Table of Contents

A.15 The optional Programming-Tools word set

These words have been in widespread common use since the earliest Forth systems.

Although there are environmental dependencies intrinsic to programs using an assembler, virtually all Forth systems provide such a capability. Insofar as many Forth programs are intended for real-time applications and are intrinsically non-portable for this reason, the Technical Committee believes that providing a standard window into assemblers is a useful contribution to Forth programmers.

Similarly, the programming aids DUMP, etc., are valuable tools even though their specific formats will differ between CPUs and Forth implementations. These words are primarily intended for use by the programmer, and are rarely invoked in programs.

One of the original aims of Forth was to erase the boundary between **user** and **programmer** - to give all possible power to anyone who had occasion to use a computer. Nothing in the above labeling or remarks should be construed to mean that this goal has been abandoned.

A.15.6 Glossary

A.15.6.1.0220 .S

.S is a debugging convenience found on almost all Forth systems. It is universally mentioned in Forth texts.

A.15.6.1.2194 SEE

SEE acts as an on-line form of documentation of words, allowing modification of words by decompiling and regenerating with appropriate changes.

A.15.6.1.2465 WORDS

WORDS is a debugging convenience found on almost all Forth systems. It is universally referred to in Forth texts.

A.15.6.2.0470 ;CODE

Typical use: `: namex ... <create> ... ;CODE ...`

where namex is a defining word, and <create> is CREATE or any user defined word that calls CREATE.

A.15.6.2.0930 CODE

Some Forth systems implement the assembly function by adding an ASSEMBLER word list to the search order, using the text interpreter to parse a postfix assembly language with lexical characteristics similar to Forth source code. Typically, in such systems, assembly ends when a word END-CODE is interpreted.

A.15.6.2.1015 CS-PICK

The intent is to reiterate a dest on the control-flow stack so that it can be resolved more than once. For example:

```
\ Conditionally transfer control to beginning of loop
\ This is similar in spirit to C's "continue" statement.

: ?REPEAT  ( dest -- dest ) \ Compilation
           ( flag -- )      \ Execution
    0 CS-PICK   POSTPONE UNTIL
; IMMEDIATE

: XX  ( -- ) \ Example use of ?REPEAT
    BEGIN
```

```
      ...
    flag ?REPEAT  ( Go back to BEGIN if flag is false )
      ...
    flag ?REPEAT  ( Go back to BEGIN if flag is false )
      ...
    flag UNTIL    ( Go back to BEGIN if flag is false )
    ...
;
```

A.15.6.2.1020 CS-ROLL

The intent is to modify the order in which the origs and dests on the control-flow stack are to be resolved by subsequent control-flow words. For example, WHILE could be implemented in terms of IF and CS-ROLL, as follows:

```
: WHILE  ( dest -- orig dest )   POSTPONE IF  1 CS-ROLL ; IMMEDIATE
```

A.15.6.2.1580 FORGET

Typical use: `... FORGET name ...`

FORGET assumes that all the information needed to restore the dictionary to its previous state is inferable somehow from the forgotten word. While this may be true in simple linear dictionary models, it is difficult to implement in other Forth systems; e.g., those with multiple address spaces. For example, if Forth is embedded in ROM, how does FORGET know how much RAM to recover when an array is forgotten? A general and preferred solution is provided by MARKER.

A.15.6.2.2531 [ELSE]

Typical use: `... flag [IF] ... [ELSE] ... [THEN] ...`

A.15.6.2.2532 [IF]

Typical use: `... flag [IF] ... [ELSE] ... [THEN] ...`

A.15.6.2.2533 [THEN]

Typical use: `... flag [IF] ... [ELSE] ... [THEN] ...`

Software that runs in several system environments often contains some source code that is environmentally dependent. Conditional compilation - the selective inclusion or exclusion of portions of the source code at compile time - is one technique that is often used to assist in the maintenance of such source code.

Conditional compilation is sometimes done with **smart comments** - definitions that either skip or do not skip the remainder of the line based on some test. For example:

```
\ If 16-Bit? contains TRUE, lines preceded by 16BIT\
\ will be skipped. Otherwise, they will not be skipped.

VARIABLE 16-BIT?

: 16BIT\  ( -- )  16-BIT? @  IF  POSTPONE \  THEN ;  IMMEDIATE
```

This technique works on a line by line basis, and is good for short, isolated variant code sequences.

More complicated conditional compilation problems suggest a nestable method that can encompass more than one source line at a time. The words included in the ANS Forth optional Programming tools extensions word set are useful for this purpose. The implementation given below works with any input source (keyboard, EVALUATE, BLOCK, or text file).

```
: [ELSE]  ( -- )
    1 BEGIN                              \ level
      BEGIN
        BL WORD COUNT  DUP  WHILE        \ level adr len
        2DUP  S" [IF]"  COMPARE 0=
        IF                               \ level adr len
          2DROP 1+                       \ level'
        ELSE                             \ level adr len
          2DUP  S" [ELSE]"
          COMPARE 0= IF                  \ level adr len
             2DROP 1- DUP IF 1+ THEN      \ level'
          ELSE                           \ level adr len
            S" [THEN]"  COMPARE 0= IF     \ level
              1-                         \ level'
            THEN
          THEN
        THEN ?DUP 0=  IF EXIT THEN        \ level'
      REPEAT  2DROP                      \ level
    REFILL 0= UNTIL                      \ level
    DROP
;  IMMEDIATE

: [IF]  ( flag -- )
0= IF POSTPONE [ELSE] THEN ;  IMMEDIATE

: [THEN]  ( -- )  ;  IMMEDIATE
```

Table of Contents

Next Section