Improving the Understandability of Forth Code

John Bowling, Starlight Forth Systems
15247 N. 35th St, Phoenix, Az 85032

One of the most often vocalized complaints about Forth is that it is unreadable, and therefore unmanagable because no one but the original author understands it. An example of this is metacompiling: One of the least understood and most complex functions in Forth.

A couple of years ago, I was looking for a Meta-Compiler to compile Forth Code for the 65SC816 using a standard fig-Forth for the 6502. I found less than a half dozen available, most not set up for the 6502, and none readily able to handle the type of cross-compilation that I wanted to do. Also, on those where I was able look at the source code, I found them almost impossible to interpret. The comments that were there were just better than none at all. This implied that modification to what I needed was out of the question.

Background: Operation of a standard Forth compiler

Standard compilation in Forth searches the current and context dictionaries for the words used in the new word. If found, the precedence bit determines it they require immediate execution or may be compiled. If not IMMEDIATE the specified word's CFA is put into the new word's address list. If the word was COMPILE the next word will be compiled even though it is an immediate word.

If the word is not found, NUMBER is executed to determine if it is a number that is valid in the given numeric BASE . If it is, LIT or DLIT is compiled into the new word's address list, followed by the value. If it is not a valid number, an error message occurs.

Foreground: Target Compilation to a different processor

When target compiling for a different processor than the system processor, you can not allow the system to execute any of the new TARGET words.

To properly compile words in the TARGET, the TARGET is searched first. When the word is found, and if the precedence bit is off, the word's CFA is compiled. If a TARGET word is IMMEDIATE, rather than execute, it triggers a search of a special dictionary, TIMM, for Target Immediate where words are imititaions of the IMMEDIATE words in the TARGET. The words execute completely within the HOST vocabularies, and compile TARGET address. DO and LOOP, which manipulate the stack and compile addresses for (DO) and (LOOP) into the TARGET , are two examples.

If not found in the special dictionary, search continues with META , ASSEMBLER and host FORTH . In this case the word does not involve compilation and word can be a standard HOST word, but must not be executed unless it is immediate. Commonly used words that are acceptable are the comment words (and \ . Words in the special TIMM dictionary may search TARGET or check for LABELS compiled into the ASSEMBLER for the addresses it needs to compile.

If the specified word is not found in the target, a search of the HOST is made. If found and if the precedence bit is set the word will be executed. If the precedence bit is not set, then a warning message is output. HOST addresses must never be compiled into the TARGET. If not found in the HOST , TNUMBER is executed, and the address of target LIT or DLIT is placed in the word list followed by the value.

One of the ways of making code more understandable is to create a Logical English Pseudo-Code translation. The following is English Pseudo-Code from the above Cross-Target compilation technique:

```
If Found                                       \ Word in Target
  If Immediate word                            \ Immediate, search
    Search Timm dictionary                     \ Target IMMediate
      If Found Execute                         \ Execute TIMM word
        If Address Available Compile Address and Complete
        Else Search Target                     \ Address not available
          If Found Compile Address and Complete
          Else Search Assembler for Lables \  not in Target
            If Found Compile Address and Complete
            Else Notify of Error and Exit  \ Not Address Label
            End If
          End IF
        End If
      Else Search Host              \ Not found in TIMM dictionary
        If Found
          If Immediate Word Execute \ With no Effect on Target
          Else Notify of Error and Exit    \ Not Immediate
                                           \ Host word
          End If
        Else                                   \ Not Found in Host
          If Valid Number
            Execute Tnumber          \ Compile Target LIT value
          Else Notify of Error and Exit    \ Not Numeric
          End If
        End If
   Else Compile Into Target                    \ Not Immedate Word
   End If
```

```
Else Search Host                     \ Not Found in Target
  If Found
    If Immediate Word Execute        \ With no Effect on Target
    Else Notify of Error and Exit \ Not Immediate Host
    End If                                   \ word
  Else                               \ Not Found in Host
    If Valid Number
      Execute Tnumber                \ Compile Target LIT
                                     \ or DLIT and value
    Else Notify of Error and Exit            \ Not Numeric
    End If
  End If
End If
```

Translation of this Pseudo-Code into Forth Code is not difficult. Understanding the resultant Forth Code, if you have access to the douments that defined the problem and execution, is not impossible. This code, because of the deeply nested conditionals, is very difficult to debug. There are better ways!

To improve understanding of code similar to the above, another translation step is needed. This time we have eliminated the multiple nested conditionals through the use of Case statements. Case operators allow us to select one of several operations based on a value passed on the stack, without treading our way through several conditionals.

Execution of Timm words:

```
CASE  Search Assembler   \ Search for Label for address
   Compile Address       \ Found Label in Assembler
   Error                 \ No Label in Assembler

CASE  Search Target      \ Search for Word in target
   Compile Address       \ Found Word in Target
   Search Assembler      \ Try for a Label

CASE  Timm Execut        \ Get needed address
   Compile Address       \ Found address, stuff
   Search Target         \ Not a Timm word, go search Target

CASE  Search Host
   Error         \ Non-Immediate Host word:  Error
   Execute       \ Execute the Immediate Host word
   Target Number \ Not a Host word,
           \ Try to convert to a number using Target Literal

CASE  Search Timm
   Error         \ Non-Immediate word, not valid for Timm words
   Timm Execute  \ Found: Execute a Timm word
   Search Host   \ Not a Timm word, Look for word in the Host
```

```
CASE  Search Target
   Compile        \ Non-Immediate word, compile
   Search Timm    \ Immediate Target word, DO NOT execute,
substitute
   Search Host    \ Word not found in Target, try Host
```

This is much easier to follow than the nested conditionals, yet still requires an understanding of what one is trying to accomplish. Understanding of the problem statement from the defining documents is still very helpfull.

To further improve understanding of this, we need to eliminate the multiple case functions. One way of doing this is through State Machines. State machines are simply a series of CASE functions, with each state representing one CASE statement. Another way of looking at them is to think of a double (or greater) subscript array, where the subcripts point to an executable function.

```
Execution of Timm words:

Search Timm command sets state        = 0
Search Target command sets state      = 1
Search Assembler command sets state   = 2

STATE:  Timm Execute            \ Is the Address Available?
\                Yes               No
\ State:
(  0 )      Compile      Search Target       \ Timm word
(  1 )      Compile      Search Assembler    \ Target Word
(  2 )      Compile      Error               \ Assembler Label

Compilation of Target words:

Search Target command sets state      = 0
Search Timm command sets state        = 1
Search Host command sets state        = 2

STATE:  Search Target

\ Search Result: Non-Imm    Immediate     Not Found
\ State:
(  0 )      Compile     Search Timm    Search Host    \ Target
(  1 )      Error       Timm Execute   Search Host    \ Timm
(  2 )      Error       Execute        Target Number \ Host
```

It looks almost too simple to do what the nested conditionals do, and it's much easier to read and understand, even without a definition document. It also eliminates many of the complaints about unreadable Forth code by presenting it in a form that is almost universal regardless of language.