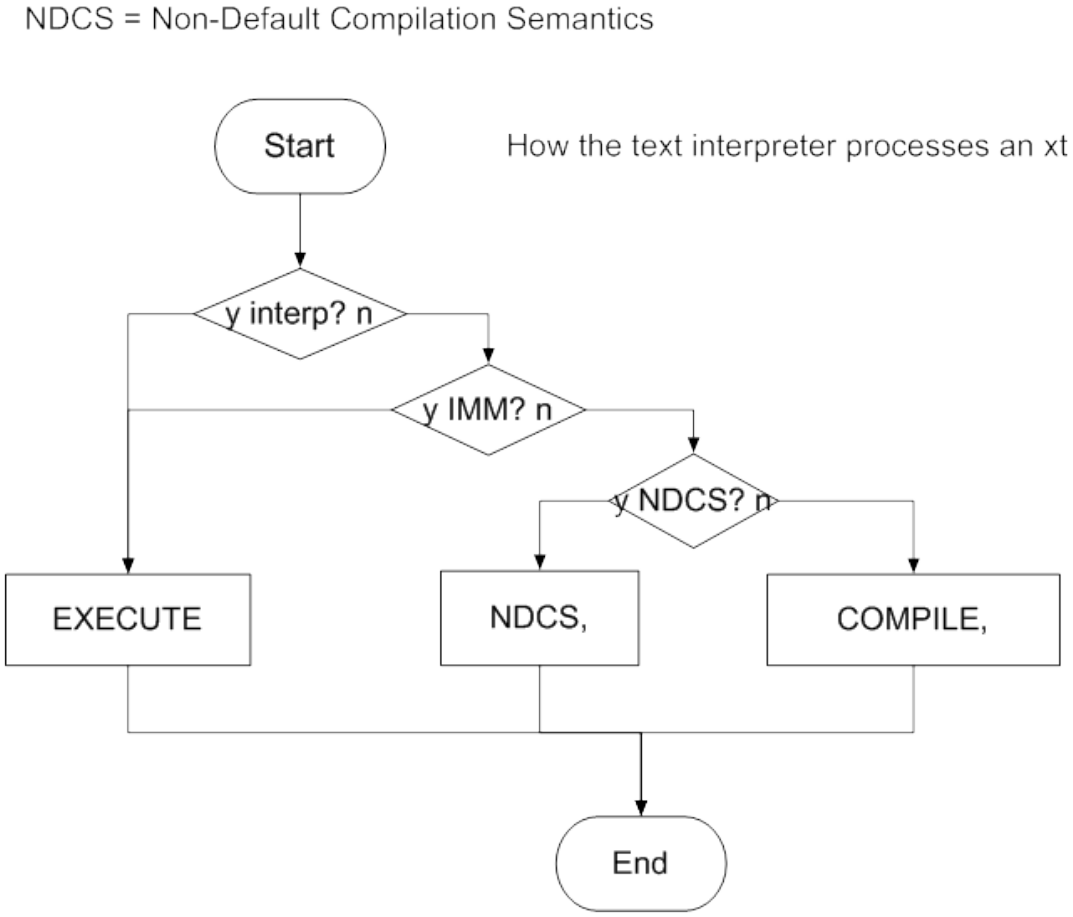

Illustration 2: Allowing for the Forth94 and Forth 2012 standards

```
: process-xt    \ i*x xt -- j*x
  state @ 0 = if
    execute
  else
    dup immediate? if
      execute
    else
      ndcs?
      if  ndcs,  else  compile,  then
  then
;
```

The picture illustrates a Forth interpreter/compiler loop that has been modified to cope with separated interpretation and compilation actions.

We also need a small number of new words that enable the loop to be constructed portably:
`IMMEDIATE?  Xt -- flag` ; return true if the word is immediate
`NDCS?  Xt -- flag` ; return true if the word has non-default compilation semantics
`NDCS,  i*x xt -- j*x` ; like **`COMPILE,`** but may parse.

In order to finish up, we need to understand what the word labelled **`NDCS,`** actually does. It finds the word that performs the non-default compilation semantics and then **`EXECUTE`**s it.

The next picture shows the loop using the definition of **IMMEDIATE** words as having the same interpretation and compilation semantics.

The significant change is the introduction of a dictionary header flag, NDCS, which indicates that a word has non-default compilation semantics.

Replace IMMEDIATE with NDCS

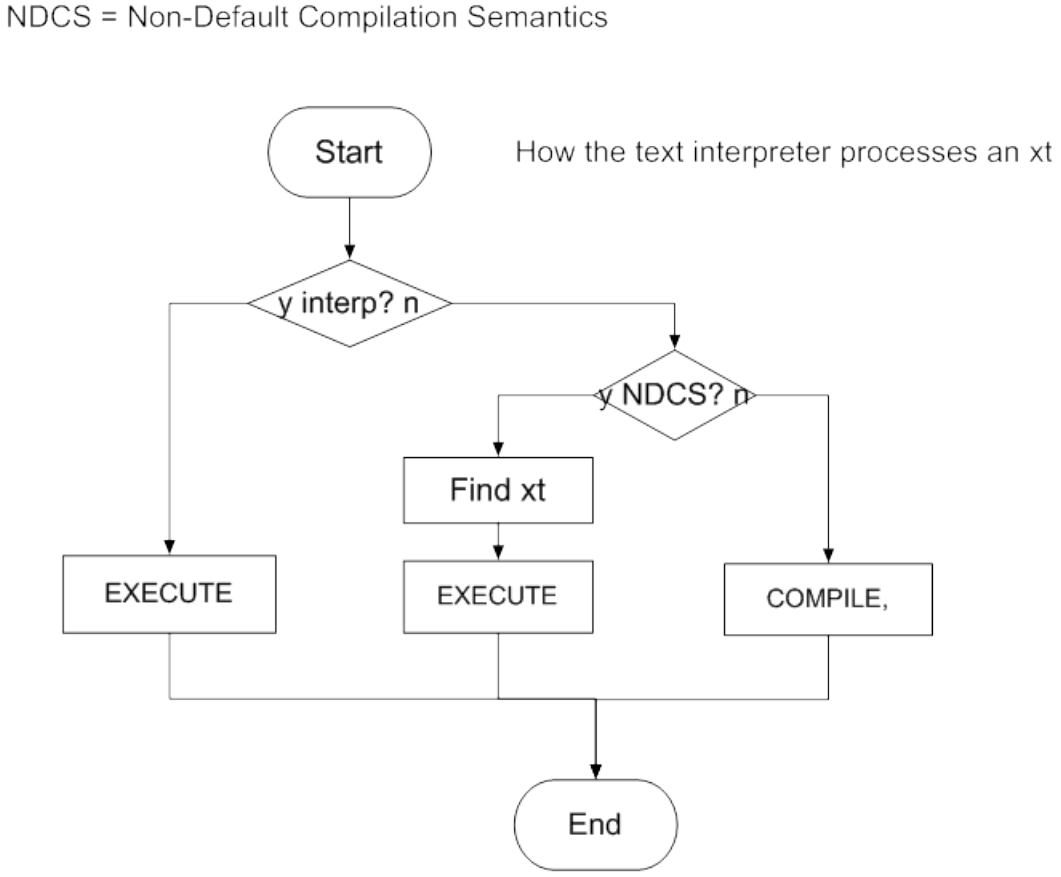

Illustration 3: The IMMEDIATE flag becomes the NDCS flag

The two boxes "Find xt" and "EXECUTE" were called NDCS, in the previous diagram. Here they are exposed to show that non-default compilation semantics are found in a system-specific manner.

```
: process-xt    \ i*x xt -- j*x
  state @ 0 = if
    execute
  else
    dup ndcs?
    if  find-ndcs-xt execute  else  compile,  then
  then
;
```

The immediate flag has disappeared because all immediate words have non-default compilation semantics. They are immediate if the NDCS xt is the same as the for interpretation xt. The definition of immediate is more complicated in standards-speak, but comes to the same thing. An alternative implementation strategy may be to keep a separate

immediate flag, but we should not hide the basic idea that immediate words have non-default compilation semantics.

Several modern Forth systems have interpreter loops that would be easy to convert to the new requirements. Coming back to our three new words:
`IMMEDIATE?  Xt -- flag` ; return true if the word is immediate
`NDCS?  Xt -- flag` ; return true if the word has non-default compilation semantics
`NDCS,  i*x xt -- j*x` ; like **`COMPILE,`** but may parse. Used by words such as **`IF`**.

We can see that the conventional immediate flag in a word's header becomes the NDCS flag, set for all words that have non-default compilation semantics. Comparison of the interpretation xt and the NDCS xt gives us a basis for the word **`IMMEDIATE?`** The word **`NDCS,`** just hides the system-specific action of obtaining the NDCS action from an xt.

Using NDCS words

The NDCS words and the notation below allow us to construct NDCS words, both for system use and for general use. It is worth considering whether a library or an application may want to construct NDCS words. The most common case that we see is when an application needs a "Domain Specific Language" (DSL) which is Forth-based. Such a DSL may wish to provide interpreted as well as compiled versions of **`IF`** … **`ELSE`** … **`THEN`** and **`DO`** … **`LOOP`**.

In the past this type of notation has been shunned because it required state-smart words. Words that use NDCS correctly in two portions that do not test **`STATE`** are not state-smart. Therefore the reasons to avoid such notations only have to do with programming taste and overcoming the limitations of 20+ years of dogma. The dogma arose because we did not have the structures to separate interpretation and compilation actions, even though the Forth94 and Forth 2012 standards described compilation in those terms. Once we have Forth words to implement such ideas, we can move forward.

Here's a potential way of building NDCS words. They illustrate a conventional **`IF`** … **`THEN`** pair. The word **`NDCS:`** modifies the previous word to have the following non-default compilation semantics – it defines a nameless word and sets system-specific flags and data.

```
: IF      \ C: -- orig ; Run: x --
\ This is the traditional interpretation behaviour
  NoInterp  ;
ndcs: ( -- orig )  s_?br>,  ;     \ conditional forward branch

: THEN     \ C: orig -- ; Run: --
\ This is the traditional interpretation behaviour
  NoInterp  ;
ndcs: ( orig -- )  s_res_br>,  ;  \ resolve forward branch
```

To produce an interpreted version, the interpretation behaviour is simply replaced by the new version. The next example shows how a contentious notation such as **`S"`** and friends becomes non-contentious.

```
: S"            \ Comp: "ccc<quote>" -- ; Run: -- c-addr u
\ Describe a string. Text is taken up to the next double-quote
\ character. The address and length of the string are
\ returned.
  [char] " parse >syspad
;
ndcs: ( -- ) postpone (s") ",  ;
```

Words we need to consider

```
: set-compiler  \ xt --
\ Set xt as the compiler (by COMPILE,) of the last
\ definition. The word whose xt is given receives
\ the of the word it is to compile ( xt -- ).
\ Used to define optimisers.

: comp:         \ xt --
\ Starts a nameless word whose xt becomes the compiler
\ for the last definition.

: set-ndcs      \ xt --
\ Set xt as the NDCS action of the last definition.
\ The word whose xt is given to SET-NDCS has the stack
\ action: i*x -- j*x

: ndcs:         \ i*x xt -- j*x
\ Starts a nameless word whose xt becomes the NDCS
\ action for the last definition.

: IMMEDIATE     \ --
\ Mark the last defined word as immediate by
\ setting the NDCS flag making the NDCS xt the same
\ as the interpretation xt.

: IMMEDIATE?    \ Xt -- flag
\ Return true if the word is immediate.

: NDCS?         \ Xt -- flag
\ Return true if the word has non-default compilation
\ semantics.

: NDCS,         \ i*x xt -- j*x
\ Like COMPILE, but may parse. Used to perform the action
\ at compile time of NDCS words such as IF.

: SEARCH-NAME   \ c-addr len -- ior | xt 0
\ Perform the SEARCH-WORDLIST operation on all wordlists
\ within the current search order. On failure, just an ior
\ (say -13) is returned. On success, the word's xt and 0
\ are returned.
```

During a review on the Forth200x mailing list, nobody liked the acronym NDCS for these words. The phrase "special complation semantics" was much preferred instead. I have left NDCS in the paper because Forth 2012 refers to "non-default compilation semantics" throughout. When the standard uses the new phrase, then the words above can change names.

Consequences for compilation

We can now define what happens during compilation of a Forth word, i.e. what happens when a source token has been recognised/found and the system is in compilation state.

NDCS = Non-Default Compilation Semantics, i.e. a special word

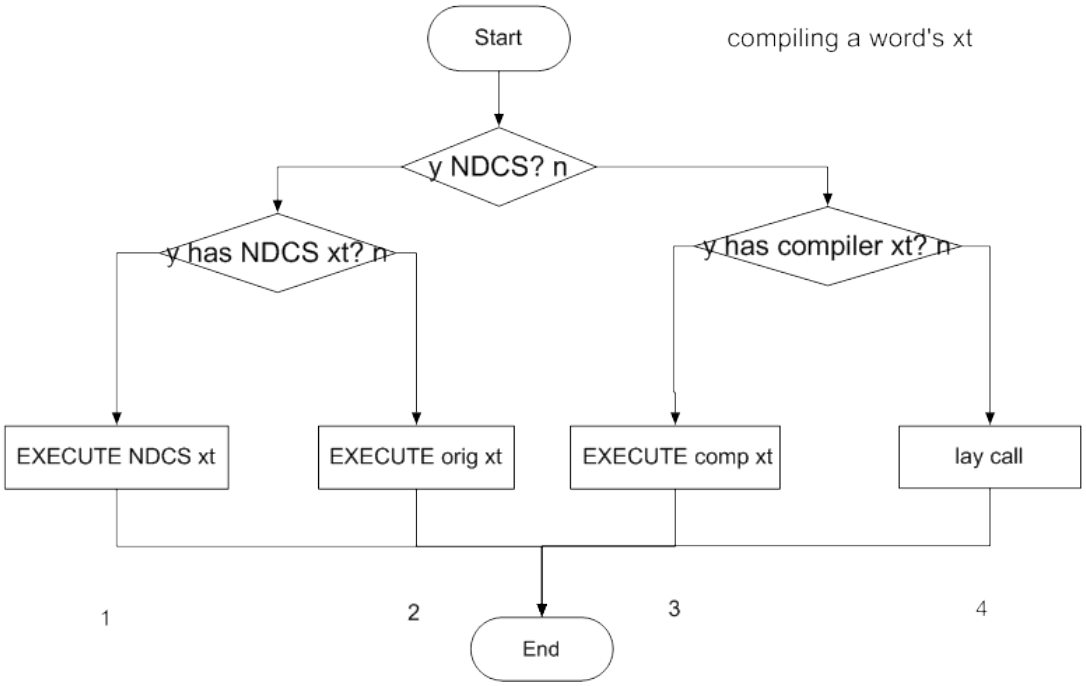

Illustration 4: Forth compilation

1. The word has an NDCS flag and an NDCS-specific action has been defined for it. The NDCS specific action is executed. These could be the compile-time actions of **`IF`** or **`S"`**.
2. The word has an NDCS flag but no NDCS-specific action has been defined for it. In this case the word's original xt is executed, corresponding directly to the current definition of an immediate word.
3. The word is normal and a code generator has been specified. The code generator is executed to lay down the required code.
4. The word is normal and no code generator exists. We just lay a Forth call to this word.

Cases 1 and 2 form the action of **`NDCS,`** in this paper. Cases 3 and 4 form the action of **`COMPILE,`** in the standard.

The test for an NDCS xt is optional. If a system can guarantee that all NDCS words have a separate xt for the NDCS portion, case 2 never happens and the check can be omitted. Similarly, systems without code generators can omit case 3.

Embedded and minimal systems

If we treat the NDCS flag as equivalent to the old immediate flag, then a minimal system can just provide cases 2 and 4 above. If such systems wish to provide both compilation and interpretation actions for words such as **`S"`** they can fall back to state-smart words, probably as they have always done.

Conclusions

The Forth94 standard (ANS) introduced the idea of "non-default compilation semantics" (NDCS) to the Forth world. However, the standard provided no facilities for dealing with NDCS words. NDCS makes immediate words a special case of NDCS. Simple changes to the Forth interpreter allow us to deal with and specify NDCS words. The classical Forth interpreter loop picture (Figure 1) needs a small change (Figure 3), and we need to introduce the words **`NDCS?`** and **`NDCS,`** to complete the picture.

Correct use of NDCS words also allows us to implement words such as **`S"`** without them being state-smart. This in turn permits us to define notations that have been deprecated for the last 20 years or so.

In implementing NDCS behaviour we find that immediate words are just a special case of NDCS. We can usefully remove the immediate flag and replace it with an NDCS flag.

It would be of benefit if we can find a name other than NDCS to describe "non-default compilation semantics".

Acknowledgements

Anton Ertl has tested my understanding of Forth standards for many years.

My belief that all standards contain bugs has sustained me over many years.

Anton Ertl, Bernd Paysan, Graham Smith and Gerald Wodni provided valuable comments on early drafts of the paper.