FORTH - 79 STANDARD CONVERSION

V. 1.1
9/15/81

FORTH-79 STANDARD CONVERSION

Robert L. Smith

Version 1.1

September 15, 1981

FORTH-79 STANDARD CONVERSION

There are a number of owners of fig-FORTH systems who desire to convert over to FORTH-79 Standard. This package is designed to aid the conversion process. The principal part of the package is a series of screens which can be loaded on most fig-FORTH systems. When these screens are loaded and the other changes made as indicated below, your system should conform to the official designation:

FORTH-79 Standard with Double-Number Standard Extensions

Using the system will allow you to create and load FORTH-79 Standard programs. Note however that there is not extensive error checking in this package. Users must take some care in writing application programs to ensure that the program meets the requirements of the FORTH-79 Standard.

The definitions to be loaded are written in high level for maximum transportability. As a consequence, some of the definitions are considerably slower than they should be. Appropriate comments have been inserted in the source screens to indicate definitions which should be re-coded in machine language.

There are two areas in which additional coding is required before your system will be 79-Standard. The Standard requires that a block of mass storage hold 1024 byte. BLOCK and BUFFER in the fig-FORTH implementations are usually smaller, and the definitions are implementation dependent. Note also that the 79-Standard requires that BLOCK 0 be accessible. Some fig-FORTH systems begin at BLOCK 1. You should modify your system appropriately. The other major problem area is in the word DOES> . There is no suitable way to write the 79-Standard version of DOES> in high level, since at least a portion is written in machine code, and other parts need constants which relate to machine code dependencies. A set of notes is included to aid you in converting your system over to the new form.

For certain word-oriented machines, such as the PDP-11, strict adherence to the 79-Standard requires that certain functions such as @ and ! be re-written to allow word manipulations on odd addresses. These functions are found at Screen 38, and need not be loaded for byte-oriented computers. There are some machines, such as the PACE, for which the overlay in Screen 38 is inappropriate.

If you do not have a fig-FORTH system, you may find the screens worth studying for suggestions on your conversion process.

BLOCK 30

```
( FIG->79-STANDARD.  COPYRIGHT 1981 BY R. L. SMITH )
DECIMAL
: >IN      IN ;
: ?DUP     -DUP ;
: NEGATE   MINUS ;
: EXIT     R> DROP ;
: CONVERT  (NUMBER) ;
: NOT      0= ;
: U/MOD    U/ ;
: DNEGATE  DMINUS ;
: 2*       DUP + ;  ( NOTE: NOT A 79-STANDARD WORD )
: 2/       ( NOTE: NOT A 79-STANDARD WORD. )
           ( SHOULD BE WRITTEN IN CODE )
           S->D 15 0 DO OVER OVER D+ LOOP SWAP DROP ;
: R@       R> R SWAP >R ;  ( OR CHANGE HEADER OF R )
;S  COPYRIGHT (C) 1981 BY ROBERT L. SMITH, PALO ALTO, CA
```

BLOCK 31

```
( FIG->79-STANDARD.  COPYRIGHT 1981 BY R. L. SMITH )
: D-       DMINUS D+ ;
: DEPTH    ( CHECK OUT CAREFULLY )  SP@ S0 @ SWAP - 2/ ;
: SAVE-BUFFERS   FLUSH ;
: U.       0 D. ;
: 0>       DUP IF  0<  0= THEN ;
: 1-       1 - ;
: 2-       2 - ;
: D<       ( EASIER IN CODE )
           ROT OVER OVER =
           IF  ROT ROT D- 0< SWAP DROP
           ELSE  SWAP < SWAP DROP SWAP DROP
           THEN  ;
: U<       0 SWAP 0 D< ;

;S  COPYRIGHT (C) 1981 BY ROBERT L. SMITH, PALO ALTO, CA
```

BLOCK 32

```
( FIG->79-STANDARD.  COPYRIGHT 1981 BY R. L. SMITH )
: PICK     DUP 0>
           IF 2* SP@ + @
           ELSE ."  PICK ARGUMENT < 1 " CR DROP
           THEN ;
: PLUNK    2* SP@ + ! ;
: SPREAD   OVER SWAP 2* SP@ 2+ DUP 2+ SWAP ROT CMOVE ;
: SLIP     2* SP@ +
           BEGIN  DUP 2 - @ OVER !  2 - SP@ OVER > 0= UNTIL
           DROP DROP ;
: ROLL     DUP 1 <
           IF  ."  ROLL ARGUMENT < 1 " CR DROP
           ELSE 1+ DUP PICK SWAP SLIP THEN ;
: FILL     OVER 0>
           IF  FILL ELSE DROP DROP DROP THEN ;
;S  COPYRIGHT (C) 1981 BY ROBERT L. SMITH, PALO ALTO, CA
```

BLOCK 33

```
( FIG->79-STANDARD.  COPYRIGHT 1981 BY R. L. SMITH )
FORTH DEFINITIONS
: -FIND1        @ @ BL WORD DUP
                IF HERE SWAP (FIND) DUP 0= ELSE 1 THEN
                IF DROP HERE [ ' FORTH 4 + ] LITERAL
                     @ (FIND) THEN ;
: -FIND-        CONTEXT -FIND1 R> DROP ;
( NOW PATCH -FIND )   ' -FIND- CFA ' -FIND !
: FIND          -FIND IF DROP ELSE 0 THEN ;
: VOCABULARY    <BUILDS  41089 ( A081 HEX ) , 0 ,
                HERE VOC-LINK @ , VOC-LINK !
                DOES>  2+ CONTEXT ! ;
: CREATE        0 VARIABLE -2 ALLOT ;
: VARIABLE      -32768 VARIABLE ;
: (             -1 >IN +!  [COMPILE] (   ; IMMEDIATE
;S  COPYRIGHT (C) 1981 BY ROBERT L. SMITH, PALO ALTO, CA
```

BLOCK 34

```
( FIG->79-STANDARD.  COPYRIGHT 1981 BY R. L. SMITH )
: FORGET CURRENT -FIND1 0=
         IF   ." NOT IN CURRENT VOCABULARY " CR
         ELSE DROP NFA DUP FENCE @ U< 21 ?ERROR
              >R R@ CONTEXT @ U< IF [COMPILE] FORTH THEN
              R@ CURRENT @ U<
              IF [COMPILE] FORTH DEFINITIONS THEN
              VOC-LINK @
              BEGIN  R@ OVER U<  WHILE @ DUP VOC-LINK ! REPEAT
              BEGIN   DUP 4 -
                   BEGIN  PFA LFA @ DUP R@ U< END
                   OVER 2- ! @ ?DUP 0=
              END R> DP !
         THEN ;
VARIABLE POCKET 258 ALLOT  ( TO HOLD STRING FROM WORD )
;S  COPYRIGHT (C) 1981 BY ROBERT L. SMITH, PALO ALTO, CA
```

BLOCK 35

```
( FIG->79-STANDARD.  COPYRIGHT 1981 BY R. L. SMITH )
: SCAN>   ( -- ADDR )
        BLK @
        IF  BLK @  BLOCK
        ELSE  TIB @
        THEN  >IN @ + ;
: WORD    (  CHAR  --  ADDR )
        DUP 0= IF DROP 1 ."  BAD DELIMITER " CR THEN
        SCAN>  SWAP DUP >R ENCLOSE DUP >IN +!
        OVER OVER >  IF   SWAP DROP DUP  THEN
        OVER OVER =   IF   R> DROP 0 >R  THEN
        DROP OVER - DUP 255 >
        IF  ."  CHARACTER STRING TOO LONG " CR 255 THEN
        >R R@ POCKET C!  + POCKET 1+ R@ CMOVE
        POCKET R> + 1+ R> SWAP C! POCKET ;
;S  COPYRIGHT (C) 1981 BY ROBERT L. SMITH, PALO ALTO, CA
```

BLOCK 36

```
( FIG->79-STANDARD.  COPYRIGHT 1981 BY R. L. SMITH )
: ."       SCAN> C@ 34 = NOT
           IF    [COMPILE] ."  ELSE  1 >IN +! THEN ;
           IMMEDIATE
( THE ABOVE DEFINITION SHOULD HANDLE THE NULL STRING CASE )
( THE FOLLOWING 3 DEFINITIONS MAY NOT BE NEEDED )
: (DO)     R ROT >R SWAP >R >R ;
: DO       COMPILE (DO) ; IMMEDIATE
: I        R> R SWAP >R ;
( THE NEXT THREE DEFINITIONS SHOULD BE WRITTEN IN CODE )
( FOR SPEED AND SAVING BYTES )
: J        R> R> R> R> R SWAP >R SWAP >R SWAP >R SWAP >R ;
: (LOOP)   R> R> 1+ DUP R <
           IF R> R SWAP >R SWAP >R >R DROP
           ELSE  DROP R> DROP R> DROP >R THEN ;
;S  COPYRIGHT (C) 1981 BY ROBERT L. SMITH, PALO ALTO, CA
```

BLOCK 37

```
( FIG->79-STANDARD.  COPYRIGHT 1981 BY R. L. SMITH )
( THE NEXT DEFINITION SHOULD BE WRITTEN IN CODE )
: (+LOOP)  R> SWAP DUP S->D R> S->D D+ ROT 0<
           IF   OVER OVER R S->D D<
              IF    DROP DROP R> DROP >R DROP R>
              ELSE  DROP R> R SWAP >R SWAP >R >R DROP THEN
           ELSE OVER OVER R S->D D<
              IF    DROP R> R SWAP >R SWAP >R >R DROP
              ELSE  DROP DROP R> DROP R> DROP >R THEN
           THEN ;
: LOOP     COMPILE (LOOP) ; IMMEDIATE
: +LOOP    COMPILE (+LOOP) ; IMMEDIATE
: :        [COMPILE] :  ;  ( NOTE: NOT IMMEDIATE )

;S  COPYRIGHT (C) 1981 BY ROBERT L. SMITH, PALO ALTO, CA
```

BLOCK 38

```
( FIG->79-STANDARD.  COPYRIGHT 1981 BY R. L. SMITH )
( THE FOLLOWING DEFINITIONS ARE FOR WORD-ORIENTED
MACHINES LIKE THE PDP-11. IF USED, RE-WRITE IN CODE )
VARIABLE %1TEMP 0 ,
: OLD@   @ ;
: OLD!   ! ;
: @      DUP C@ %1TEMP C!  1+ C@ %1TEMP 1+ C! %1TEMP @ ;
: ?      @ . ;
: !      SWAP %1TEMP !  DUP %1TEMP C@ SWAP C! %1TEMP 1+
         C@ SWAP 1+ C! ;
: +!     DUP @ ROT + SWAP ! ;
: 2@     4 0 DO   DUP I + C@ %1TEMP I + C! LOOP
         DROP %1TEMP OLD@ %1TEMP 2+ OLD@ ;
: 2!     SWAP %1TEMP OLD!  SWAP %1TEMP 2+ OLD!
         4 0 DO   %1TEMP I + C@ OVER I + C! LOOP  DROP ;
;S  COPYRIGHT (C) 1981 BY ROBERT L. SMITH, PALO ALTO, CA
```

BLOCK 39

```
( FIG->79-STANDARD.  COPYRIGHT 1981 BY R. L. SMITH )
( ADD NEXT 2 DEFINITIONS IF SCREEN #38 WAS NOT LOADED )
: 2@          DUP 2+ @ SWAP @ ;
: 2!          OVER OVER !  2+ SWAP DROP ! ;
: 2CONSTANT CREATE  ,  ,  DOES> 2@ ;
: 2DROP       DROP DROP ;
: 2DUP        OVER OVER ;
: 2OVER       4 PICK 4 PICK ;
: 2ROT        6 ROLL 6 ROLL ;
: 2SWAP       4 ROLL 4 ROLL ;
: 2VARIABLE VARIABLE Ø , ;
: DØ=         OR Ø= ;
: D=          D- DØ= ;
: DMAX        2OVER 2OVER D< IF  2SWAP THEN 2DROP ;
: DMIN        2OVER 2OVER D< IF 2DROP ELSE 2SWAP 2DROP THEN ;
;S  COPYRIGHT (C) 1981 BY ROBERT L. SMITH, PALO ALTO, CA
```

BLOCK 40Ø

```
( FIG->79-STANDARD.  COPYRIGHT 1981 BY R. L. SMITH )
: DU<         3 PICK OVER = IF DROP SWAP DROP
              ELSE SWAP DROP ROT DROP THEN U< ;
: MOVE
              DUP 1 <
              IF  DROP DROP DROP
              ELSE 2* CMOVE
              THEN ;
: CMOVE
              DUP 1 <
              IF DROP DROP DROP
              ELSE CMOVE
              THEN ;

;S  COPYRIGHT (C) 1981 BY ROBERT L. SMITH, PALO ALTO, CA
```

BLOCK 41

```
( FIG->79-STANDARD.  COPYRIGHT 1981 BY R. L. SMITH )
: FATHER      CREATE 1979 , DOES> ;
FATHER        SON
( THE NEW DOES> AND BLOCK AND BUFFER MUST BE INSTALLED )
: 79-STANDARD
              ' SON @ 1979 = Ø=
              B/BUF 1Ø24 - OR IF  ." NOT 79-STANDARD SYSTEM "
              CR THEN ;
( SET UP FENCE TO PROTECT THE NEW SYSTEM )
HERE FENCE !



;S  COPYRIGHT (C) 1981 BY ROBERT L. SMITH, PALO ALTO, CA
```

Implementing the 79-Standard DOES>

In fig-FORTH, DOES> is a word used in conjunction with <BUILDS to define new classes of defining words. In 79-Standard FORTH, DOES> performs a similiar role in conjunction with the word CREATE . There are some differences between the two constructs. In the fig-FORTH model, <BUILDS allocates the first word in the parameter field of a new word. In the 79-Standard, CREATE does not allocate that position. A result of the 79-Standard implementation is that the parameter field address of newly created words points to the first parameter, as we would tend to assume. Such is not the case with the fig-FORTH model. An additional difference is that the 79-Standard DOES> is an immediate word, whereas the fig-FORTH version is not.

The new version of DOES> is closely allied with ;CODE . Indeed, in creating a new defining word the action of the new DOES> is to emplace (;CODE) after the high level compile time sequence, followed by a very short code fragment which switches control back to the high level interpreter.

Because the action of DOES> involves some machine language code and a detailed knowledge of particular implementations, it is not possible to make a general overlay for the new DOES> . If you wish to convert your present DOES> to the 79-Standard DOES> , it will be necessary to understand exactly how your present system handles the following entities:

IP	the FORTH Interpretive Pointer
RP	the FORTH Return Stack Pointer
SP	the FORTH Parameter Stack Pointer
HP	the machine Hardware Stack Pointer
W	the FORTH Intermediate Address Pointer

These pointers may reside in hardware registers or as simple memory locations. There is a no fixed rule on the precise nature of the pointers between various machines. IP, for example, may point to the current word being interpreted, or it may point to the next word to be interpreted. On some machines (such as the 8080), W is only partly incremented past the current word, and is fully incremented only when it is required. We will use a pseudo-machine for part of the following exposition. For that machine we will assume that all the pointers, except the hardware stack pointer, reside in memory.

Suppose we make a defining word VECTOR as follows:

```
: VECTOR  CREATE 2* ALLOT DOES> SWAP 2* + ;
```

Assume that VECTOR is used to make a vector named V1 by the construction:

```
5  VECTOR  V1
```

Further assume that V1 is used within another definition:

```
: TEST     ...   I  V1  !   ...   ;
```

The dictionary structure of these three words is shown in Figure 1.

In the dictionary entry for VECTOR we see an unusual set of entries where DOES> might otherwise be expected to appear. Note again that DOES> is an immediate word. Its action takes place when compiling the word VECTOR . It emplaces the high level word (;CODE) and then a small piece of code, namely JSR DODOES . The JSR means a machine language "Jump to Subroutine" instruction, and DODOES is that address to which control is to be transfered. When the JSR is executed, we assume that the machine address immediately following the complete instruction is pushed onto the hardware stack. We have given the address of the JSR the name DOVECTOR , which indicates the run-time action of words created by VECTOR . Note that the code field of V1 points to DOVECTOR , which is machine code.

Consider that the word I within the word TEST is being executed. We assume that IP has been post-incremented, and therefore points to the next word, V1 , in TEST , and has the value indicated by IP0 in Figure 1. When I is finished, control will be transfered to NEXT . Execution of NEXT will place the contents of what IP is pointing to, namely V1, into W . IP will be incremented by 2 bytes to the value IP1. W will be incremented by 2, to have the value W1. Finally control is passed to the location named at V1, namely DOVECTOR .

The following is pseudo-machine code for DODOES :

```
DODOES: DEC     RP          ; push IP1 on Return Stack.
        DEC     RP
        MOVE    IP,(IP)
        MOVE    (HS),IP     ; Top of hardware stack to IP
        INC     HS          ; and adjust hardware stack.
        INC     HS
        DEC     SP          ; Push W1 on Parameter Stack.
        DEC     SP
        MOVE    W,(SP)
NEXT:   MOVE    (IP),W      ; (IP2) -> W
        INC     IP          ; Post-increment IP to IP3.
        INC     IP
        MOVE    (W),TEMP
        INC     W           ; Post-increment W
        INC     W
        JUMP    (TEMP)      ; Execute the next word.
```

At DODOES, the current value of IP , namely IP1, is pushed onto the Return Stack. The top of the hardware stack is placed in IP . (Note that for some machines this value may have to be appropriately incremented or decremented first). The value in

W (pointing to the parameter field of V1) is pushed onto the parameter stack. (Again, for certain machines this value may need some adjustment). Finally, NEXT is executed, which will begin high level interpretation at IP2. When the word EXIT is executed at the end of VECTOR , high level control will revert back to the point IP1 in word TEST .

The only additional information required is the definition of DOES> . This will vary somewhat with the implementation. The general format of the definition is:

```
: DOES>
    ?CSP  COMPILE  (;CODE)
    nnnn  ( value of JSR code )  ,
    mmmm  ( address of DODOES )  ,  ;  IMMEDIATE
```

If the JSR is a single byte, you may either change the comma to C, or insert a NOP instruction. I have assumed the simple kind of JSR instruction where the actual address follows the JSR. In some machines, an offset may be required. If DODOES is created using a CODE word, you may have to compensate to get the actual starting address. If your machine does not have a JSR or its equivalent, you may have to be somewhat more creative.

If you have not already redefined CREATE to be compatible with the 79-Standard, one simple definition is:

```
: CREATE   VARIABLE  -2  ALLOT  ;
```

This assumes that the 79-Standard version of VARIABLE is active. Otherwise the definition would be:

```
: CREATE   0 VARIABLE  -2  ALLOT  ;
```

The following are notes and suggestions for various machines known to have implementations in fig-FORTH. Some of the code has not been tested by the author, and can therefore be used only as a rough guide. In the PDP-11 (fig-FORTH version 1.2), IP is a hardware register (#2), and it is post-incremented at NEXT. SP is hardware register #3. RP is the same as the hardware Stack Pointer (#6). W is hardware register #1, and it is post-incremented in NEXT (in version 1.2). NEXT is a macro which is expanded in-line into the following 2 word sequence:

```
MOV     (IP)+,W
JMP     @(W)+
```

In the PDP-11, a simple trick speeds up DODOES. The JSR that is inserted after the (;CODE) specifies the use of register #2 (IP). The destination address is DOVAR, the code routine used for putting the address of a variable on the parameter stack. DODOES itself is completely eliminated! When

the JSR is executed, the value of IP will be pushed onto the return stack (the same as the hardware stack), and the nominal return address will be placed in the IP register. The function of DOVAR is to push the value in W on the parameter stack, then execute NEXT, which will start high level interpretation where desired.

The form of DOES> for the PDP-11 is currently in a pre-compiled form which has the general effect of the following definition:

```
OCTAL
: DOES>   ?CSP  COMPILE  (;CODE)
     4237  ,   LIT  DOVAR  ,  ;  IMMEDIATE
```

The JSR instruction in normal MACRO assembly language would appear as:

```
JSR    IP,@#DOVAR
```

DOVAR is defined in MACRO as follows:

```
DOVAR:  MOV  W,-(S)
        NEXT
```

where NEXT is expanded as discussed above.

In the 6800, IP is a pointer kept in memory. It is different from most of the other fig-FORTH implementations in that IP is pre-incremented in NEXT, and therefore points to the word currently undergoing high-level interpretation. Similiarly, W is not post-incremented and therefore points to the Code Field Address (CFA) of a machine language or code word. RP is a memory location which points to the first free word of the return stack (the reason relates to indexing which can only be positive). The machine stack pointer is the parameter stack pointer SP. However, SP actually points to the first free byte of the stack. (There is a compensating modification when the stack pointer is moved to the index register with a TSX instruction). The following code appears to be appropriate for the 6800 DODOES:

```
DODOES: LDX   RP
        DEX
        DEX
        STX   RP
        LDAA  IP+1
        STAA  3,X
        PULA
        STAA  IP
        PULA
        STAA  IP+1
        LDX   W
        INX
        INX
        STX   W
        LDAA  W+1
        PSHA
        LDAA  W
        PSHA
        LDX   IP
        LDX   0,X
        STX   W
        LDX   0,X
        JMP   0,X
```

The form of DOES> is something like:

```
HEX
: DOES>   ?CSP  COMPILE  (;CODE)
          0BD  C,
          nnnn ( address of DODOES)  ,  ;
          IMMEDIATE
```

The following implementation of the new form of DOES> on the 6502 was supplied by Bill Ragsdale in the format used in the fig-FORTH implementation notes. Note that IP is post-incremented in NEXT, that W is not post-incremented (it points to the CFA of the current word being executed), and that the hardware stack is used for the Return Stack. Those unfamiliar with the 6502 should note that the JSR instruction behaves in an unexpected way: the address pushed onto the hardware stack is one location less than the actual address to which control will ultimately be returned. This is compensated in the Return instruction to return control to the correct location.

```
: (;CODE)     R> LATEST PFA CFA ! ;

HEX
: DOES>   COMPILE (;CODE)  20 C,
          COMPILE [  HERE 4 + ,  ]  ;     IMMEDIATE
ASSEMBLER
     DEX,    DEX,    CLC,
     W   LDA,
     2 # ADC,
     BOT STA,  ( W+2 to parameter stack )
     TYA,
     W 1+ ADC,
     BOT 1+ STA,
     SEC,   PLA,
     1 # SBC,
     W   STA,
     PLA,
     0 # SBC,
     W 1+ STA,
     ' QUIT CFA @ JMP,  ( jumps to DOCOLON )
```

The 8080 is a very common microprocessor for implementing FORTH. In the fig-FORTH model, IP is generally held in the B and C hardware registers. IP is post-incremented in NEXT. At the end of execution of NEXT, W is held in the D and E registers. It is incremented 1 byte past the CFA of the current word being executed. (Since it is rarely used, it is not worth the machine time to further increment it until needed). The following code may be loaded without an assembler (fig-FORTH version 1.1 is assumed) :

```
HEX
28 +ORIGIN CONSTANT RPP   ( RP POINTER )
45 +ORIGIN CONSTANT NEXT  ( LOCATION OF NEXT )
: DOES>   ?CSP  COMPILE (;CODE)  CD ( CALL ) C,
          COMPILE [ HERE 4 + , ] ; IMMEDIATE
( DODOES STARTS HERE )
13   C,             ( INX D     )
2A   C,   RPP ,     ( LHLD  RPP )
2B   C,             ( DCX   H   )
70   C,             ( MOV   M,B )
2B   C,             ( DCX   H   )
71   C,             ( MOV   M,C )
22   C,   RPP ,     ( SHLD  RPP )
C1   C,             ( POP   B   )
D5   C,             ( PUSH  D   )
C3   C,   NEXT ,    ( JMP   NEXT )
```

Response to Error Conditions

The 79-Standard requires that a Standard System be provided with a tabulation of the action taken for all specified error conditions. Since the information provided in this document covers a variety of implementations and machines, the actions specified below will not be appropriate for all cases. Use the following list as a guide-line and check list for error conditions. The author would appreciate feedback from users on conditions that he has missed.

The following is a list of words which may be executed only during compilation. Their use outside of compilation is an error condition.

+LOOP	ELSE	R>
;	EXIT	R@
;CODE	I	REPEAT
>R	IF	THEN
BEGIN	J	UNTIL
COMPILE	LEAVE	WHILE
DOES>	LOOP	[COMPILE]
		SIGN

The word SIGN is included in the list because it is so listed in the 79-Standard, but it appears that its designation as "compile-only" is a probable typographical error. Only the following words from the above list are likely to be flagged as an error when used outside of a colon definition:

+LOOP	DOES>	LOOP
;	ELSE	UNTIL
BEGIN	IF	

When used in a fig-FORTH system with the overlays and added code suggested in this document, the normal error message for using these words outside of a colon definition is "COMPILATION ONLY, USE ONLY IN DEFINITION".

There are error conditions associated with specific words. A list of these words, associated errors, and system response, if any, appears below.

Error if not used between <# and #>. No system check.

#S Error if not used between <# and #>. No system check.

' Error if following name not found in dictionary. Usual system response is to print the offending name and a question mark "?".

(Error if input stream is exhausted befor terminating right parenthesis is found. No error message given.

`*` Error if product greater than 15 bits plus sign. Usual system response is truncated low order 16 bits of the product. Normally no error message is given.

`*/` Error if division by Ø or quotient overflows. System response is undefined, or depends on the particular installation.

`*/MOD` Error if division by Ø. System response is either undefined or system dependent.

`+` Error if sum overflows. Usual system response returns the 16 bit truncated unnormalized sum. Normally no error message is given.

`+!` Error if sum overflows. Normally no error message is given.

`+LOOP` Error if not matched by a preceeding DO . No system error in current version. Possible error if preceeded by an argument with value of Ø. Current system treats Ø as a positive number for the purposes of determining looping condition.

`-` Error if the difference overflows. Usual system response returns a 16 bit value similiar to that of the case of overflow from addition. Usually no error message is given.

`-TRAILING` Error if character count is negative. No error message is given. Character count may be reduced by one.

`/` Error if division by Ø. System response is highly system dependent.

`/MOD` Error if division by Ø. Sytem response is highly system dependent.

`1+` Error if sum overflows. See +.

`1-` Error if difference overflows. See -.

`2+` Error if sum overflows. See +.

`2-` Error if difference overflows. See -.

`79-STANDARD` Error if 79-Standard system not available. Current version checks block size. If not 1Ø24 bytes long, prints " NOT 79-STANDARD SYSTEM ".

: Although not specified by the 79-Standard, if the new name has been previously defined in either the CONTEXT vocabulary or in FORTH, the warning message "ISN'T UNIQUE" is printed on most systems. An error also occurs if a word to be compiled is not found in the dictionary and is not convertable into a number. The usual system response is to print the offending word with a following "?".

; Error if input stream from mass storage is terminated while compiling and before a ";" is encountered. When the end of the input stream is encountered while compiling, the error message "EXECUTION ONLY" will be given on most systems.

>IN Error if the value at >IN is outside of the range 0 to 1023. No system response.

>R Error if not balanced inside of a colon definition with a matching R> . No predictable system response.

ABS Error when the argument is the most negative 16 bit number. Probable system response is to return that argument unchanged with no error message.

BASE Error if the value at BASE is outside of range 2 to 70. Most systems do not check this range.

BEGIN Error if not matched by UNTIL or WHILE ... REPEAT within the current definition. Error may be detected at end of definition.

BLOCK Error if specified block is unavailable. System response is system dependent.

BUFFER Error if previous block cannot be written to mass storage. Response is system dependent.

CMOVE Error if count is less than 1. Usual system response is to treat the count as an unsigned number.

CONVERT Probable error if at least part of the conversion occurs outside of the specified string. Most systems do not check for this.

D+ Error occurs if sum overflows. Result is system dependent. Normally no error message is given.

D- Error if difference causes an overflow. Result is sytem dependent. Normally no error message is given.

DABS Error if argument is the most negative double precision number. Result is system dependent. Normally no error message is given.

DNEGATE Error if the argument is the most negative double precision value. Result is system dependent. Normally no error message is given.

DO Error if not matched later in the same colon definition by LOOP or +LOOP . Error may be detected at the end of the definition.

ELSE Error if not preceeded by IF and followed by THEN within the current colon definition. An error message is normally given in a fig-FORTH system.

EXIT Error if used within a DO ... LOOP , a DO ... +LOOP , or within a >R ... R> pair. Most systems do not check for these possibilities.

FORGET Error if the following name is not found in the CURRENT or FORTH vocabulary. In this system, the error message "NOT IN CURRENT VOCABULARY" is given if the name is not found. Error if the FORGETting procedure would destroy part of the basic system or the 79-STANDARD vocabulary. The usual error message for this case is "IN PROTECTED DICTIONARY".

HOLD Technically an error if used outside of <# ... #> . No system response for this error.

I Error if used outside of DO ... LOOP or DO ... +LOOP . No sytem response.

IF Error if not followed by THEN in the current colon definition. Error is usually detected at end of the definition.

J Error if used outside of DO ... LOOP or DO ... +LOOP . No system response.

LEAVE Error if used outside of DO ... LOOP or DO ... +LOOP . Usually no check is made for this error, and the results are unprecdictable.

LIST Note that some systems may not be able to list screen Ø. Note: Since fig-FORTH systems usually use screens 4 and 5 to hold error messages, this is a potential source of strange error messages from disks or programs which use these screens for other purposes.

LOAD Error if specified block cannot be loaded from mass storage. Result is system dependent.

LOOP Error if not preceeded by DO within the current colon definition. This error not checked in the version defined in this document.

MOD Error if division by Ø. Result is system dependent.

MOVE Error if the specified addresses are not on cell boundaries. Most systems do not check this.

NEGATE Error if argument is the most negative single precision value. Result is system dependent, but the usual result is to return the same value.

PICK Error if argument is less than 1. System response is message "PICK ARGUMENT < 1".

R> Error if not matched by >R in current colon definition. This condition is rarely checked. The results are usually fatal.

R@ Error if used outside of >R ... R> pair. Not checked by most systems.

REPEAT Error if not preceeded by BEGIN and WHILE within the current colon definition. This condition is checked in most fig-FORTH systems and results in the error message "CONDITIONALS NOT PAIRED".

ROLL Error condition if the argument is less than 1. System response is the message " ROLL ARGUMENT < 1".

SAVE-BUFFERS Error condition results if mass storage writing is not completed. Response is system dependent.

SIGN Error if used outside of <# and #> pair. Most systems do not check for this.

THEN Error if not preceeded by IF in the current colon definition. This condition is checked in most fig-FORTH systems and results in the message "CONDITIONALS NOT PAIRED".

U/MOD Error if division by Ø. Result is system dependent.

UNTIL Error if not preceeded by BEGIN in current colon definition. In most fig-FORTH systems the error message "CONDITIONALS NOT PAIRED" is given.

WHILE Error if not preceeded by BEGIN in the current colon definition. Condition generally not checked. Error if not followed by REPEAT in the same definition. This error may be checked at the end of the definition.

WORD Error if character specified is null. The current version prints an error message "BAD DELIMITER" and replaces the value of Ø representing the null with the value of 1. An error condition also arises if the string length exceeds 255. In the current version, an error message "CHARACTER STRING TOO LONG" is printed.

Other errors.

There are a number of other errors which may occur. The fig-FORTH system may detect some of these. A common error is an insufficient number of parameters on the stack. When this results in a stack empty or stack underflow condition, the message "EMPTY STACK" is usually printed. Some systems check to see that sufficient room remains when space is allocated in the dictionary. If not, the usual error message is "DICTIONARY FULL". If the stack is found to be in an overflow condition, for those implementations that check for it the error message "FULL STACK" is usually printed.

Suggested Usage Rules

In addition to the rules of usage suggested in the 79-Standard, the following rules are suggested to minimize transportability problems:

(1) Do not use empty strings as in ( ) or ." " . There are only a few implementations of the 79-Standard which handle null strings in strict accordance with the Standard. Note that this version is in compliance.

(2) If you are moving characters resulting from the use of WORD to newly created dictionary space, first transfer the characters to an intermediate buffer, then create the new space, and finally move the characters from the buffer to the desired final position.

(3) Do not use I or J as addresses within a DO-loop. The result may work on some systems, but not all. Since the 79-Standard uses pure signed numbers in DO-loop calculations, if a region of memory crosses the boundary between the lower half and the upper half of the 64 Kbyte memory space, use of I or J will lead to anomolous results.

(4) For double precision literal numbers, use only a trailing period "." to indicate such a number. Most systems will recognize the resulting number as double precision.

Corrections

See FORTH DIMENSIONS for possible corrections to this document.

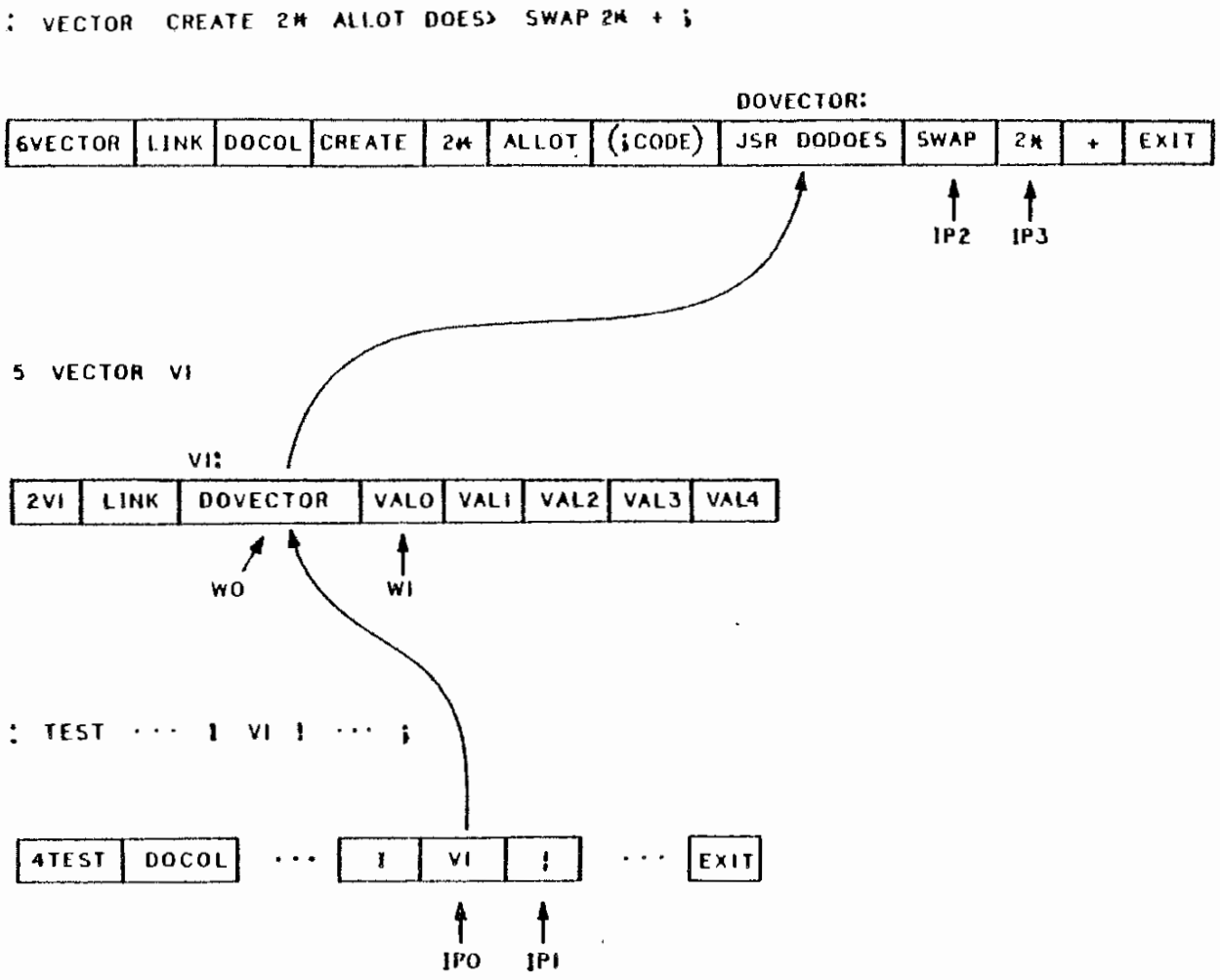

Figure 1. Example of use of 79-Standard DOES>