Technical Notes

Vectored Versions of Forth Compiling Words

N. Solntseff

Unit for Computer Science
McMaster University
Hamilton, Ontario L8S 4K1

The words C, and , are extensively used by the Forth compiler interpreter for inserting 8- and 16-bit values into the dictionary. In both cases, values are transferred from the top of the data stack into the next available location in the dictionary. The address of the latter is given by the user variable DP or the auxiliary word HERE .The word ALLOT results in the allocation (but not initialization) of a group of dictionary locations. There are several other words in the model implementation (see [RAG80]) which are defined in terms of C, and , .

There are occasions when the fact that compilation can only be done into dictionary locations is too restrictive. A good example is provided by a meta-compiler where two separate dictionaries have to be used—one for the executing meta-compiler and the other for the Forth system being generated ([BOU80], [BAS81], [CAS81]). The need to keep the two separate and to ensure that the correct one is used at each stage of the meta-compilation leads to considerable complications in meta-compiler design. Vectored execution has been proposed as a means of simplifying the structure of meta-compilation ([BIL81], [BAS81]). This note presents an alternate approach to the use of execution variables which can be applied in the case of meta-compiler design, but, in addition, can lead to simplification in other cases where "remote compilation" is required.

Figure 1 (screen 120)[1] gives the definition of an execution variable ^CP which can be set to the address given by the dictionary pointer DP , if compilation into the dictionary is required. In other cases, it can be set to the address given by any other variable when compilation into an arbitrary range of addresses is required. The latter action can be achieved through the execution of the statement

CP <--- <address>

where <address> specifies the next location that is available for compilation. After the above statement is executed HERE , as defined in screen 121, will leave <address> on the data stack. By analogy with DP , the new variable CP can be called a compilation pointer. Similarly, n ALLOT will change the value of CP by n bytes.

Figure 2 provides an example of the way in which the new versions of the words shown above can be used. It contains the definition of an intelligent word !" for moving a string of the form $ ... $, where $ is an arbitrary delimiter, to memory locations specified by the contents of CP . If the system is in an interpreting mode, the input stream is the source of the move, otherwise the run-time word (!") followed by the string itself are compiled into locations given by CP . The word INPUT used in the definition of !" has been introduced in [SOL83]. It returns the address of the current input-stream buffer. When executed, the word (!") will move the string immediately following it to the locations specified by value contained in CP at the time of execution. The use of vectored versions of C, and , in the definition of the words described above gives the user great flexibility in specifying the target address for compilation or the target address for string moves.

Lastly, Figures 1 and 2 illustrate the programming style recommended in [SOL82].

*Received May 1983.

[1] FIG FORTH and 79-Standard has been used for the screens given in this note.

References

[BAS81] J. Basile and H. Goodell, "Pre-FORTH — A Vectored FORTH compiler," *Proc. 1981 FORML Conference*, FIG, San Carlos, CA 94070 (1981), pp. 303-319.

[BIL81] W. Bilobran, "Execution Variables," *Proc. 1981 FORML Conference*, FIG, San Carlos, CA 94070 (1981), pp. 245-255.

[BOU80] J. Boutelle, "Does a FORTH Metacompiler Represent a Metaprocess, or a Process of Self Reference?" *Proc. 1981 FORML Conference*, FIG, San Carlos, CA 94070 (1980), pp. 111-121.

[CAS81] J. J. Cassady, *METAFORTH — A Metacompiler For FigFORTH*, J. J. Cassady, Orinda, CA 95403 (1981), 76 pp.

[RAG80] W. F. Ragsdale, *Fig-FORTH Installation Manual — Glossary Model*, Forth Interest Group, P.O. Box 1105, San Carlos, CA 94070 (November 1980).

[SOL82] N. Solntseff, "Forth Programming Style," *Dr. Dobb's Journal*, 7, No. 9 (September 1982), pp. 30-32.

[SOL83] N. Solntseff, "An Improved Version of the Standard Word WORD," *Journal of Forth Application and Research*, submitted for publication (1983).

Figure 1

```
SCR # 120
 0 ( New version of compiling words --              120/ NS/06feb83)
 1
 2 FORTH DEFINITIONS
 3
 4  :  <---                                    ( adr --; store CFA of following word into adr)
 5       ( adr) [COMPILE]
 6       ( adr) STATE @ IF  COMPILE CFA COMPILE SWAP COMPILE !
 7       ( adr)         ELSE CFA SWAP !
 8                      ENDIF ; IMMEDIATE
 9
10 VARIABLE  ^CP                              ( execution variable to select compilation point)
11
12  : CP  ^CP @ 2+ EXECUTE ;                          ( "alternate" compilation pointer)
13
14 --->
15

SCR # 121
 0 ( New version of compiling words --              121/ NS 06feb83)
 1
 2  ^CP <--- DP                                        ( set compile pointer to dictionary))
 3
 4  : HERE  CP @ ;                              ( address of next available loc in compile area)
 5
 6  : ALLOT ( n) CP +! ;                                ( allocate n bytes in compile area)
 7
 8  : ,    ( word) HERE ! 2 ALLOT ;                      ( move word to compile area)
 9
10  : C,    ( byte) HERE C! 1 ALLOT ;                     (move byte to compile area)
11
12  : COMPILE  ?COMP R> DUP 2+ >R @ , ;                  ( move CFA to compile area)
13
14  : [COMPILE]  -FIND 0= 0 ?ERROR DROP CFA , ; IMMEDIATE
15
```

Figure 2.

```
SCR # 123
 0 ( Perform/compile string move --                    123/ NS/08feb83)
 1
 2 FORTH DEFINITIONS
 3
 4  : (!")                                    ( -- ; store string following starting at CP)
 5                        R DUP COUNT                  ( get fwa of string and length)
 6 ( fwa fwa+1 len)       DUP R> 1+ + >R                ( adjust return address)
 7 ( fwa  fwa+1 len)      >R DROP                       ( save character count)
 8 ( fwa)                 HERE R> 1+                    ( get string length)
 9 ( fwa here  cnt)       CMOVE ;                       ( store string in memory)
10
11 --->
12
13
14
15

SCR # 124
 0 ( Perform/compile string move --                    124/ NS/08feb83)
 1
 2  : !"                                      ( -- "string" ; store string at MP)
 3                      INPUT IN @ +                    ( get start of string)
 4 ( fwa)               DUP @                    ( obtain delimiting character)
 5 ( fwa del)           ENCLOSE                        ( find ending delimiter)
 6 ( fwa n1 n2 n3)      IN +!                  ( step over string and delimiter)
 7 ( fwa n1 n2)         OVER - >R              ( calculate and save string length)
 8 ( fwa n1)            + HERE                              ( leave string fwa)
 9 ( fwa here)          STATE @                ( in compile or interpret state?)
10 ( fwa here B)        IF     COMPILE (!") 2+           ( compile run-time code)
11 ( fwa here)                 R 1+ ALLOT          ( reserve space for string)
12 ( fwa here)          ENDIF
13 ( fwa here)          R OVER C! 1+ R> CMOVE ;                ( move string to HERE)
14                      IMMEDIATE
15
```