Manuel de référence pour Z79FORTH

version 1.2 - vendredi 4 octobre 2024

Auteur

- François LAAGEL
- Marc PETREMANN

Collaborateur

- ….

Contents

Tous les mots

Moi, François Laagel, atteste par la présente que, à ma connaissance, Z79Forth/A est un système ANS Forth répondant aux exigences minimales énoncées dans la norme ANSI-X3.215-1994.

La spécification standard est disponible à l'adresse http://lars.nocrew.org/dpans/

Rapport sur l'état d'avancement de la mise en œuvre :

- \- Non supporté
- E Résident EEPROM
- C Résident CompactFlash

Ensemble de mots de base

```
E     <           *less-than*             CORE
E     <#          *less-number-sign*      CORE
E     =           *equals*                CORE
E     >           *greater-than*          CORE
E     -           *minus*                 CORE
E     ,           *comma*                 CORE
E     ;           *semicolon*             CORE
E     :           *colon*                 CORE
E     !           *store*                 CORE
E     /           *slash*                 CORE
E     .           *dot*                   CORE
E     ."          *dot-quote*             CORE
E     '           *tick*                  CORE
E     (           *paren*                 CORE
E     [           *left-bracket*          CORE
E     [']         *bracket-tick*          CORE
E     ]           *right-bracket*         CORE
E     @           *fetch*                 CORE
E     *           *star*                  CORE
E     */          *star-slash*            CORE
E     #           *number-sign*           CORE
E     #>          *number-sign-greater*   CORE
E     +           *plus*                  CORE
E     +!          *plus-store*            CORE
E     0<          *zero-less*             CORE
E     0=          *zero-equals*           CORE
E     1-          *one-minus*             CORE
E     1+          *one-plus*              CORE
E     2!          *two-store*             CORE
E     2/          *two-slash*             CORE
```

```
E     2@          *two-fetch*             CORE
E     2*          *two-star*              CORE
E     2DROP       *two-drop*              CORE
E     2DUP        *two-dupe*              CORE
E     2OVER       *two-over*              CORE
E     2SWAP       *two-swap*              CORE
E     ABORT       -                       CORE
C     ABORT"      *abort-quote*           CORE
E     ABS         *abs*                   CORE
E     ACCEPT      -                       CORE
E     ALIGN       -                       CORE
E     ALIGNED     -                       CORE
E     ALLOT       -                       CORE
E     AND         -                       CORE
E     BASE        -                       CORE
E     BEGIN       -                       CORE
E     BL          *b-l*                   CORE
C     >BODY       *to-body*               CORE
E     C,          *c-comma*               CORE
E     C!          *c-store*               CORE
E     C@          *c-fetch*               CORE
E     CELL+       *cell-plus*             CORE
E     CELLS       -                       CORE
E     CHAR        *char*                  CORE
E     [CHAR]      *bracket-char*          CORE
E     CHAR+       *char-plus*             CORE
E     CHARS       *chars*                 CORE
E     CONSTANT    -                       CORE
E     COUNT       -                       CORE
E     CR          *c-r*                   CORE
E     CREATE      -                       CORE
E     DECIMAL     -                       CORE
E     DEPTH       -                       CORE
E     DO          -                       CORE
E     DOES>       *does*                  CORE
E     DROP        -                       CORE
E     DUP         *dupe*                  CORE
E     ?DUP        *question-dupe*         CORE
E     ELSE        -                       CORE
E     EMIT        -                       CORE
C     ENVIRONMENT? *environment-query*    CORE
E     EVALUATE    -                       CORE
E     EXECUTE     -                       CORE
E     EXIT        -                       CORE
E     FILL        -                       CORE
E     FIND        -                       CORE
E     FM/MOD      *f-m-slash-mod*         CORE
E     HERE        -                       CORE
```

E	HOLD	-	CORE
E	I	-	CORE
E	IF	-	CORE
E	IMMEDIATE	-	CORE
E	>IN	*to-in*	CORE
E	INVERT	-	CORE
E	J	-	CORE
E	KEY	-	CORE
E	LEAVE	-	CORE
E	LITERAL	-	CORE
E	LOOP	-	CORE
E	+LOOP	*plus-loop*	CORE
C	LSHIFT	*l-shift*	CORE
E	M*	*m-star*	CORE
E	MAX	-	CORE
E	MIN	-	CORE
E	MOD	-	CORE
E	/MOD	*slash-mod*	CORE
E	*/MOD	*star-slash-mod*	CORE
E	MOVE	-	CORE
E	NEGATE	-	CORE
E	>NUMBER	*to-number*	CORE
E	OR	-	CORE
E	OVER	-	CORE
E	POSTPONE	-	CORE
E	QUIT	-	CORE
E	>R	*to-r*	CORE
E	R>	*r-from*	CORE
E	R@	*r-fetch*	CORE
E	RECURSE	-	CORE
E	REPEAT	-	CORE
E	ROT	*rote*	CORE
C	RSHIFT	*r-shift*	CORE
E	#S	*number-sign-s*	CORE
E	S"	*s-quote*	CORE
E	S>D	*s-to-d*	CORE
E	SIGN	-	CORE
E	SM/REM	*s-m-slash-rem*	CORE
E	SOURCE	-	CORE
E	SPACE	-	CORE
E	SPACES	-	CORE
E	STATE	-	CORE
E	SWAP	-	CORE
E	THEN	-	CORE
E	TYPE	-	CORE
E	U<	*u-less-than*	CORE
E	U.	*u-dot*	CORE
E	UM*	*u-m-star*	CORE

E	UM/MOD	*u-m-slash-mod*	CORE
E	UNLOOP	-	CORE
E	UNTIL	-	CORE
E	VARIABLE	-	CORE
E	WHILE	-	CORE
E	WORD	-	CORE
E	XOR	*x-or*	CORE

Ensemble de mots d'extension de base

E	<>	*not-equals*	CORE_EXT
C	.(	*dot-paren*	CORE_EXT
E	\	*backslash*	CORE_EXT
E	0<>	*zero-not-equals*	CORE_EXT
E	0>	*zero-greater*	CORE_EXT
C	2>R	*two-to-r*	CORE_EXT
C	2R>	*two-r-from*	CORE_EXT
C	2R@	*two-r-fetch*	CORE_EXT
E	AGAIN	-	CORE_EXT
-	C"	*c-quote*	CORE_EXT
-	CASE	-	CORE_EXT
-	[COMPILE]	*bracket-compile*	CORE_EXT
E	COMPILE,	*compile-comma*	CORE_EXT
-	CONVERT	-	CORE_EXT
E	?DO	*question-do*	CORE_EXT
-	ENDCASE	*end-case*	CORE_EXT
-	ENDOF	*end-of*	CORE_EXT
C	ERASE	-	CORE_EXT
-	EXPECT	-	CORE_EXT
E	FALSE	-	CORE_EXT
E	HEX	-	CORE_EXT
E	MARKER	-	CORE_EXT
E	NIP	-	CORE_EXT
E	:NONAME	*colon-no-name*	CORE_EXT
-	OF	-	CORE_EXT
E	PAD	-	CORE_EXT
-	PARSE	-	CORE_EXT
E	PICK	-	CORE_EXT
-	QUERY	-	CORE_EXT
E	.R	*dot-r*	CORE_EXT
E	REFILL	-	CORE_EXT
-	RESTORE-INPUT	-	CORE_EXT
E	ROLL	-	CORE_EXT
-	SAVE-INPUT	-	CORE_EXT
-	SOURCE-ID	*source-i-d*	CORE_EXT
-	SPAN	-	CORE_EXT
-	TIB	*t-i-b*	CORE_EXT
-	#TIB	*number-t-i-b*	CORE_EXT

C	TO	-	CORE_EXT
E	TRUE	-	CORE_EXT
E	TUCK	-	CORE_EXT
E	U>	*u-greater-than*	CORE_EXT
C	UNUSED	-	CORE_EXT
E	U.R	*u-dot-r*	CORE_EXT
C	VALUE	-	CORE_EXT
C	WITHIN	-	CORE_EXT

Ensemble de mots de blocs

E	BLK	*b-l-k*	BLOCK
E	BLOCK	-	BLOCK
E	BUFFER	-	BLOCK
E	EVALUATE	-	BLOCK
E	FLUSH	-	BLOCK
E	LOAD	-	BLOCK
E	UPDATE	-	BLOCK
E	SAVE-BUFFERS	-	BLOCK

Ensemble de mots d'extension de bloc

E	EMPTY-BUFFERS	-	BLOCK_EXT
E	LIST	-	BLOCK_EXT
E	REFILL	-	BLOCK_EXT
E	SCR	*s-c-r*	BLOCK_EXT
E	THRU	-	BLOCK_EXT

Ensemble de mots à double précision

-	2CONSTANT	*two-constant*	DOUBLE
-	2LITERAL	*two-literal*	DOUBLE
-	2VARIABLE	*two-variable*	DOUBLE
E	D<	*d-less-than*	DOUBLE
-	D=	*d-equals*	DOUBLE
E	D-	*d-minus*	DOUBLE
-	D.	*d-dot*	DOUBLE
E	D+	*d-plus*	DOUBLE
-	D0<	*d-zero-less*	DOUBLE
E	D0=	*d-zero-equals*	DOUBLE
-	D2/	*d-two-slash*	DOUBLE
-	D2*	*d-two-star*	DOUBLE
-	DABS	*d-abs*	DOUBLE
-	DMAX	*d-max*	DOUBLE
-	DMIN	*d-min*	DOUBLE
E	DNEGATE	*d-negate*	DOUBLE
-	D.R	*d-dot-r*	DOUBLE
-	D>S	*d-to-s*	DOUBLE
-	M*/	*m-star-slash*	DOUBLE

```
-     M+          *m-plus*                DOUBLE
```

Mots utiles

```
-     AT-XY       *at-x-y*                FACILITY
E     KEY?        *key-question*          FACILITY
E     PAGE        -                       FACILITY
```

Ensemble de mots pour l'extension des installations

```
-     EKEY        *e-key*                 FACILITY_EXT
-     EKEY?       *e-key-question*        FACILITY_EXT
-     EKEY>CHAR   *e-key-to-char*         FACILITY_EXT
-     EMIT?       *emit-question*         FACILITY_EXT
E     MS          -                       FACILITY_EXT
-     TIME&DATE   *time-and-date*         FACILITY_EXT
```

Ensemble de mots pour les outils de programmation

```
E     .S          *dot-s*                 TOOLS
E     ?           *question*              TOOLS
C     DUMP        -                       TOOLS
C     SEE         -                       TOOLS
E     WORDS       -                       TOOLS
```

Ensemble de mots d'extension des outils de programmation

```
-     ;CODE       *semicolon-code*        TOOLS_EXT
-     [ELSE]      *bracket-else*          TOOLS_EXT
-     [IF]        *bracket-if*            TOOLS_EXT
-     [THEN]      *bracket-then*          TOOLS_EXT
E     AHEAD       -                       TOOLS_EXT
-     ASSEMBLER   -                       TOOLS_EXT
E     BYE         -                       TOOLS_EXT
-     CODE        -                       TOOLS_EXT
-     CS-PICK     *c-s-pick*              TOOLS_EXT
-     CS-ROLL     *c-s-roll*              TOOLS_EXT
-     EDITOR      -                       TOOLS_EXT
-     FORGET      -                       TOOLS_EXT
E     STATE       -                       TOOLS_EXT
```

Note: BYE will reset the system. Dirty (UPDATEd) buffers, if any, will not be flushed to mass storage.

Ensemble de mots de chaîne

```
C     -TRAILING    *dash-trailing*          STRING
C     /STRING      *slash-string*           STRING
E     BLANK        -                        STRING
E     CMOVE        *c-move*                 STRING
E     CMOVE>       *c-move-up*              STRING
C     COMPARE      -                        STRING
-     SLITERAL     -                        STRING
```

Mots de Forth2012

Voir https://forth-standard.org pour plus d'information.

```
ACTION-OF DEFER DEFER! DEFER@ IS
```

Mots 79-STANDARD

Voir *SW/reference/FORTH-79.TXT* pour plus d'information.

```
I' INDEX INTERPRET K LAST LINE NOT SHIFT
```

forth

! n addr --

Stocke une valeur entière n à l'adresse addr.

```
variable TEMPERATURE
32 TEMPERATURE !
```

ud1 -- ud2

EN COMPILATION SEULEMENT

Convertit un chiffre vers la chaine numérique formatée.

Ce mot doit être utilisé entre <# et #>.

La conversion en digit s'effectue par rapport à la base numérique courante.

#> ud1 -- c-addr u

COMPILATION SEULEMENT

Empile l'adresse et taille d'une chaîne numérique formatée.

#s ud1 -- ud2

COMPILATION SEULEMENT

Convertit le reste des chiffres vers une sortie numérique formatée.

' -- cfa

Analyse le mot et le recherche dans le dictionnaire. S'il est trouvé, empile le cfa de ce mot.

```
' words     \ leave cfa of words on stack
execute     \ execute words
```

(-TEXT) c_addr1 u c_addr2 -- n

Compare la chaîne spécifiée par c-addr1 u1 à la chaîne spécifiée par c-addr2 u2. Les chaînes sont comparées, en commençant aux adresses données, caractère par caractère, jusqu'à la longueur de la chaîne la plus courte ou jusqu'à ce qu'une différence soit trouvée.

- Si les deux chaînes sont identiques, n est égal à zéro.

- Si les deux chaînes sont identiques jusqu'à la longueur de la chaîne la plus courte, n est égal à moins un (-1) si u1 est inférieur à u2 et à un (1) sinon.

- Si les deux chaînes ne sont pas identiques jusqu'à la longueur de la chaîne la plus courte, n est égal à moins un (-1) si le premier caractère non correspondant dans la chaîne spécifiée par c-addr1 u1 a une valeur numérique inférieure à celle du caractère correspondant dans la chaîne spécifiée par c-addr2 u2 et à un (1) sinon.

(SGN) n -- -1|0|1

Empile un drapeau indiquant si le nombre n est positif, nul ou négatif.

```
4 (SGN) .  \ display: 1
-3 (SGN) .  \ display:-1
 0 (SGN) .  \ display:0
```

* n1 n2 -- n1*n2

Multiplication entière de deux nombres.

```
6 3  *   \ push 18 operation 6*3
7 3  *   \ push 21 operation 7*3
-7 3 *   \ push -21
7 -3 *   \ push -21
-7 -3 *  \ push 21
```

*/ n1 n2 n3 -- n4=(n1*n2)/n3

Multiplie n1 par n2 produisant le résultat intermédiaire à double précision d. Divise d par n3 en donnant le quotient entier n4.

```
5000 1000 4000 */ .    \ display    1250
```

*/MOD n1 n2 n3 -- rem quot

Multiplie n1 et N2 puis divise par n3 avec résultat intermédiaire 32 bits.

```
50000 10 4001 */MOD .    \ display    124 3876
```

+ n1 n2 -- n3=n1+n2

Addition de deux entiers signés simple précision.

```
6 3  + .    \ display 9
-6 3 + .    \ display -3
```

+! n addr --

Incrémente le contenu d'une variable de la valeur n

```
variable counter
15 counter +!
```

```
counter @ .  \ display 15
```

+LOOP n --

Incrémente l'index de boucle de n.

Marque la fin d'une boucle `n1 0 do ... n2 +loop`.

```
: loopTest
   100 0 do
       i .
   5 +loop
  ;
loopTest  \ display 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
95
```

, x --

Ajoute x a la section courante de données.

```
create datas
    126 ,   352 ,   447 ,
datas 0 cells + @ .     \ display 126
datas 2 cells + @ .     \ display 447
```

- n1 n2 -- n3=n1-n2

Soustration de deux entiers.

```
6 3  -   \ push 3  operation 6-3
-6 3 -   \ push -9 operation -6-3
```

-ROT n1 n2 n3 -- n3 n1 n2

Equivaut à `ROT ROT`.

-TRAILING addr n1 -- addr n2

Ajustez le nombre de caractères n1 d'une chaîne de texte commençant à addr pour exclure les espaces de fin, c'est-à-dire que les caractères de addr+n2 à `-TRAILING` addr+n1-1 sont des espaces. Une condition d'erreur existe si n1 est négatif.

```
: myTxt
    s" this is my example   "
  ;
mytxt       type \ display: this is my example    ok
mytxt
  -trailing type \ display: this is my example ok
```

. n --

Dépile la valeur au sommet de la pile et l'affiche en tant qu'entier simple précision signé.

```
1 .                     \ display 1
1 2 .                   \ display 2  leave 1 on stack
1 2 + .                 \ display 3  addition 1 and 2, leave nothing on the
stack
6 3  * .                \ display 18
7 3  * 6 3 * + .        \ display 39 operation (7*3)+(6*3)
```

." -- <string>

Le mot ." est utilisable exclusivement dans une définition compilée.

A l'exécution, il affiche le texte compris entre ce mot et le caractère " délimitant la fin de chaîne de caractères.

```
: TITLE
    ."      GENERAL MENU" cr
    ."      ============" ;
: line1
    ." 1.. Enter datas" ;
: line2
    ." 2.. Display datas" ;
: last-line
    ." F.. end program" ;
: MENU ( -- )
    title cr cr cr
    line1 cr cr
    line2 cr cr
    last-line ;
```

.' addr --

Un mot SwiftForth qui affiche le nom de la définition la plus proche avant addr, et le décalage de addr par rapport au début de cette définition. « dot-tick »

.("ccc" --

Analyser et afficher ccc délimité par) (parenthèse droite). .(est un mot immédiat.

.R n1 n2 --

Affiche n1 aligné à droite dans un champ de n2 caractères de large.

Si le nombre de caractères requis pour afficher n1 est supérieur à n2, tous les chiffres sont affichés sans espaces de début dans un champ aussi large que nécessaire.

```
3 6 .r      \ display        3
```

```
254 6 .r     \ display      254
```

.S --

Affiche le contenu de la pile de données, sans action sur le contenu de cette pile.

.TAB --

Affiche le caractère TAB (tabluation).

/ n1 n2 -- n3

Division de deux entiers. Laisse le quotient entier sur la pile.

```
6 3  / .   \ display 2 opération 6/3
7 3  / .   \ display 2 opération 7/3
8 3  / .   \ display 2 opération 8/3
9 3  / .   \ display 3 opération 9/3
```

/MOD n n -- rem quot

16/16 -> division 16-bit signée.

/STRING c-addr1 u1 n -- c-addr2 u2

Ajuste la chaîne de caractères à c-addr1 de n caractères. La chaîne de caractères résultante, spécifiée par c-addr2 u2, commence à c-addr1 plus n caractères et mesure u1 moins n caractères.

0< n -- fl

Laisse flag vrai si n est inférieur à zéro.

0<> n -- fl

Empile -1 si n <> 0

```
1 0<>       \ push TRUE  on stack
0 0<>       \ push FALSE on stack
```

0= x -- fl

Teste si l'entier simple précision situé au sommet de la pile est nul.

```
5 0=       \ push  FALSE on stack
0 0=       \ push  TRUE  on stack
```

0> x1 -- fl

Teste si x1 est supérieur à zéro.

```
3 0> .    \ display: -1
-5 0> .   \ display: 0
```

1+ n -- n+1

Incrémente la valeur située au sommet de la pile.

```
17 1+ .    \ display 18
```

1- n -- n-1

Décrémente la valeur située au sommet de la pile.

```
17 1- .    \ display 16
```

2! d addr --

Stocke la valeur double précision d à l'adresse addr.

2* n -- n*2

Multiplie n par deux.

```
7 2* .     \ display 14
```

2+ n -- n+2

Incrémente de 2 la valeur située au sommet de la pile.

```
35 2+ .      \ display 37
```

2- TODELETE n -- n-2

Décrémente de 2 la valeur située au sommet de la pile.

```
5 2- .    \ display 3
```

2/ n -- n/2

Divise n par deux.

```
6 2/ .     \ display 3
7 2/ .     \ display 3
```

2>R S: d -- R: d

Transfère d vers la pile de retour.

Cette opération doit toujours être équilibrée avec `2r>`

2@ addr -- d

Récupère le contenu de 2 cellules.

2CONSTANT comp: d -- <name> | exec: -- d

Mot de création. Définit une constante double précision.

```
3.141529 2constant PI
```

2DROP d --

Enlève du sommet de la pile de données le nombre entier double précision qui s'y trouvait.

```
2. 5. 8.  2DROP   \ leave 2 and 5 in double precision on stack
```

2DUP d -- d d

Copie le nombre double précision situé au sommet de la pile de données. Equivaut à n1 n2 --- n1 n2 n1 n2.

```
5.  2DUP   \ leave 5. et 5. on stack
           \ 5. is a double precision number
```

2NIP d1 d2 -- d2

Supprime d1 et laisse d2 sur la pile.

```
1. 2. 2nip d.   \ display:  2
```

2OVER d1 d2 -- d1 d2 d1

Duplique d1 sur la pile.

```
1. 2. 2over d.   \ display:  1
```

2R> R: d -- S: d

Transfère d depuis la pile de retour.

Cette opération doit toujours être équilibrée avec `2>r`

2R@ R: d -- S: d

Récupère un entier double précision qui a été préalablement stocké sur la pile de retour par `2>r`

L'entier double précision sauvegardé sur la pile de retour par `2>r` doit être récupéré par `2r>` avant de quitter le mot.

```
: test ( d -- d d )
    2>r  2r@  2r>
  ;
```

2SWAP d1 d2 -- d2 d1

Inverse les deux valeurs double précision au sommet de la pile.

2VARIABLE comp: -- <name> | exec: -- addr

Mot de création. Définit une variable double précision.

```
2variable resistance
```

: comp: --

Le mot `:` est le mot de création le plus utilisé en FORTH.

L'exécution ultérieure de `NOM` réalise l'enchainement d'exécution des mots compilés dans sa définition "deux-points".

Après `: NOM`, l'interpréteur entre en mode compilation. Tous les mots non immédiats sont compilés dans la définition, les nombres sont compilés sous forme littérale. Seuls les mots immédiats ou placés entre crochets (mots `[` et `]`) sont exécutés pendant la compilation pour permettre de contrôler celle-ci.

Une définition "deux-points" reste invalide, c'est à dire non inscrite dans le vocabulaire courant, tant que l'interpréteur n'a pas exécuté `;` (point-virgule).

```
: NAME  nomex1 nomex2 ... nomexn ;
NAME  \ execute NAME
```

:NONAME -- cfa

Définit un code FORTH sans en-tête. cfa-addr est l'adresse d'exécution d'une définition.

```
:noname s" Saterday" ;
:noname s" Friday" ;
:noname s" Thursday" ;
:noname s" Wednesday" ;
:noname s" Tuesday" ;
```

```
:noname s" Monday" ;
:noname s" Sunday" ;

create (ENday) ( --- addr)
  , , , , , , ,

:noname s" Samedi" ;
:noname s" Vendredi" ;
:noname s" Jeudi" ;
:noname s" Mercredi" ;
:noname s" Mardi" ;
:noname s" Lundi" ;
:noname s" Dimanche" ;

create (FRday) ( --- addr)
  , , , , , , ,

defer (day)

: ENdays
    ['] (ENday) is (day) ;

: FRdays
    ['] (FRday) is (day) ;

3 value dayLength
: day
    (day)
    swap cells +
    @ execute
    dayLength ?dup if
        min
    then
    type
;
ENdays
0 day   \ display Sun
1 day   \ display Mon
2 day   \ display Tue
FRdays  ok
0 day   \ display Dim
1 day   \ display Lun
2 day   \ display Mar
```

; --

Mot d'exécution immédiate terminant habituellement la compilation d'une définition "deux-points".

```
: NAME
    nomex1 nomex2 ... nomexn ;
```

< x1 x2 -- fl

Teste si x1 est inférieur à x2.

```
3 5 < .      \ display -1
5 -1 < .     \ display 0
-5 -1 < .    \ display -1
```

<# --

Compilation seulement.

Débute une conversion numérique.

```
\ display byte in binary format
: x. ( c -- )
    0 base @ >r
    2 base !
    s>d
    <# # # # # # # # # #> type
    r> base !
  ;
```

<= n1 n2 -- fl

Laisse fl vrai si n1 <= n2

```
4 10 <=   \ leave -1 on stack
4 4  <=   \ leave -1 on stack
4 3  <=   \ leave  0 on stack
```

<> x1 x2 -- fl

Teste si l'entier simple précision x1 n'est pas égal à x2.

```
5 5 <>      \ push  FALSE on stack
5 4 <>      \ push  TRUE  on stack
```

= x1 x2 -- fl

Teste si l'entier simple précision x1 est égal à x2.

```
5 5 =       \ push  TRUE  on stack
5 4 =       \ push  FALSE on stack
```

> x1 x2 -- fl

Teste si x1 est supérieur à x2.

```
3 5 > .       \ display  0
5 -1 > .      \ display -1
-1 -5 > .     \ display -1
```

>BODY xt -- a-addr

a-addr est l'adresse du champ de données correspondant à xt.

>IN -- a-addr

a-addr est l'adresse d'une cellule contenant le décalage en caractères du début du tampon d'entrée au début de la zone d'analyse.

>NUMBER ud1 c-addr1 u1 -- ud2 c-addr2 u2

ud2 est le résultat non signé de la conversion des caractères de la chaîne spécifiée par c-addr1 u1 en chiffres, en utilisant le nombre dans BASE, et en ajoutant chacun dans ud1 après avoir multiplié ud1 par le nombre dans BASE.

La conversion continue de gauche à droite jusqu'à ce qu'un caractère non convertible, y compris un « + » ou un « - », soit rencontré ou que la chaîne soit entièrement convertie. c-addr2 est l'emplacement du premier caractère non converti ou du premier caractère après la fin de la chaîne si la chaîne a été entièrement convertie. u2 est le nombre de caractères non convertis dans la chaîne.

>R x -- | R: -- x

COMPILATION UNIQUEMENT.

Pousse x de la pile de paramètres vers la pile de retour.

```
\ display n in binary format
: b. ( n -- )
    base @ >r
    2 base !  .
    r> base !
  ;
```

? a-addr --

Affiche le contenu stocké à l'adresse a-addr.

?DO n1 n2 --

Exécute une boucle `do loop` ou `do +loop` si n1 est strictement supérieur à n2.

```
DECIMAL
: qd ?DO I LOOP ;
   789   789 qd \
 -9876 -9876 qd \
     5     0 qd \  display: 0 1 2 3 4
```

?DUP n -- n | n n

Duplique n si n n'est pas nul.

@ addr -- n

Récupère la valeur entière n stockée à l'adresse addr.

```
variable SCORE
36 SCORE !
1 SCORE +!
SCORE @ .     \ display 37
```

ABORT --

Restaure la pile de retour et exécute `quit`.

ABORT" comp: -- <error message>

Affiche un message d'erreur et interrompt toute exécution FORTH en cours.

```
: abort-test
    if
        abort" stop program"
    then
    ." continue program"
  ;

0 abort-test   \ display: continue program
1 abort-test   \ display: stop program ERROR
```

ABS n -- n'

Renvoie la valeur absolue de n.

```
-7 abs .     \ display 7
```

ACCEPT c-addr +n -- +n'

Saisie d'une ligne depuis le terminal.

AGAIN addr --

Termine une structure `begin ... again`.

ALIGN --

Aligne le pointeur du dictionnaire de la section de données actuelle sur la limite de la cellule.

ALIGNED addr -- a-addr

Aligne addr sur un espace de cellule.

ALLOT n --

Ajuste le pointeur de dictionnaire dans la section de données courante.

```
create myScores
    4 cells allot   \ allot 8 bytes in memory
```

AND n1 n2 -- n3

Effectue un ET logique.

```
false  false and .  \ display 0
false  true  and .  \ display 0
true   false and .  \ display 0
true   true  and .  \ display -1
```

BASE -- addr

Variable simple précision déterminant la base numérique courante.

La variable `BASE` contient la valeur 10 (décimal) au démarrage de FORTH.

```
DECIMAL     \ select decimal base
2 BASE !    \ select binary base
```

BEGIN -- addr

COMPILATION SEULEMENT

egin ... again

begin ... until

begin ... while ... repeat

BL -- 32

Constant. Stack 32 which is the ascii code for the 'space' character.

BLANK addr len --

Si len est supérieur à zéro, range un caractère de code $20 (espace) dans toute la zone de longueur len à l'adresse mémoire commençant à addr.

BLK -- a-addr

a-addr est l'adresse d'une cellule contenant zéro ou le numéro du bloc de stockage de masse en cours d'interprétation.

BLOCK u -- a-addr

a-addr est l'adresse du premier caractère du tampon de bloc attribué au bloc de stockage de masse u.

BUFFER u -- a-addr

a-addr est l'adresse du premier caractère du tampon de bloc affecté au bloc u. Le contenu du bloc n'est pas spécifié.

Le bloc n'est pas lu à partir de la mémoire de masse. Si le contenu précédent du tampon a été marqué comme UPDATEd, il est écrit dans la mémoire de masse. Si l'écriture correcte dans la mémoire de masse n'est pas possible, une condition d'erreur existe. L'adresse restante est le premier octet dans la mémoire tampon pour le stockage des données.

n est un nombre non signé.

BYE --

Relance FORTH.

```
bye
\ display:
Z79Forth/AI 6309 ANS Forth System
20240628 (C) Francois Laagel 2019

RAM OK: 32 KB
SanDisk SDCFJ-128
Block #1 loaded
```

C! c addr --

Stocke une valeur 8 bits c à l'adresse addr.

C, c --

Ajoute c à la section courante de données.

```
create myPresets
    $04 c,  $2f c,  $40 c,
```

C@ addr -- c

Récupère la valeur 8 bits c stockée à l'adresse addr.

```
35 constant PINB  \ adresse registre données PIN de PORT B sur Arduino
PINB c@           \ empile contenu registre pointé par PINB
```

CELL+ a-addr1 -- a-addr2

Ajoute la taille en unités d'adresse d'une cellule à a-addr1, ce qui donne a-addr2.

CELLS n1 -- n2

n2 est la taille en unités d'adresse de n1 cellules.

```
CREATE NUMBERS
    100 CELLS ALLOT
```

CHAR -- <string>

Mot utilisable en interprétation seulement.

Empile le premier caractère de la chaîne qui suit ce mot.

```
char v .          \ display: 118 (ascii code for "v")
char house .      \ display: 104 - code for "h"
```

CHAR+ c-addr1 -- c-addr2

Ajoute la taille en unités d'adresse d'un caractère à c-addr1, ce qui donne c-addr2.

CHARS n1 -- n2

n2 est la taille en unités d'adresse de n1 caractères.

CMOVE addr1 addr2 u --

Déplace u caractères de addr1 vers addr2.

COMPARE c-addr1 u1 c-addr2 u2 -- n

Compare la chaîne spécifiée par c-addr1 u1 à la chaîne spécifiée par c-addr2 u2.

Les chaînes sont comparées, en commençant aux adresses données, caractère par caractère, jusqu'à la longueur de la chaîne la plus courte ou jusqu'à ce qu'une différence soit trouvée. Si les deux chaînes sont identiques, n est égal à zéro. Si les deux chaînes sont identiques jusqu'à la longueur de la chaîne la plus courte, n est égal à moins un (-1) si u1 est inférieur à u2 et à un (1) sinon. Si les deux chaînes ne sont pas identiques jusqu'à la longueur de la chaîne la plus courte, n est égal à moins un (-1) si le premier

caractère non correspondant dans la chaîne spécifiée par c-addr1 u1 a une valeur numérique inférieure à celle du caractère correspondant dans la chaîne spécifiée par c-addr2 u2 et à un (1) sinon.

CONSTANT comp: n -- <name> | exec: -- n

Mot de création. Définit une constante simple précision:

```
\ definition of constant
130 constant speed-max
speed-max .     \ display 130
```

COUNT c-addr1 -- c-addr2 u

Renvoie la spécification de chaîne de caractères pour la chaîne comptée stockée à c-addr1. c-addr2 est l'adresse du premier caractère après c-addr1. u est le contenu du caractère à c-addr1, qui est la longueur en caractères de la chaîne à c-addr2.

CR --

Affiche un retour à la ligne suivante.

```
: .result ( ---)
    ." Port analys result" cr
    . "pool detectors" cr ;
```

CREATE comp: -- | exec: -- addr

Crée un nouveau mot.

Le mot **CREATE** peut être utilisé seul.

```
CREATE DATAS  ( --- addr)
    25 c, 32 c, 44 c, 17 c,
```

D+ d1 d2 -- d3

Addition de nombres double précision.

```
35. 7. d+      \ leave 35. + 7. on the stack
```

D- d1 d2 -- d3

Soustraction de nombres double précision.

```
35. 7. d-   \ leave result of 35-7 in double precision
```

D. d --

Affiche d.

Ce mot n'est pas défini dans le dictionnaire Z79Forth.

```
\ leave absolute value of d if d<
: DABS ( d -- d' )
    dup 0< if
        dnegate
    then
;

\ leave addr len on stack resulting of d conversion in string
: (D.)  ( d -- a l )
    tuck dabs <# #s rot sign #>
;

\ display d
: D.  ( d -- )
  (d.) type space
;
```

D0= d -- fl

Vrai si d égal zéro.

D< d1 d2 -- flag

flag est vrai si et seulement si d1 est inférieur à d2.

DECIMAL --

Sélectionne la base numérique décimale. C'est la base numérique par défaut au démarrage de FORTH.

```
255 hex . decimal   \ display FF
```

DEFER -- <vec-name>

Définit un vecteur d'exécution différée.

`vec-name` exécute le mot dont le code d'exécution est stocké dans l'espace de données de vec-name.

```
defer xEmit
: vxEmit ( c ---)
    1+ emit ;
' vxEmit is xEmit
```

DEFER! xt2 xt1 --

Définit le mot xt1 pour exécuter xt2.

DEFER@ xt1 -- xt2

xt2 est le jeton d'exécution que xt1 est configuré pour exécuter.

DEPTH -- n

n est le nombre de valeurs simple précision contenues dans la pile de données avant que n ne soit placé sur la pile.

```
depth .    \ display 0
55 22 4    depth .    \ display 3
```

DNEGATE d -- -d

Rend négatif un nombre entier double précision.

```
4. dnegate d.    \ display -4
```

DO n1 n2 --

Configure les paramètres de contrôle de boucle avec l'index n2 et la limite n1.

```
: testLoop
    256 32 do
        I emit
    loop
  ;
```

DOES> comp: -- | exec: -- addr

Le mot **CREATE** peut être utilisé dans un nouveau mot de création de mots...

Associé à **DOES>**, on peut définir des mots qui disent comment un mot est créé puis exécuté.

DROP n --

Enlève du sommet de la pile de données le nombre entier simple précision qui s'y trouvait.

```
2 5 8  drop   \ leave 2 and 5 on stack
```

DUMPLOAD --

Compile le contenu des blocs 150 à 154.

Ces blocs contiennent les définitions permettant de procéder à un dump mémoire.

DUP n -- n n

Duplique le nombre entier simple précision situé au sommet de la pile de données.

```
: SQUARE ( n --- nE2)
    DUP * ;
 5 SQUARE  .    \ display 25
10 SQUARE  .    \ display 100
```

ELSE --

Mot d'exécution immédiate et utilisé en compilation seulement. Marque une alternative dans une structure de contrôle du type:

```
(condition) IF ... ELSE ... THEN ...
```

A l'exécution, si la condition située sur la pile avant **IF** est fausse, il y a rupture de séquence avec saut à la suite de **ELSE**, puis reprise en séquence après **THEN**

```
: TEST ( ---)
    cr ." Press a key " key
    dup 65 122 between
    if
        cr 3 spaces ." c'est une lettre "
    else
        dup 48 57 between
        if
            cr 3 spaces ." is a digit "
        else
            cr 3 spaces ." is a special character "
        then
    then
    drop ;
```

EMIT c --

Affiche le caractère de code c.

Si x est un caractère graphique dans le jeu de caractères défini par l'implémentation, affiche x.

L'effet d'**emit** pour toutes les autres valeurs de x est défini par l'implémentation.

Lors du passage d'un caractère dont les bits de définition de caractère ont une valeur comprise entre hex 20 et 7E inclus, le caractère standard correspondant s'affiche. Étant donné que différents périphériques de sortie peuvent répondre différemment aux caractères de contrôle, les programmes qui utilisent des caractères de contrôle pour exécuter des fonctions spécifiques ont une dépendance environnementale. Chaque **emit** ne traite qu'avec un seul caractère.

```
65 emit    \ display A
66 emit    \ display B
```

EMPTY-BUFFERS --

Annule l'affectation de tous les tampons de blocs. Ne transfère pas le contenu d'un tampon de bloc mis à jour vers le stockage de masse.

ERASE addr u --

Si u est supérieur à zéro, effacez tous les bits de chacune des u unités d'adresse consécutives de la mémoire commençant à addr.

EVALUATE addr-c u --

Enregistre la spécification de la source d'entrée actuelle. Fait de la chaîne décrite par c-addr et u la source d'entrée et le tampon d'entrée, définit `>IN` sur zéro et interprète le contenu de la chaîne. Lorsque la zone d'analyse est vide, restaure la spécification de la source d'entrée précédente.

```
: GE1 S" 123" ;
: GE2 S" 123 1+" ;
GE1 EVALUATE .   \ display: 123
GE2 EVALUATE .   \ display: 124
```

EXECUTE cfa --

Exécute un mot depuis son adresse cfa

Pred l'adresse d'exécution cfa de la pile de données et exécute le mot.

```
' words
execute     \ execute WORDS from his execution code address
```

EXIT --

Interrompt l'exécution d'un mot et rend la main au mot appelant.

Utilisation typique: `: X ... test IF ... EXIT THEN ... ;`

FALSE -- 0

Constante pré-définie. Empile 0.

```
false .    \ display: 0
```

FILL addr len c --

Si len est supérieur à zéro, range c dans toute la zone de longueur len à l'adresse mémoire commençant à addr.

FIND c-addr -- c-addr 0 | xt 1 | xt -1

Recherche la définition nommée dans la chaîne comptée à c-addr.

Si la définition n'est pas trouvée, renvoie c-addr et zéro. Si la définition est trouvée, renvoie son jeton d'exécution xt. Si la définition est immédiate, renvoie également un (1), sinon renvoie également moins un (-1). Pour une chaîne donnée, les valeurs renvoyées par FIND lors de la compilation peuvent différer de celles renvoyées sans compilation.

FLUSH --

Synonyme pour **SAVE-BUFFERS**.

FM/MOD d1 n1 -- n2 n3

Divise d1 par n1, ce qui donne le quotient plancher n3 et le reste n2. Les arguments de la pile d'entrée et de sortie sont signés. Une condition ambiguë existe si n1 est nul ou si le quotient se situe en dehors de la plage d'un entier signé à une seule cellule.

H. n --

Affiche n au format hexadécimal.

```
decimal
16 h.       \ display 10
35 h.       \ display 23
```

HERE -- addr

Restitue l'adresse courante du pointeur de dictionnaire.

HEX --

Sélectionne la base numérique hexadécimale.

```
255 HEX .    \ display FF
DECIMAL      \ return to decimal base
```

HOLD c --

Insère le code ASCII d'un caractère ASCII dans la chaîne de caractères initiée par <#.

I -- n

n est une copie de l'index de boucle actuel.

```
: mySingleLoop ( -- )
    cr
    10 0 do
        i .
    loop
  ;
mySingleLoop
\ display 0 1 2 3 4 5 6 7 8 9
```

IF fl --

Le mot **IF** est d'exécution immédiate.

IF marque le début d'une structure de contrôle de type **IF..THEN** ou **IF..ELSE..THEN**.

Lors de l'exécution, la partie de définition située entre **IF** et **THEN** ou entre **IF** et **ELSE** est exécutée si le flag booléen situé au sommet de la pile de données est vrai (f<>0).

Dans le cas contraire, si le flag booléen est faux (f=0), c'est la partie de définition située entre **ELSE** et **THEN** qui sera exécutée. S'il n'y a pas de **ELSE**, l'exécution se poursuit après **THEN**.

```
: Nice? ( fl ---)
    if
        ." Nice weather "
    else
        ." Cloudy weather "
    then
  ;
1 Nice?    \ display: Nice weather
0 Nice?    \ display: Cloudy weather
```

IMMEDIATE --

rend la définition la plus récente comme mot immédiat.

INDEX n1 n2 --

Imprimez la première ligne de chaque écran sur la plage {n1..n2}.

Ceci affiche la première ligne de chaque écran de texte source, qui contient traditionnellement un titre.

INTERPRET addr len --

Interprète le contenu du tampon.

Commence l'interprétation au niveau du caractère indexé par le contenu de **>IN** par rapport au numéro de bloc contenu dans **BLK**, en continuant jusqu'à ce que le flux d'entrée soit épuisé. Si **BLK** contient zéro, interprète les caractères à partir du tampon d'entrée du terminal.

INVERT x1 -- x2

Complément à un de x1.

```
1 invert .  \ display -2 (%1111111111111110)
```

IS --

Assigns the execution code of a word to a vectorized execution word.

```
defer xEmit
: vxEmit ( c ---)
    1+ emit ;
' vxEmit is xEmit
```

J -- n | u

n | u est une copie de l'index de la boucle externe suivante.

```
: twoLoops ( -- )
  cr
  5 0 do
    5 0 do
      j i . . cr
    loop
  loop ;
twoLoops  \ display:
0 0
1 0
2 0
3 0
4 0
0 1
1 1
2 1
3 1.....
```

J' -- n

Renvoie le quatrième élément de la pile de retour.

K -- n

Dans une boucle **DO**..**LOOP** imbriquée, renvoyez l'index de la deuxième boucle externe.

KEY -- c

Attend l'appui sur une touche. L'appui sur une touche renvoie son code ASCII.

```
key .    \ display 97 if key "a" is active
key .    \ affiche 65 if key "A" is active
```

KEY? -- fl

Renvoie *vrai* si une touche est appuyée.

```
: keyLoop
    begin
    key? until
  ;
```

LAST -- addr

Variable contenant l'adresse du début de la dernière entrée de dictionnaire créée, qui peut ne pas encore être une entrée complète ou valide.

LIST n --

Affiche le contenu du bloc n.

LITERAL x --

Compile la valeur x comme valeur litérale.

```
: valueReg ( --- n)
   [ 36 2 * ] literal
 ;
```

LOOP --

Ajoute un à l'index de la boucle. Si l'index de boucle est alors égal à la limite de boucle, supprime les paramètres de boucle et poursuit l'exécution immédiatement après la boucle. Sinon, continue l'exécution au début de la boucle.

```
: myLoop ( -- )
    10 0 do
        i .
    loop
  ;
myLoop \ display: 0 1 2 3 4 5 6 7 8 9
```

LSHIFT x1 n -- x2

Décalage de x1 vers la gauche de n bits.

```
2 3 lshift .   \ display: 16
```

M* n1 n2 -- d

d est le produit signé de n1 par n2.

MARKER comp: -- <name> | exec: --

Ignorer les espaces de délimiteurs de début. Analyser le nom délimité par un espace. Créer une définition pour le nom avec la sémantique d'exécution définie ci-dessous.

L'exécution de restaure tous les pointeurs d'allocation de dictionnaire et d'ordre de recherche à l'état qu'ils avaient juste avant la définition du nom. Supprimer la définition du nom et toutes les définitions suivantes. La restauration de toutes les structures encore existantes qui pourraient faire référence à des définitions supprimées ou à un espace de données désalloué n'est pas nécessairement fournie. Aucune autre information contextuelle telle que la base numérique n'est affectée.

```
marker --wx
: w1 ;
: w2 ;
--wx  \ delete words --wx w1 w2 in dictionnary
```

MAX n1 n2 -- n1|n2

Laisse max de n1 et n2.

MIN n1 n2 -- n1|n2

Laisse min de n1 et n2.

MOD n1 n2 -- n3

Divise n1 par n2, laisse le reste simple précision n3.

```
21 7 mod . \ display 0
22 7 mod . \ display 1
23 7 mod . \ display 2
24 7 mod . \ display 3

\ The modulo function can be used to determine the
\ divisibility of one number by another:
: div? ( n1 n2 ---)
    over over mod cr
    if
        swap . ." is not "
    else
        swap . ." is "
```

```
    then
    ." divisible by " .
  ;
```

MOVE c-addr1 c-addr2 u --

Si u est supérieur à zéro, copier u caractères consécutifs de l'espace de données commençant à c-addr1 vers celui commençant à c-addr2, en procédant caractère par caractère des adresses inférieures aux adresses supérieures.

MS n --

Attente en millisecondes.

ATTENTION : le mot `ms` bloque tous les autres processus. Pour les attentes longues, il est conseillé de fractionner l'attente longue en une succession d'attentes courtes dans une boucle de type **begin..until** par exemple.

```
500 ms \ delay for 1/2 second
```

NCLR i*x --

Vide la pile de données.

NEGATE n -- n'

Le complément à deux de n.

```
5 negate .   \ display -5
```

NIP n1 n2 -- n2

Enlève n1 de la pile.

NOT x1 -- x2

Inverse tous les bits.

OR n1 n2 -- n3

Effectue un OU logique.

```
false  false or  .  \ display 0
false  true  or  .  \ display -1
true   false or  .  \ display -1
true   true  or  .  \ display -1
```

OVER n1 n2 -- n1 n2 n1

Place une copie de n1 au sommet de la pile.

```
2 5 OVER  \ duplicate 2 on top of the stack
```

PAGE --

Efface l'écran.

PAYLOAD -- nbytes

Cette sortie primitive n'est pertinente qu'après une invocation de ' ou FIND.

Il récupère la longueur de définition d'un mot (section de code) correspondant à la dernière recherche dans le dictionnaire. Ce mot est d'une importance marginale. Cependant, il facilite une implémentation du désassembleur dans laquelle cette charge utile n'a pas besoin d'être spécifiée au moment de l'invocation de DIS (voir SW/examples/dis.4th pour une implémentation minimale du désassembleur).

PICK xu ... x1 x0 u -- xu ... x1 x0 xu

Enlève u. Copie xu au sommet de la pile.

POSTPONE -- <name>

Ignore les délimiteurs d'espaces de début. Analyse *name* délimité par un espace. Trouve *name*. Ajoutez la sémantique de compilation de *name* à la définition actuelle.

POSTPONE remplace la plupart des fonctionnalités de COMPILE et [COMPILE]. COMPILE et [COMPILE] sont utilisés dans le même but : reporter le comportement de compilation du mot suivant dans la zone d'analyse. COMPILE a été conçu pour être appliqué aux mots non immédiats et [COMPILE] aux mots immédiats.

```
: ACTION-OF
   STATE @ IF
     POSTPONE [']  POSTPONE DEFER@
   ELSE
     ' DEFER@
   THEN ; IMMEDIATE
```

R> R: n -- S: n

Transfère n depuis la pile de retour.

Cette opération doit toujours être équilibrée avec >r

```
\ display n in binary format
: b. ( n -- )
```

```
    base @ >r
    2 base !
    .
    r> base !
  ;
```

R@ -- x | R: x -- x

Copie x de la pile de retour vers la pile de données.

Attention: ne fonctionne pas dans une boucle `do..loop`.

RCLR R: i*x --

Vide la pile de retour.

RECURSE --

Ajoute un lien d'exécution correspondant à la définition actuelle.

L'exemple habituel est le codage de la fonction factorielle.

```
: FACTORIAL ( +n1 -- +n2)
   DUP 2 < IF DROP 1 EXIT THEN
   DUP 1- RECURSE *
;
```

REPEAT --

Achève une boucle indéfinie `begin.. while.. repeat`

RESTRICT --

Rend la dernière définition non anonyme disponible en mode compilation uniquement.

ROT n1 n2 n3 -- n2 n3 n1

Faites pivoter trois éléments au sommet de la pile.

RSHIFT x1 u -- x2

Décalage vers la droite de u bits de la valeur x1.

```
64 2 rshift .  \ display 16
```

RTC! rtcbyteval rtcregoffset --

Écrit dans un registre RTC.

Veuillez noter que le support RTC dans Z79Forth/A n'est pas officiel.

RTC@ rtcregoffset -- rtcbyteval

Récupère le contenu d'un registre RTC.

Veuillez noter que le support RTC dans Z79Forth/A n'est pas officiel.

S -- sreg

Renvoie le contenu du pointeur de pile système (S).

S" comp: -- <string> | exec: addr len

En interprétation, laisse sur la pile de données la chaine délimitée par "

En compilation, compile la chaine délimitée par "

Lors de l'exécution du mot compilé, restitue l'adresse et la longueur de la chaîne...

```
\ header for DUMP
: headDump
    s" --addr----- 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F"
  ;
headDump         \ push addr len on stack
headDump type    \ display: --addr----- 00 01 02 03 04 05 06 07 08 09 0A 0B
0C 0D 0E 0F
```

S>D n -- d

Convertit un entier simple précision signé en entier double précision.

```
57 s>d . .    \ display 0 57
```

S@ -- retaddr

Renvoie l'adresse de retour à l'appelant.

Ceci est particulièrement utile à des fins de débogage lorsqu'il est utilisé en conjonction avec `.'`. Veuillez noter que pour que `.'` puisse fournir des informations symboliques utiles, `RELFEAT` aurait dû être activé au moment de la compilation (par défaut).

SEE -- <name>

Décompile/désassemble le mot <name>.

```
see dup    \ display:
FC85 BDE9D7      jsr      CKDPTRA
FC88 AE          fcb      $AE
FC89 C4          fcb      $C4
FC8A 7EE7DB      jmp      NPUSH ok
```

SHIFT n1 n2 --

Décale n1 de n2 bits vers la gauche si n2 est positif, sinon décale vers la droite si n2 est négatif.

```
4  1 shift .   \ display: 8
 4  2 shift .   \ display: 16
 2  0 shift .   \ display: 2
16 -1 shift .   \ display: 8
```

SIGN n --

Si n est négatif, ajoutez un signe moins au début de la chaîne de sortie numérique mise en forme par `<# #>`.

SPACE --

Affiche un caractère espace.

SPACES n --

Si n est supérieur à zéro, affiche n fois le caractère espace.

STATE -- fl

État de compilation. L'état ne peut être modifié que par `[` et `]`.

SWAP n1 n2 -- n2 n1

Echange les valeurs situées au sommet de la pile.

```
2 5 swap
.   \ display 2
.   \ display 5
```

TAB -- 9

Empile la constante 9.

Cette constante permet d'utiliser le caractère *tabulation* qui ne peut pas être récupéré par `char` ou `[char]`.

THEN --

Mot d'exécution immédiate utilisé en compilation seulement. Marque la fin d'une structure de contrôle de type `IF..THEN` ou `IF..ELSE..THEN`.

THRU n1 n2 --

Charge le contenu d'un fichier de blocs, du bloc n1 au bloc n2.

TICKS -- ud

Impulsions système. 64 impulsions par milliseconde.

Empile un entier double contenant le nombre de ticks depuis le démarrage. Il y a 64 ticks par seconde. Veuillez noter que la prise en charge RTC dans Z79Forth/A n'est pas officielle.

TO n --- <valname>

`to` affecte une nouvelle valeur à *valname*

TOUPPER c -- c'

Transforme le code ASCII d'un caractère compris entre [a..z] en son équivalent de code compris entre [A..Z].

```
char e toupper emit  \ display E
```

TRUE -- -1

Constante pré-définie. Empile -1.

```
true .    \ display: -1
```

TUCK n1 n2 -- n2 n1 n2

Insère n2 avant n1 dans la pile de données.

TYPE addr len --

Affiche la chaine de caractères sur len octets.

```
: hello ( --- addr c)
  s" Hello world" ;
hello type
```

U. x --

Dépile la valeur au sommet de la pile et l'affiche en tant qu'entier simple précision non signé.

```
1 u.                    \ display 1
-1 u.                   \ display 65535
```

U.R u n --

Affiche u aligné à droite dans un champ de n caractères de large.

Si le nombre de caractères requis pour afficher u est supérieur à n, tous les chiffres sont affichés sans espaces de début dans un champ aussi large que nécessaire.

U< u1 u2 -- flag

flag est vrai si et seulement si u1 est inférieur à u2.

U> u1 u2 -- flag

flag est vrai si et seulement si u1 est supérieur à u2.

UM* u1 u2 -- ud

Multiplication non signée 16x16 -> 32 bit

UM/MOD ud u1 -- u2 u3

Divise ud par u1, ce qui donne le quotient u3 et le reste u2. Toutes les valeurs et opérations arithmétiques sont non signées. Une condition ambiguë existe si u1 est nul ou si le quotient se situe en dehors de la plage d'un entier non signé à une seule cellule.

UNLESS --

Le bloc de code couvert par cette primitive d'exécution conditionnelle doit être terminé par un **THEN** correspondant.

Inspiré de Perl. Fonctionnellement équivalent à :

```
: UNLESS

  ['] 0= COMPILE, POSTPONE IF

  ; IMMEDIATE RESTRICT
```

UNLOOP --

Ignorez les paramètres de contrôle de boucle pour le niveau d'imbrication actuel. Un **UNLOOP** est requis pour chaque niveau d'imbrication avant que la définition puisse être arrêtée.

UNMONITOR --

Marque la dernière définition non anonyme comme non ciblée pour les contrôles d'intégrité par **ICHECK**. La raison d'être de ce mot est de prendre en charge les VALEURS (voir SW/examples/ansiextern.4th).

UNTIL fl --

COMPILATION SEULEMENT

begin..until

UNUSED -- u

u est la quantité d'espace restant dans la région adressée par **HERE**, en unités d'adresse.

VALUE comp: n -- <valname> | exec: -- n

Crée un mot de type *value*

valname empile la valeur.

Un mot défini par **value** est semblable à une constante, mais dont la valeur peut être modifiée.

```
12 value APPLES     \ Define APPLES with an initial value of 12
34 to APPLES        \ Change the value of APPLES. to is a parsing word
APPLES              \ puts 34 on the top of the stack
```

VARIABLE comp: -- <name> | exec: -- addr

Mot de création. Définit une variable simple précision.

```
\ define variable speed
variable speed
35 speed !          \ store 32 in speed
10 speed +!         \ increment content of speed
speed @ .           \ display 45
```

WHILE fl --

Marque la partie d'exécution conditionnelle d'une structure **begin..while..repeat**

WITHIN n1 | u1 n2 | u2 n3 | u3 -- flag

Effectuer une comparaison d'une valeur de test n1 | u1 avec une limite inférieure n2 | u2 et une limite supérieure n3 | u3, en renvoyant vrai si (n2 | u2 < n3 | u3 et (n2 | u2 <= n1 | u1 et n1 | u1 < n3 | u3)) ou (n2 | u2 > n3 | u3 et (n2 | u2 <= n1 | u1 ou n1 | u1 < n3 | u3)) est vrai, en renvoyant faux dans le cas contraire.

WORD char "ccc" -- c-addr

Ignorer les délimiteurs de début. Analyser les caractères ccc délimités par char.

c-addr est l'adresse d'une région transitoire contenant le mot analysé sous forme de chaîne comptée. Si la zone d'analyse était vide ou ne contenait aucun caractère autre que le délimiteur, la chaîne résultante a une longueur nulle. Un programme peut remplacer des caractères dans la chaîne.

WORDS --

Répertorie les noms de définition dans la première liste de mots de l'ordre de recherche. Le format de l'affichage dépend de l'implémentation.

XOR n1 n2 -- n3

Effectue un OU eXclusif logique.

```
false false xor .  \ display 0
false true  xor .  \ display -1
true  false xor .  \ display -1
true  true  xor .  \ display 0
```

[--

Entre en mode interprétation. [est un mot d'exécution immédiate.

['] -- <name>

COMPILATION SEULEMENT.

compile l'adresse d'exécution de name comme valeur litérale.

[CHAR] comp: -- <spaces>name | exec: -- xchar

En compilation, enregistre le code ASCII du caractère indiqué après ce mot.

En exécution, le code xchar est déposé sur la pile de données.

```
: GC1 [CHAR] X     ;
: GC2 [CHAR] HELLO ;
GC1 \ push   58
GC2 \ push   48
```

\ --

Ignore le reste de la ligne.

The word \ must be followed by at least one space character and completed by the line comment.

```
2 3 + .  \ display: 5
```

] --

Retour en mode compilation.] est un mot immédiat.