Expanding the Forth dictionary beyond the 64k limit by using 'Bodiless' code

Robert Boni
Kodak Research Labs
Rochester, New York 14650

ABSTRACT

At the Kodak Research Labs (KRL), experimental test beds are controlled from Forth. The most complex test beds require a Forth system with a dictionary larger than 64 kilobytes of memory will allow. An existing Forth system, modified to produce bodiless code, now supports a dictionary of several megabytes. These modifications, which generate the bodiless code, provide a method to increase the dictionary space of a Forth system running on a computer with memory management capabilities.

This paper describes the modifications made to an existing Forth system to allow compilation and execution of source code beyond the normal dictionary space. High-level extensions are added to polyFORTH (a product of FORTH, Inc.) which runs on a DEC 11/23 plus computer. These extensions manipulate the Memory Management Unit (MMU) while source code is loading and provide for a linkage between definitions in high memory and the Forth dictionary. Using these extensions, most of the addressable memory can become extended dictionary space.

These extensions consist of three basic commands:

1) [MODULE] <name>
2) LINK
3) [END]

The word [MODULE] executes immediately before loading the source code to be compiled into high memory. Following the [MODULE] command is a word which is the name given to the section of code being loaded. After a word is defined in a module, LINK makes this word from high memory part of the Forth dictionary by linking it to the Forth dictionary. The word [END] identifies the end of the module being loaded into high memory. No word follows the [END] command.

For the purpose of illustration, here is a module using these commands:

```
[MODULE] EDITOR        ( a large full screen editor   )
```

The source code for the editor is placed in this section following the [MODULE] command. The source code is written exactly as if it were to be compiled into the normal Forth dictionary.

```
: EDIT                    ( word to invoke the editor    )
      START.EDITOR ;      ( start the editor             )
LINK                      ( link EDIT to the dictionary  )
[END]                     ( end high memory compiling    )
```

The word EDIT -- and all other definitions in this module -- will have headers and compiled Forth code in high memory. There will be a new word in the Forth dictionary with the same name -- EDIT. Using the word EDIT will cause the Forth system to memory map to the module EDITOR and execute the high memory definition EDIT. When the definition of EDIT is finished executing, EDIT restores the original memory mapping.

If a word defined as part of a high memory module is linked to the Forth dictionary, it can be used in a definition that is part of a different high memory module. A simple example of this capability follows:

```
[MODULE] ALPHA            ( name for the first module    )

: ALFRED                  ( colon def. in this module    )
       ." I am Alfred" ( this word identifies itself  )
       ." Who are you?" ; ( asks who else is there    )
LINK                      ( links Alfred to the dict.    )

: ALICE                   ( second def. in this module   )
       ." Hello, I'm Alice"   ( this word says hello   )
       ." Where is Alfred?" ; ( Alice asks for Alfred  )

[END]                     ( note: ALICE not linked       )

[MODULE] BETA             ( second module in high memory)

: BRUNO                   ( first name, second module    )
      ALFRED              ( use def. from first module   )
  ." This is Bruno, Alfred" ; ( Bruno answers question )
LINK                      ( link BRUNO to the dictionary )

: BRENDA                  ( this definition will fail    )
      ." Is that you Alice?" ( Brenda sking for Alice )
      ALICE ;             ( ALICE not linked, this fails )

[END]                     ( end this second module       )
```

When the definition BRENDA tries to compile the word ALICE , the error message -- ALICE? indicates that the word cannot be found. When words are defined in one module and used in other modules, they must be linked to the dictionary using the word LINK.

Only the amount of addressable memory limits the number of different modules allowed. Different modules are needed because the 11/23 memory manager can only map 8k sections of memory at a time with each MMU register. In terms of loading Forth source code, this MMU feature means that there must be less than 8k bytes of compiled source code between each matched [MODULE] and [END] command. This is not as limiting as it sounds because array and buffer space can be allotted outside a module.

The manipulations performed by the words [MODULE] , LINK and [END] are remarkably simple as can be seen in the following step-by-step explanation of how a module is created.

[MODULE] TEST (Part of the source code)

A dummy word ZZZZ is defined in the Forth dictionary.

HERE is moved to address 40k.

The section of memory from 40k to 48k is mapped to high memory. The module is now ready for definitions.

The word TEST is automatically defined as the first word in the module.

All the following definitions are compiled.

: NEW.WORD ; LINK (This word in the source code)

LINK saves the Name Field Address (NFA) of NEW.WORD in a buffer.

Other words in the module TEST can be linked.

[END] (Part of the source code)

The command FORGET ZZZZ executes, restoring HERE and breaking vocabulary links to high memory.

The NFAs saved in the buffer are used to create new words in the Forth dictionary with the same names as their high memory counterparts.

These new words will memory map to the module TEST and execute the original definitions when executed.

The module is closed. The system has returned to normal operation.

By dividing a Forth application into well-defined modules with only a few words from each module being linked, the effective dictionary space can be increased by a factor of at least one hundred. Modules with only one word linked to the dictionary can provide a five hundredfold increase in dictionary space. If jumping between modules is minimized, the reduction in execution speed will be negligible.

This method of extending the dictionary space of a Forth system represents only one of many possibilities. Its primary strengths are that no changes need to be made in the Forth nucleus and very few are required in the existing source code.