Manuel en ligne

TELE-FORTH v1.2

LANGAGE pour ORIC TELESTRAT

par

Christophe LAVARENNE (version 1.1),
Thierry BESTEL (version 1.2)

Mises à jour du manuel :

- 28/11/95 - Thierry BESTEL
  création (scannérisation du manuel papier de la version 1.1, mise en page sous groff - jeu de caractères ISO_8859_1 latin1 - et mise à jour pour TELE-FORTH v1.2)

SOMMAIRE

1. UNE SEULE PAGE POUR CONVAINCRE...

Je n'ai plus à adapter au langage ma manière de penser :
avec FORTH, j'adapte le langage à ma manière de penser.

CE MANUEL en lui-même ne constitue pas un cours de programmation en FORTH ; il existe maintenant d'excellents ouvrages en langue française, et à différents niveaux de compétence, dont les références sont récapitulées en annexe bibliographique ; le but de cet ouvrage est bien plus ambitieux : vous permettre de tout savoir sur le fonctionnement du FORTH pour pouvoir en tirer le meilleur parti ; vous avez entre vos mains les sources mêmes qui ont été utilisés pour générer votre FORTH en cartouche. Nous avons également pensé à votre budget, aussi ne nous sommes-nous pas étendus sur les fonctions du TELESTRAT qui sont déjà décrites en détail dans les manuels BASIC et TELEMATIC ; l'effort maximum a été fait tant pour intégrer au FORTH toutes les instructions spécifiques au TELESTRAT (sonores, graphiques, télématiques, de gestion des disquettes et des entrées/sorties) sous le même nom et avec les mêmes arguments qu'en BASIC, que pour proposer de nouveaux outils tirant le meilleur parti de la puissance du TELESTRAT.

POUR CEUX QUI AIMENT FORTH, TELE-FORTH supporte les 180 mots du standard Fig-FORTH ; de plus, la plupart des extensions habituelles sont implémentées (la construction CASE, les opérateurs 32 bits, #IN...), et plus (opérateurs sur bit, vecteurs "defer", décompilateur, modules relogeables, fichier de démarrage...). Toutes les familles d'outils du TELESTRAT sont accessibles par des interfaces génériques FORTH, et plus de 90% des outils sont directement accessibles par des mots FORTH dédiés, regroupés dans plusieurs vocabulaires : au total, plus de 500 mots... En plus, un éditeur de lignes standard est intégré en ROM, et trois utilitaires sont disponibles sur disquette, en source et en relogeable : un éditeur pleine page, un macro-assembleur et une librairie de calcul en virgule flottante (a ma connaissance, cette disquette n'a jamais été commercialisée. Au besoin, on pourra porter les sources des mêmes utilitaires qui sont disponibles en FORTH 83-STANDARD). Enfin, un programme de démonstration à but pédagogique est intégré en ROM : il s'agit du Jeu de la Vie, fonctionnant aussi bien en mode texte qu'en mode haute résolution, dans une implémentation originale sans effets de bords.

POUR LES PROFESSIONNELS qui désirent profiter de l'excellent rapport performance/prix du TELESTRAT pour des applications télématiques par exemple, le FORTH permet des développements rapides, les tests se faisant intéractivement au fur et à mesure des développements et sans perdre le contexte de compilation, génère des applications performantes, compactes et raisonnablement protégées, et même des applications d'un volume supérieur à la mémoire disponible, grâce à la possibilité de diviser un programme en plusieurs segments relogeables dynamiquement. Un macro-assembleur, relogeable dans l'environnement FORTH, permet d'optimiser les temps d'exécution des fonctions critiques (repérées par des

outils d'analyse statistique automatique des fréquences d'utilisation, ceci afin de minimiser les efforts d'amélioration de performance). Ceux qui désirent utiliser FORTH pour générer des applications en EPROM peuvent envisager l'emploi du "méta-générateur" (Micro Processor Eng Ltd, Nautilus cross compiler v2.4), lui-même écrit en FORTH, et des sources qui ont permis de générer "TELE-FORTH pour ORIC".

QUE LE FORTH SOIT AVEC VOUS !

2. BIENVENUE SUR TELE-FORTH

Ce manuel est le boitier transparent didactique du chronomètre FORTH : il permet d'en voir tous les rouages dans les moindres détails.

La manière d'utiliser ce manuel varie suivant l'angle sous lequel vous envisagez l'utilisation de TELE-FORTH. En général, commencez par lire les pages suivantes sur les notations et les conventions employées tout au long du manuel.

Si vous désirez simplement vous amuser avec les graphiques et les sons, et que vous savez déjà vous en servir en BASIC, vous pouvez y aller tout de suite : les noms des fonctions sont les mêmes, l'ordre des arguments aussi, sauf qu'il doivent être écrits avant le nom de la fonction et avec des espaces entre chaque mot ; jetez quand même un coup d'oeil aux chapitres "Graphiques Haute Résolution", "Gestion Sonore" et "Gestion de l'Horloge", ils se pourrait qu'ils contiennent des surprises... et si vous voulez vous distraire tout en égayant l'espace visuel de votre lieu de vie, essayez le "Jeu de la Vie".

Si vous voulez vous mettre au FORTH pour de bon, procurez-vous un des ouvrages proposés en références bibliographiques, et au fur et à mesure de vos lectures, observez la construction, la longueur moyenne et la présentation des définitions, comment l'ensemble des concepts s'articulent entre eux ; étudiez à fond la démonstration du jeu de la vie : de nombreux concepts fondamentaux y sont abordés ; et surtout, expérimentez : c'est le meilleur moyen de se poser les bonnes questions.

Si vous êtes déjà expert, en FORTH ou d'autres langages, vous trouverez tous les détails de l'implémentation de TELE-FORTH, dans toute sa simplicité. L'auteur aurait préféré implémenter un FORTH83 mais les outils correspondants de méta-génération ne sont pas encore au niveau de ceux utilisés pour cette implémentation. La génération et l'emploi de modules relogeables devraient en intéresser plus d'un par sa simplicité. Si vous avez des doutes, ou peur qu'une coquille se soit subrepticement infiltrée à l'insu de l'auteur, utilisez le petit décompilateur pour lever le doute : l'EPROM est là.

Si vous voulez générer des applications TELEMATIC pour minitel, vous apprécierez sans doute la souplesse des outils proposés par TELE-FORTH pour le traitement des

"pages langage", ainsi que la possibilité de pouvoir monter vos bibliothèques d'outils.

Enfin, si vous êtes un fan de BASIC et que vous passez par là, comparez la rapidité de développement et la maintenabilité, donc la durée de vie des produits FORTH ; comparez aussi la rapidité d'exécution et la place mémoire requise pour une même application ; enfin, mais il faudra un peu plus de temps pour vous y faire, comparez la puissance d'expression, et s'il vous reste encore un peu de partialité, faites un jeu de la vie en BASIC pour comparer...

3. NOTATIONS ET CONVENTIONS

Chaque mot clé de TELE-FORTH est documenté de la manière suivante :

Son nom se trouve dans la marge.

Son entête de définition, donnant son créateur, et décrivant son intervention sur la pile (voir ci-dessous les conventions de notation pour l'emploi de la pile).

Sa définition complète (oui, c'est une première dans ce domaine !), tant pour les mots définis en FORTH que pour ceux définis en macro-assembleur.

Un descriptif de sa fonction.

Si possible un exemple d'utilisation, où ce que vous devez taper est suivi par ce que répond normalement FORTH.

Les piles sont représentées conventionnellement :

comme affiché par .S, la valeur au sommet de la pile étant le plus à droite ;

l'état de la pile en entrant est séparé par deux tirets -- de l'état de la pile en sortant ;

seules les valeurs au sommet de la pile utiles au fonctionnement sont mentionnées ;

s'il n'y a pas le même nombre de valeurs avant -- et après, c'est que le mot a consommé ou produit des valeurs sur la pile ;

pour certains mots, il peut y avoir deux configurations de retour différentes : elles sont séparées par un slash-deux-tirets ; exemple : (ad -- c 1 /-- 0) signifie que dans un cas ad est remplacé par c et 1, et dans l'autre par un 0 seulement ;

lorsque la pile de retour est utilisée, ce sont deux signes égale == qui sont employés ; exemple : >R (n -- | == n)

dans le cas des mots générateurs de familles, la première transformation de pile concerne la partie construction de

l'objet, la deuxième concerne son exécution ; exemple : CONSTANT (n -- | -- n)

Les valeurs sur la pile peuvent être de différent type :

b : booléen (nul=faux, sinon vrai) codé sur le bit 0 d'une valeur 16 bits
c : octet, généralement un code ascii sur les 7 bits de poids faible d'une valeur 16 bits
cnt : un octet de compte de chaîne, 5 ou 8 bits de poids faible d'une valeur 16 bits
n : entier signé, en complément à 2 sur 16 bits (-32768 à 32767)
u : entier non signé sur 16 bits (0 à 65535)
ad : une adresse 16 bits
d : un entier signé, en complément à 2 sur 32 bits (-2147483648 à 2147483647)
ud : un entier non signé, sur 32 bits (0 à 4294967296)

4. OPERATEURS DE MANIPULATION DES PILES

Comme tous les langages, le FORTH utilise des piles ; dans la plupart des langages les piles ne sont pas "visibles" par le programmeur : deux piles sont généralement gérées par le compilateur (comme en PASCAL ou en FORTRAN) ou par l'interpréteur (comme en BASIC ou en LISP) ; l'une permet de sauvegarder les adresses de retour d'appel de sous-programmes, l'autre sert à passer les arguments des sous-programmes et à stocker les résultats intermédiaires des calculs. En FORTH, le programmeur gère explicitement la pile "calcul", et peut utiliser (prudemment) la pile "retour".

DROP

```
CODE DROP ( n -- )
POP HERE 2- ! ;C
```

efface l'élément au sommet de la pile

```
SP! 10 20 .S DROP .S <RETURN>  (-- 10 20 ) (-- 10 ) ok.1
```

NIP

```
CODE NIP ( n2 n1 -- n1 | SWAP DROP )
1L LDA, 2L STA, 1H LDA, 2H STA, POP, ;C
```

efface le second élément de la pile

```
SP! 10 20 .S NIP .S <RETURN>  (-- 10 20 ) (-- 20 ) ok.1
```

2DROP

```
CODE 2DROP ( d -- | DROP DROP )
POPTWO HERE 2- ! ;C
```

efface les deux éléments du sommet de la pile (ou un entier double)

```
SP! 10 20 30 .S 2DROP .S <RETURN>  (-- 10 20 30 ) (-- 10 )
ok.1
SP! 123456. 98765. 2DROP D. <RETURN>  123456 ok.0
```

-DUP

```
CODE -DUP ( DUP IF DUP THEN )
1L LDA, 1H ORA, 0= NOT IF, ( DUP :) DEX, DEX, 2L LDA, 1L
STA, 2H LDA, 1H STA, THEN, NEXT, ;C
```

duplique le sommet de la pile s'il est non nul

```
SP! 10 -DUP .S <RETURN>  (-- 10 10 ) ok.2
```

```
      SP! 0 -DUP .S <RETURN>  (-- ) ok.0
```

DUP

```
      CODE DUP ( n -- n n )
      ' -DUP 6 + HERE 2- ! ;C
      duplique le sommet de la pile
      SP! 20 DUP .S <RETURN>  (-- 20 20 ) ok.2
```

2DUP

```
      CODE 2DUP ( d -- d d | OVER OVER )
      4  # LDY, BEGIN, DEX, 3L LDA, 1L STA, DEY, 0= UNTIL, NEXT,
      ;C
      duplique les deux éléments du sommet de  la  pile  (ou  un
      entier double)
      SP! 10 20 .S 2DUP .S <RETURN>  (-- 10 20 ) (-- 10 20 10 20
      ) ok.4
      SP! 98765. 2DUP D. D. <RETURN>  98765 98765 ok.0
```

OVER

```
      CODE OVER ( n2 n1 -- n2 n1 n2 )
      2L LDA, PHA, 2H LDA, PUSH, ;C
      duplique l'élément sous le sommet de la pile
      SP! 10 20 .S OVER .S <RETURN>  (-- 10 20 ) (-- 10 20 10  )
      ok.3
```

PICK

```
      CODE PICK ( k -- nk | 2* SP@ + @ )
      XSAVE  STX,  1L  LDA, .A ASL, CLC, XSAVE ADC, TAX, 1L LDA,
      PHA, 1H LDA, XSAVE LDX, PUT, ;C
      remplace le sommet de la pile (k) par une copie  du  kième
      élément de la pile
      SP! 4 3 2 1 .S 4 PICK .S <RETURN>  (-- 4 3 2 1 ) (-- 4 3 2
      1 4 ) ok.5
```

SWAP

```
      CODE SWAP ( n2 n1 -- n1 n2 )
      1L LDA, 2L LDY, 2L STA, 1L STY, 1H LDA, 2H LDY, 2H STA, 1H
      STY, NEXT, ;C
      échange les deux éléments du sommet de la pile
      SP!  20  10  .S  SWAP .S <RETURN>  (-- 20 10 ) (-- 10 20 )
      ok.2
```

2SWAP

```
      : 2SWAP ( d2 d1 -- d1 d2 )
      ROT >R ROT R> ;
      comme pour SWAP, mais pour des entiers doubles
```

ROT

```
      CODE ROT ( n3 n2 n1 -- n2 n1 n3 )
      3L LDA, PHA, 3H LDY, 2L LDA, 3L STA, 2H LDA,  3H  STA,  1L
      LDA, 2L STA, 1H LDA, 2H STA, TYA, PUT, ;C
      effectue une permutation circulaire sur les trois éléments
      du sommet de la pile (le troisième est rapatrié au  sommet
      de la pile)
      SP!  30 20 10 .S ROT .S <RETURN>  (-- 30 20 10 ) (-- 20 10
      30 ) ok.3
```

ROLL

```
      : ROLL ( nk+1 nk nk-1 ... n1 k -- nk+1 nk-1 ... n1 nk )
      DUP>R PICK SP@ DUP 2+ R> 2* CMOVE> DROP ;
      effectue une permutation circulaire sur les k  (la  valeur
```

au sommet de la pile) éléments du sommet de la pile (le kième élément est rapatrié au sommet de la pile)
SP! 4 3 2 1 .S 4 ROLL .S <RETURN> (-- 4 3 2 1) (-- 3 2 1 4) ok.4

>R

CODE >R (n -- | == n)
1H LDA, PHA, 1L LDA, PHA, POP, ;C
transporte le sommet de la pile calcul sur le sommet de la pile retour ; à utiliser avec précaution, uniquement à l'intérieur d'une définition, comme pile secondaire : mettre autant de R> (ou R>DROP) que de >R (ou DUP>R) pour ne pas modifier la pile retour ! Voir la définition de SWAP comme exemple

R>

R> (-- n | n ==)
DEX, DEX, PLA, 1L STA, PLA, 1H STA, NEXT, ;C
transporte le sommet de la pile retour sur le sommet de la pile calcul ; voir >R

R

CODE R (-- n | n == n)
' I HERE 2- ! ;C
copie le sommet de la pile retour sur le sommet de la pile calcul

DUP>R

CODE DUP>R (n -- n | == n)
1H LDA, PHA, 1L LDA, PHA, NEXT, ;C
copie le sommet de la pile calcul sur le sommet de la pile retour ; voir >R

R>DROP

CODE R>DROP (-- | n ==)
PLA, PLA, NEXT, ;C
efface le sommet de la pile retour ; voir >R

SP!

CODE SP! (... n1 -- | ==)
06 # LDY, UP)Y LDA, TAX, NEXT, ;C
vide la pile calcul ; utilisé pour initialisation ou nettoyage

SP@

CODE SP@ (-- sp)
TXA,
L: PUSH0A PHA, 0 # LDA, PUSH, ;C
retourne l'adresse du sommet de la pile ; voir la définition de PICK comme exemple d'utilisation

RP!

CODE RP! (-- | ... n1 ==)
X!, 08 # LDY, UP)Y LDA, TAX, TXS, X@, NEXT, ;C
vide la pile retour ; à n'utitiler que dans une boucle infinie (cas de l'interpréteur) sous peine de tuer le FORTH !

RP@

CODE RP@ (-- rp)
X!, TSX, TXA, PHA, 1 # LDA, X@, PUSH, ;C

retourne l'adresse du sommet de la pile retour

DEPTH

```
: DEPTH ( -- n )
SP@ S0 @ SWAP - 2/ ;
```

retourne le nombre de mots stockés dans la pile

.S

```
: .S ( -- )
."  (--  " DEPTH 0> IF SP@ 2- S0 @ 2- DO I ? -2 +LOOP THEN
." ) " ;
```

affiche le contenu de la pile avec les mêmes conventions que pour les commentaires de définitions ; la base courante est utilisée pour l'affichage

5. OPERATEURS D'ACCES MEMOIRE

FORTH travaille en standard sur des octets et des mots de 16 bits ; ont été rajoutées les extensions courantes en 32 bits, et des primitives d'accès au niveau du bit, moins courantes, mais souvent utiles, soit pour accéder à des ports d'entrées/sorties, soit pour gérer des structures de données compactes (bitmaps par exemple).

Rappel : le 6502 stocke les adresses 16 bits avec l'octet de poids faible à l'adresse basse et l'octet de poids fort à l'adresse haute (+1) ; FORTH fait de même pour les valeurs 16 bits, et stocke les valeurs 32 bits avec le mot de poids fort à l'adresse basse et le mot de poids faible à l'adresse haute (+2), afin d'avoir la même organisation en mémoire que dans la pile (qui croit vers les adresses basses, avec le mot de poids fort au sommet de la pile).

Note : sur un Z80, les adresses sont stockées en sens inverse, c'est-à-dire l'octet de poids fort à l'adresse basse et l'octet de poids faible à l'adresse haute ; éviter donc les "astuces" utilisant la convention particulière au 6502, pour assurer une bonne portabilité des programmes FORTH sur d'autres machines.

C!

```
CODE C! ( c adr -- )
2L LDA, 1L X) STA, POPTWO, ;C
```

primitive d'écriture à adr de l'octet c

```
65 PAD C! PAD C@ . <RETURN>  65 ok.0
```

C@

```
CODE C@ ( adr -- c )
1L X) LDA, 1L STA, 1H STY, NEXT, ;C
```

primitive de lecture de l'octet situé à adr (voir C!)

!

```
CODE ! ( n adr -- )
2L LDA, 1L X) STA, 1L INC, 0= IF, 1H INC, THEN, 2H LDA, 1L
X) STA, POPTWO, ;C
```

primitive d'écriture à adr du mot n (prononcer "store")

```
656 PAD ! PAD @ . <RETURN>  656 Ok.0
```

@

```
CODE @ ( adr -- n )
```

1L X) LDA, PHA, 1L INC, 0= IF, 1H INC, THEN, 1L X) LDA, PUT, ;C
primitive de lecture du mot situé à adr (prononcer "fetch", voir !)

2!

: 2! (d adr --)
DUP>R ! R> 2+ ! ;
écriture à adr du double mot d ; le mot de poids fort, au sommet de la pile, est stocké à adr, le mot de poids faible à adr+2
656565. PAD 2! PAD 2@ D. <RETURN> 656565 ok.0

2@

: 2@ (adr -- d)
DUP>R 2+ @ R> @ ;
lecture du double-mot stocké à adr (voir 2!)

+!

CODE +! (n adr --)
CLC, 1L X) LDA, 2L ADC, 1L X) STA, 1L INC, 0= IF, 1H INC, THEN, 1L X) LDA, 2H ADC, 1L X) STA, POPTWO, ;C
primitive d'incrémentation par n du mot situé à adr
5 PAD ! 3 PAD +! PAD ? <RETURN> 8 ok.0

COUNT

CODE COUNT (adr -- adr+1 c)
1L X) LDA, PHA, 1L INC, 0= IF, 1H INC, THEN, TYA, PUSH, ;C
primitive de lecture séquentielle de l'octet situé à adr ; habituellement utilisée pour retourner l'octet de longueur et l'adresse du premier caractère d'une chaîne de caractères (voir TYPE), cette primitive est très utile pour lire en séquence une suite d'octets (pour comprendre l'exemple, cherchez un peu...)
HEX 41034342. PAD 2! DECIMAL PAD COUNT TYPE <RETURN> ABC ok.0

WCOUNT

CODE WCOUNT (adr -- adr+2 n)
1L X) LDA, PHA, 1L INC, 0= IF, 1H INC, THEN, 1L X) LDA, 1L INC, 0= IF, 1H INC, THEN, PUSH, ;C
primitive de lecture séquentielle du mot situé à adr ; rarement intégrée au FORTH, cette primitive n'en est pas moins souvent utile.
HEX 12345678. PAD 2! PAD WCOUNT . ? DECIMAL <RETURN> 1234 5678 ok.0

TOGGLE

CODE TOGGLE (adr c -- | OVER C@ XOR SWAP C!)
1L LDA, 2L X) EOR, 2L X) STA, POPTWO, ;C
primitive d'inversion de bits : chaque bit à 1 du masque c inverse le bit correspondant de l'octet situé à adr.
HEX 20 PAD C! PAD 30 TOGGLE PAD C@ . DECIMAL <RETURN> 10 Ok.0

CSET

CODE CSET (adr c -- | OVER C@ OR SWAP C!)
1L LDA, 2L X) ORA, 2L X) STA, POPTWO, ;C
primitive d'écriture à 1 de bits : chaque bit à 1 du masque c force à 1 le bit correspondant de l'octet situé à adr.

HEX 55 PAD C! PAD 30 CSET PAD C@ . DECIMAL <RETURN> 75 ok.0

CRST

CODE CRST (adr c -- | NOT OVER C@ AND SWAP C!)
1L LDA, 0FF # EOR, 2L X) AND, 2L X) STA, POPTWO, ;C
primitive d'écriture à 0 de bits : chaque bit à 1 du masque c force à 0 le bit correspondant de l'octet situé à adr.
HEX 55 PAD C! PAD 30 CRST PAD C@ . DECIMAL <RETURN> 45 ok.0

CTST

CODE CTST (adr c -- f | SWAP C@ AND 0= 0=)
1L LDA, 2L X) AND, 2H STY, 0= NOT IF, INY, THEN, 2L STY, POP, ;C
primitive de test de bits : si aucun des bits à 1 du masque c n'est à 1 dans l'octet situé à adr, le booléen retourné est faux ; inversement, si un des bits à 1 du masque c est à 1 dans l'octet situé à adr, le booléen retourné est vrai.
HEX 55 PAD C! PAD 34 CTST . PAD 62 CTST . DECIMAL <RETURN> 1 0 ok.0

BMAP

CODE BMAP (adr n -- adr' c | 0 8 U/ ROT + SWAP 1 0 ROT 2** DROP)
1L LDA, 7 # AND, (A = numéro du bit, puis adr' = adr + n/8) 3 # LDY, BEGIN, 1H LSR, 1L ROR, DEY, 0= UNTIL, TAY, CLC, 1L LDA, 2L ADC, 2L STA, 1H LDA, 2H ADC, 2H STA, BMAP: ,Y LDA, 1L STA, 0 # LDA, 1H STA, NEXT,
L: BMAP: 01 C, 02 C, 04 C, 08 C, 10 C, 20 C, 40 C, 80 C, ;C
primitive pour traitement de bitmaps : adr est l'adresse de base de la bitmap, n est le numéro de bit de la bitmap à traiter ; retourne l'adresse de l'octet correspondant et un masque ayant pour seul bit à 1 celui correspondant au bit à traiter ; s'utilise avec TOGGLE, CSET, CRST et CTST (dans l'exemple, PAD est en 12E0 hexa)
HEX 1000 PAD ! PAD 0C BMAP .S CTST . DECIMAL <RETURN> (-- 12E1 10) 1 ok.0

CMOVE

CODE CMOVE (srceAdr destAdr byteCnt --)
3 # LDA, SETUP, BEGIN, BEGIN, N 4 +)Y LDA, N CPY, 0= IF, N 1+ DEC, 0< IF, NEXT, THEN, THEN, N 2 +)Y STA, INY, 0= UNTIL, N 5 + INC, N 3 + INC, AGAIN, ;C
primitive de mouvement de bloc mémoire par adresses croissantes : byteCnt octets sont transférés de srceAdr à destAdr ; le premier octet est transféré de srceAdr à destAdr, le dernier octet est transféré de srceAdr+byteCnt-1 à destAdr+byteCnt-1. ATTENTION : utiliser CMOVE> si destAdr est entre srceAdr et srceAdr+byteCnt-1 ! (voir MOVE)

CMOVE>

CODE CMOVE> (srceAdr destAdr byteCnt --)
3 # LDA, SETUP, N LDY, 0= IF, N 1+ LDA, 0= NOT IF, 2SWAP THEN, CLC, N 5 + LDA, N 1+ ADC, N 5 + STA, CLC, N 3 + LDA, N 1+ ADC, N 3 + STA, CM> JMP, BEGIN, BEGIN, DEY, N 4 +)Y LDA, N 2+)Y STA,

L: CM> 0 # CPY, 0= UNTIL, N 5 + DEC, N 3 + DEC, N 1 + DEC, 0< UNTIL, THEN, NEXT, ;C
primitive de mouvement de bloc mémoire par adresses décroissantes : byteCnt octets sont transférés de srceAdr à destAdr ; le premier octet est transféré de srceAdr+byteCnt-1 à destAdr+byteCnt-1, le dernier octet est transféré de srceAdr à destAdr. ATTENTION : utiliser CMOVE si srceAdr est entre destAdr et destAdr+byteCnt-1 ! (voir MOVE)

MOVE
: MOVE (srceAdr destAdr byteCnt --)
>R 2DUP < IF R> CMOVE> ELSE R> CMOVE THEN ;
MOVE transfère (sans risque de repliement) un bloc mémoire de byteCnt octets, de srceAdr (adresse de début du bloc) à destAdr, à utiliser de préférence chaque fois qu'il y a recouvrement des zones source et destination.

FILL
: FILL (adr cnt c --)
OVER 0> IF SWAP >R OVER C! DUP 1+ R> 1- CMOVE ELSE DROP 2DROP THEN ;
FILL remplace les cnt octets situés à adr avec l'octet c (noter que l'emploi de CMOVE avec repliement sur le premier octet).

ERASE
: ERASE (adr cnt --)
0 FILL ;
ERASE met à zéro les cnt octets situés à adr.

BLANKS
: BLANKS (adr cnt --)
BL FILL ;
BLANKS met à BL (l'espace, code ASCII 32) les cnt octets situés à adr.

6. OPERATEURS LOGIQUES

Ils travaillent bit à bit sur des entiers de 16 bits, soit pour des opérations de masquage, soit pour du calcul sur des valeurs logiques retournées par les opérateurs de comparaison (le travail utile se fait alors sur le bit de poids faible).

Pratique : la dualité entre le ET et le OU permet de simplifier certaines expressions logiques (donc d'améliorer leur lisibilité) :
NONa ET NONb = NON(a OU b) : NOT SWAP NOT AND se simplifie en : OR NOT(a b -- f)
NONa OU NONb = NON(a ET b) : NOT SWAP NOT OR se simplifie en : AND NOT (a b -- f)

AND
CODE AND (n2 n1 -- n)
1L LDA, 2L AND, PHA, 1H LDA, 2H AND, INX, INX, PUT, ;C
ET logique ; si n1 est le masque, ses bits à 0 forceront à 0 ceux de n, et ses bits à 1 donneront à ceux de n la valeur de ceux de n2
HEX EA39 FF0 AND . DECIMAL <RETURN> A30 ok.0

OR

CODE OR (n2 n1 -- n)
1L LDA, 2L ORA, PHA, 1H LDA, 2H ORA, INX, INX, PUT, ;C
OU logique inclusif ; si n1 est le masque, ses bits à 1 forceront à 1 ceux de n, et ses bits à 0 donneront à ceux de n la valeur de ceux de n2
HEX EA39 FF0 OR . DECIMAL <RETURN> EFF9 ok.0

XOR

CODE XOR (n2 n1 -- n | OVER NOT OVER AND >R NOT AND R> OR)
1L LDA, 2L EOR, PHA, 1H LDA, 2H EOR, INX, INX, PUT, ;C
OU logique exclusif ; si n1 est le masque, ses bits à 1 donneront à ceux de n la valeur opposée de ceux de n2, et ses bits à 0 donneront à ceux n la valeur de ceux de n2
HEX EA39 FF0 XOR . DECIMAL <RETURN> E5C9 ok.0

NOT

CODE NOT (n -- n' | -1 XOR)
DEY, TYA, 1L EOR, 1L STA, TYA, 1H EOR, 1H STA, NEXT, ;C
NON logique ; c'est un complément à 1 sur 16 bits, à ne pas confondre avec 0= qui transforme une valeur logique fausse (nulle) en vraie (non nulle) ou inversement
HEX EA39 NOT . DECIMAL <RETURN> 15C6 ok.0

OFF

: OFF (adr --)
0 SWAP ! ;
force à 0 le mot situé à adr ; à utiliser pour mettre une variable logique à l'état faux.
PAD OFF PAD ? <RETURN> 0 ok.0

ON

: ON (adr --)
1 SWAP ! ;
primitive de forçage à 1 du mot situé à adr ; à utiliser pour mettre une variable logique à l'état vrai.
PAD ON PAD ? <RETURN> 1 ok.0

7. OPERATEURS DE COMPARAISON

Les valeurs logiques en FORTH sont représentées par des entiers 16 bits, de valeur nulle pour la valeur logique fausse, non nulle (normalement 1 en Fig-FORTH) pour la valeur logique vraie.

0=

CODE 0= (n -- b)
1H LDA, 1H STY, 1L ORA, 0= IF, INY, THEN, 1L STY, NEXT, ;C
f est vrai si n est nul ; également utilisé pour la négation logique
SP! 0 0= . <RETURN> 1 ok.0
10 0= . <RETURN> 0 ok.0

0<

CODE 0< (n -- b)
1H ASL, 1H STY, TYA, .A ROL, 1L STA, NEXT, ;C
f est vrai si n est strictement négatif
-10 0< . <RETURN> 1 ok.0

```
10 0< . <RETURN>  0 ok.0
0 0< . <RETURN>  0 ok.0
```

0>

```
CODE 0> ( n -- b | DUP>R 0< R> 0= OR NOT )
1H LDA, 1H STY, 0< NOT IF, 1L ORA, 0= NOT IF,  INY,  THEN,
THEN, 1L STY, NEXT, ;C
f est vrai si n est strictement positif
10 0> . <RETURN>  1 ok.0
-10 0> . <RETURN>  0 ok.0
0 0> . <RETURN>  0 ok.0
```

=

```
: = ( n2 n1 -- b )
- 0= ;
f est vrai si n2 égale n1
10 10 = . <RETURN>  1 ok.0
-10 10 = . <RETURN>  0 ok.0
```

<

```
CODE < ( n2 n1 -- b )
SEC,  2L  LDA, 1L SBC, 2H LDA, 1H SBC, 2H STY, VS IF, 80 #
EOR, THEN, 0< IF, INY, THEN, 2L STY, POP, ;C
f est vrai si n2 est strictement inférieur à n1
-10 10 < . <RETURN>  1 ok.0
10 -10 < . <RETURN>  0 ok.0
10 10 < . <RETURN>  0 ok.0
```

>

```
: > ( n2 n1 -- b )
SWAP < ;
f est vrai si n2 est strictement supérieur à n1
10 -10 > . <RETURN>  1 ok.0
-10 10 > . <RETURN>  0 ok.0
10 10 > . <RETURN>  0 ok.0
```

U<

```
: U< ( u2 u1 -- b )
0 SWAP 0 DMINUS D+ NIP 0< ;
f est vrai si u2 est  strictement  inférieur  à  u1  (pour
comparaison d'adresses)
HEX FFFE FFFF U< . DECIMAL <RETURN>  1 ok.0
```

S=

```
CODE S= ( ad2 ad1 cnt -- b )
2 SETUP, BEGIN, N 2+ LDA, N 4 + EOR, .A ASL, ( sans bit7 )
0= IF, INY, N CPY, 2SWAP 0= UNTIL, 1 # LDA, PUSH, THEN,  0
# LDA, PUSH, ;C
primitive  de  comparaison  de deux chaînes de caractères,
l'une située à ad2 l'autre à ad1 ; la comparaison se  fait
sur cnt caractères ; retourne une valeur logique, vraie si
les deux chaines sont identiques
```

8. OPERATEURS ARITHMETIQUES

L'arithmétique FORTH de base se fait sur des entiers signés en complément à 2, de 16 bits avec certaines extensions en 32 bits. Cette arithmétique est largement suffisante dans la plupart des cas (calculs en entiers ou en virgule fixe), et présente l'avantage d'être simple et

rapide ; si des calculs plus sophistiqués sont nécessaires, on pourra avoir recours à une extension "calcul en virgule flottante" (supportée par la primitive 2**).

S->D

```
CODE S->D ( n -- d )
1H LDA, 0< IF, DEY, THEN, TYA, PHA, PUSH JMP, ;C
```

conversion (extension de signe) de 16 à 32 bits (utiliser DROP pour convertir de 32 en 16 bits)

```
SP! -5 .S S->D .S <RETURN>  (-- -5 ) (-- -5 -1 ) ok.2
```

MINUS

```
CODE MINUS ( n -- -n )
SEC, TYA, 1L SBC, 1L STA, TYA, 1H SBC, 1H STA, NEXT, ;C
```

change le signe du sommet de la pile

```
12 MINUS . <RETURN>  -12 ok.0
-7 MINUS . <RETURN>  7 ok.0
```

DMINUS

```
CODE DMINUS ( d -- -d )
SEC,  TYA, 2L SBC, 2L STA, TYA, 2H SBC, 2H STA, ' MINUS 1+
JMP, ;C
```

change le signe du sommet de la pile (32 bits)

```
-123456 . DMINUS D. <RETURN>  123456 ok.0
```

D+-

```
: D+- ( d2 n1 -- d )
0< IF DMINUS THEN ;
```

donne à d la valeur de d2 (entiers 32 bits) avec un signe opposé si n1 est négatif

```
-123456. 7 D+- D. <RETURN>  -123456 ok.0
```

+-

```
: +- ( n2 n1 -- n )
0< IF MINUS THEN ;
```

donne à n la valeur de n2 avec un signe opposé si n1 est négatif

```
SP! 12 -5 +- . <RETURN>  -12 ok.0
```

DABS

```
: DABS ( d -- d' )
DUP D+- ;
```

remplace le sommet de la pile par sa valeur absolue (32 bits)

```
-123456. DABS D. <RETURN>  123456 ok.0
```

ABS

```
: ABS ( n -- n' )
DUP +- ;
```

remplace le sommet de la pile par sa valeur absolue

```
-5 ABS . <RETURN>  5 ok.0
12 ABS . <RETURN>  12 ok.0
```

MAX

```
: MAX ( n2 n1 -- n )
2DUP < IF NIP ;S THEN DROP ;
```

ne laisse sur la pile que le plus grand des deux nombres n1 ou n2.

```
12 7 MAX . <RETURN>  12 ok.0
```

MIN

```
: MIN ( n2 n1 -- n )
2DUP < IF DROP ;S THEN NIP ;
ne laisse sur la pile que le plus petit des  deux  nombres
n1 ou n2.
12 7 MIN . <RETURN>  7 ok.0
```

D+

```
CODE D+ ( d2 d1 -- d | d=d2+d1 )
CLC,  4L  LDA,  2L ADC, 4L STA, 4H LDA, 2H ADC, 4H STA, 3L
LDA, 1L ADC, 3L STA, 3H LDA, 1H ADC, 3H STA, POPTWO, ;C
addition signée 32 bits  (la  soustraction  s'obtient  par
DMINUS D+)
123456. 654321. D+ D. <RETURN>  777777 ok.0
```

+

```
CODE + ( n2 n1 -- n | n=n2+n1 )
CLC,  2L LDA, 1L ADC, 2L STA, 2H LDA, 1H ADC, 2H STA, POP,
;C
addition signée 16 bits
12 5 + . <RETURN>  17 ok.0
-12 -5 + . <RETURN>  -17 ok.0
```

-

```
CODE - ( n2 n1 -- n | n=n2-n1 )
SEC, 2L LDA, 1L SBC, 2L STA, 2H LDA, 1H SBC, 2H STA,  POP,
;C
soustraction signée 16 bits
12 5 - . <RETURN>  7 ok.0
-12 -5 - . <RETURN>  -7 ok.0
```

1+

```
CODE 1+ ( n -- n' | n'=n+1 )
1L INC, 0= IF, 1H INC, THEN, NEXT, ;C
incrémentation 16 bits
5 1+ . <RETURN>  6 ok.0
```

1-

```
CODE 1- ( n -- n' | n'=n-1 )
1L LDA, 0= IF, 1H DEC, THEN, 1L DEC, NEXT, ;C
décrémentation 16 bits
5 1- . <RETURN>  4 ok.0
```

2+

```
: 2+ ( n -- n' | n'=n+2 )
1+ 1+ ;
incrémentation double
5 2+ . <RETURN>  7 ok.0
```

2-

```
: 2- ( n -- n' | n'=n-2 )
1- 1- ;
décrémentation double (voir CFA par exemple)
5 2- . <RETURN>  3 ok.0
```

U*

```
CODE U* ( u2 u1 -- ud | ud=u2*u1 )
2L LDA, N STA, 2L STY, 2H LDA, N 1+ STA, 2H STY, 16 # LDY,
BEGIN, 2L ASL, 2H ROL, 1L ROL, 1H ROL, CS IF, CLC, N  LDA,
2L  ADC,  2L STA, N 1+ LDA, 2H ADC, 2H STA, CS IF, 1L INC,
THEN, THEN, DEY, 0= UNTIL, NEXT, ;C
```

primitive de multiplication, non signée, 16 par 16 résultat sur 32 bits
1000 2000 U* D. <RETURN> 2000000 ok.0

M*

```
: M* ( n2 n1 -- d | d=n2*n1 )
2DUP XOR >R ABS SWAP ABS U* R> D+- ;
```

produit mixte, multiplication signée 16 par 16 résultat sur 32 bits
-1000 2000 M* D. <RETURN> -2000000 ok.0

*

```
: * ( n2 n1 -- n | n=n2*n1 )
M* DROP ;
```

multiplication signée 16 bits, résultat sur 16 bits (la plus couramment employée)
25 -25 * . <RETURN> -625 ok.0

U/

```
CODE U/ ( ud u -- uR uQ | ud=u*uQ+uR )
3L LDA, 2L LDY, 3L STY, .A ASL, 2L STA, 3H LDA, 2H LDY, 3H
STY, .A ROL, 2H STA, 16 # LDA, N STA, BEGIN, 3L ROL, 3H
ROL, SEC, 3L LDA, 1L SBC, TAY, 3H LDA, 1H SBC, CS IF, 3L
STY, 3H STA, THEN, 2L ROL, 2H ROL, N DEC, 0= UNTIL, POP,
;C
```

primitive de division entière avec reste, non signée, dividende ud 32 bits par diviseur u 16 bits, résultats reste uR et quotient uQ sur 16 bits non signés
100003. 5 U/ . . <RETURN> 20000 3 ok.0

M/MOD

```
: M/MOD ( ud u -- uR udQ | ud=u*udQ+uR )
>R 0 R U/ R> SWAP >R U/ R> ;
```

division entière mixte avec reste, non signée, dividende ud 32 bits par diviseur u 16 bits, résultats reste uR sur 16 bits et quotient uQ sur 32 bits, non signés
400003. 5 M/MOD D. . <RETURN> 80000 3 ok.0

M/

```
: M/ ( d n -- nR nQ | d=n*nQ+nR )
OVER >R >R DABS R ABS U/ R> R XOR +- SWAP R> +- SWAP ;
```

division entière mixte avec reste, signée, dividende d 32 bits par diviseur n 16 bits, résultats reste nR et quotient nQ sur 16 bits, signés (reste du signe du dividende)
-100003 . -5 M/ . . <RETURN> 20000 -3 ok.0

/MOD

```
: /MOD ( n2 n1 -- nR nQ | n2=n1*nQ+nR )
>R S->D R> M/ ;
```

division entière avec reste, signée, dividende n2 16 bits par diviseur n1 16 bits, résultats reste nR et quotient nQ sur 16 bits, signés (reste du signe du dividende)
-103 -5 /MOD . . <RETURN> 20 -3 ok.0

MOD

```
: MOD ( n2 n1 -- nR )
/MOD DROP ;
```

"modulo", reste 16 bits de la division entière signée 16 bits, dividende n2 par diviseur n1 (le reste est du signe du dividende)

```
      -103 -5 MOD . <RETURN>  -3 ok.0

/
      : / ( n2 n1 -- nQ )
      /MOD NIP ;
      quotient 16 bits de la division entière signée 16 bits,
      dividende n2 par diviseur n1 (opérateur de division le
      plus couramment employé)
      -103 -5 / . <RETURN>  20 ok.0

*/MOD
      : */MOD ( n3 n2 n1 -- nR nQ | n3*n2=n1*nQ+nR )
      >R M* R> M/ ;
      "règle de trois", multiplication signée de n3 16 bits par
      n2 16 bits avec résultat intermédiaire sur 32 bits, suivie
      d'une division entière par n1, résultat reste nR et
      quotient nQ sur 16 bits signés (reste du signe du
      dividende n3*n2)
      1027 25 100 */MOD . . <RETURN>  256 3 ok.0

*/
      : */ ( n3 n2 n1 -- nQ )
      */MOD NIP ;
      "règle de trois" sans reste, multiplication signée de n3
      par n2 avec résultat intermédiaire sur 32 bits, suivie
      d'une division entière par n1, résultat quotient entier
      seulement (opérateur souvent employé pour les
      pourcentages)
      1027 25 100 */ . <RETURN>  256 ok.0 (25% de 1027)

2/
      CODE 2/ ( n -- n' | n'=n/2 )
      TYA, 1H CMP, 1H ROR, 1L ROR, NEXT, ;C
      division par deux (décalage arithmétique à droite)
      -1025 2/ . <RETURN>  512 ok.0
      (mais ATTENTION : ) -1 2/ . <RETURN>  -1 ok.0

2*
      CODE 2* ( n -- n' | n'=n*2 )
      1L ASL, 1H ROL, NEXT, ;C
      multiplication par deux (décalage arithmétique à gauche)
      1024 2* . <RETURN>  2048 ok.0

2**
      CODE 2** ( d c -- d' | d'=d*2^c )
      1L LDY, 0= NOT IF, 0< IF, TYA, BEGIN, 2H LDA, .A ASL, 2H
      ROR, 2L ROR, 3H ROR, 3L ROR, INY, 0= UNTIL, ( décalages à
      droite ) ELSE, BEGIN, 3L ASL, 3H ROL, 2L ROL, 2H ROL, DEY,
      0= UNTIL, ( décalages à gauche ) THEN, THEN, POP, ;C
      primitive de décalages arithmétiques multiples (pour
      extensions calcul en flottant) ; le nombre de décalages,
      c, peut être positif ou négatif ; il n'y a aucun contrôle
      de débordement (ni overflow ni underflow ; underflow
      positif donne 0, négatif donne -1)
      2. 3 2** D. <RETURN>  16 ok.0

0
      0 CONSTANT 0 ( -- 0 )
1
      1 CONSTANT 1 ( -- 1 )
2
```

2 CONSTANT 2 (-- 2)

BL

020 CONSTANT BL (-- 020)
0, 1, 2 et BL sont définies comme constantes parce que fréquemment utilisées

9. CONVERSION NUMERIOUE

FORTH sait lire et écrire les nombres dans n'importe quelle base numérique : il suffit de changer le contenu de la variable BASE ; le signe d'invite affiche en permanence la base courante (et le nombre de valeurs dans la pile) :
- ok.0 base décimale (caractère point), pile vide
- ok#2 base hexadécimale (caractère dièse), 2 valeurs dans la pile
- ok%5 base binaire (caractère pourcent), 5 valeurs dans la pile
- ok*3 autre base (caractère étoile), 3 valeurs dans la pile

En lecture, un nombre peut contenir un signe négatif "-" initial, et/ou un point décimal, dont la position par rapport au dernier digit est mémorisée dans la variable DPL (par exemple 2 pour 1234.56) ; s'il n'y a pas de point décimal, DPL vaudra -1, et le nombre sera tronqué aux 16 bits de poids faible après conversion sur 32 bits (voir INTERPRET).
En écriture, la chaîne résultat de la conversion est formée "à reculons" (pointeur HLD) dans un tampon, habituellement le "PAD" placé 68 octets après la fin du dictionnaire ; les nombres sur 16 bits sont étendus sur 32 bits avant conversion.

DECIMAL

: DECIMAL (--)
0A BASE ! ;
la prochaine conversion numérique se fera en décimal (base dix).

HEX

: HEX (--)
010 BASE ! ;
la prochaine conversion numérique se fera en hexadécimal (base seize).

BIN

: BIN (--)
2 BASE ! ;
la prochaine conversion numérique se fera en base 2.

DIGIT

CODE DIGIT (c base -- n 1 /-- 0)
SEC, 2L LDA, ASCII 0 # SBC, 0< NOT IF, (ascii >= '0') 0A # CMP, 0< NOT IF, (ascii > '9') SEC, 7 # SBC, 0A # CMP, 0< NOT IF, (ascii >= 'A') 2SWAP THEN, (ascii in [0 ... 9] u [A ... .] => accu = digit) 1L CMP, 0< IF, 2L STA, 1 # LDA, PHA, TYA, PUT, THEN, THEN, THEN, (ascii incorrect) TYA, PHA, INX, INX, PUT, ;C
primitive de conversion d'un caractère de code ASCII c en digit : le caractère doit être un chiffre ou une lettre (A, B, C, ...), strictement inférieur à la base (donnée au

sommet de la pile), dans ce cas le digit n correspondant est retourné sur la pile avec une valeur booléenne vraie (1), sinon seule une valeur booléenne fausse (0) est retournée.
DECIMAL ASCII C 16 DIGIT . . <RETURN> 1 12 ok.0
ASCII H 16 DIGIT . <RETURN> 0 ok.0

(NUMBER)
: (NUMBER) (ud adr -- ud' adr')
BEGIN 1+ DUP >R C@ BASE @ DIGIT WHILE SWAP BASE @ U* DROP ROT BASE @ U* D+ DPL @ 1+ IF 1 DPL +! THEN R> REPEAT R> ;
utilisé par NUMBER pour convertir une chaîne de caractères située à adr+1 (octet de longueur ou caractère "-" ou "." à adr) en entier 32 bits non signé, jusqu'à trouver un caractère non compatible avec la base courante ; le résultat est "concaténé à droite" de d (donné au sommet de la pile) pour donner d' ; afin de retenir la position d'un éventuel point décimal (sans lequel INTERPRET tronquera le résultat final à 16 bits) la variable DPL est incrémentée à chaque digit si elle est différente de -1 ; voir INTERPRET.

INUMBER
: INUMBER (adr -- d)
0 0 ROT DUP 1+ C@ ASCII - = DUP>R + (signe) -1 BEGIN DPL ! (NUMBER) DUP C@ BL - WHILE DUP C@ ASCII . - 5 ?ERROR 0 REPEAT DROP R> IF DMINUS THEN ;
c'est la valeur à l'initialisation du vecteur NUMBER utilisé par INTERPRET pour les conversions numériques en entrée (son équivalent en virgule flottante est FNUMBER - Cf la remarque du premier paragraphe sur la librairie de calcul en virgule flottante) ; convertit une chaîne de caractères située à [adr+1] (octet de longueur à adr) en entier 32 bits signé ; (NUMBER) est appelé avec la variable DPL initialisée à -1 ; le premier caractère incompatible avec la base courante, trouvé par (NUMBER), arrête la conversion si c'est un espace ; si c'est un point décimal, (NUMBER) est appelé à nouveau avec DPL à zéro pour lancer le comptage de la position du point décimal ; si c'est un autre caractère, l'erreur "inconnu dans ce vocabulaire" est générée pour signaler l'échec de la conversion, tentée à la suite de l'échec de la recherche lexicale ; voir INTERPRET et ?ERROR.

NUMBER
DEFER ?NUMBER ' INUMBER IS NUMBER
voir INUMBER

#IN
: #IN (-- n)
BLK @ >R IN @ >R 0 BLK ! QUERY BL WORD HERE NUMBER DROP R> IN ! R> BLK ! ;
attend une chaîne de caractères au clavier terminée par la touche <RETURN>, puis tente de la convertir en un nombre 16 bits ; le flot d'entrée n'est pas perturbé (sauf dans le cas d'un flot d'entrée au clavier, dont le nombre de caractères déjà traités est inférieur au nombre de caractères saisis par #IN ; dans l'exemple, DECIMAL est suffisant)
DECIMAL #IN H. <RETURN> 128 <RETURN> 80 ok.0

PAD

```
: PAD ( -- adr )
HERE 0 44 + ;
```

retourne l'adresse de fin du tampon de formatage pour conversion en écriture.

<#

```
: <# ( -- )
PAD HLD ! ;
```

initialise la conversion en écriture : HLD pointe sur PAD.

#>

```
: #> ( d -- adr cnt )
2DROP HLD @ PAD OVER - ;
```

termine la conversion en écriture : le reste à convertir est remplacé par l'adresse et le compte en octets de la chaîne formatée.

HOLD

```
: HOLD ( c -- )
-1 HLD +! HLD @ C! ;
```

concatène en tête de la chaîne en cours de formatage le caractère de code ASCII c.

#

```
: # ( d -- d' )
BASE @ M/MOD ROT 9 OVER < IF 7 + THEN ASCII 0 + HOLD ;
```

concatène en tête de la chaîne en cours de conversion le caractère correspondant au digit de poids faible (dans la base courante) du nombre d (32 bits) ; le nombre d' laissé sur la pile est prêt pour la conversion du digit suivant.

#S

```
: #S ( d -- 0000 )
BEGIN # 2DUP OR 0= UNTIL ;
```

convertit le nombre d dans la base courante (par appels successifs à # jusqu'à un résultat nul).

SIGN

```
: SIGN ( n d -- d )
ROT 0< IF ASCII - HOLD THEN ;
```

concatène en tête de la chaîne en cours de formatage le caractère - si n est négatif.

D.R

```
: D.R ( d c -- )
>R SWAP OVER DABS <# #S SIGN #> R> OVER - SPACES TYPE ;
```

affiche le nombre d (32 bits) dans la base courante, cadré à droite dans un champ de c caractères si la longueur de la chaîne formatée ne dépasse pas n (sinon toute la chaîne est affichée).

```
." /" 12345678. 10 D.R ." /" <RETURN>  /  12345678/ ok.0
```

D.

```
: D. ( d -- )
0 D.R SPACE ;
```

affiche le nombre d dans la base courante, suivi d'un espace (format "libre").

```
." /" 12345678 . D. ." /" <RETURN>  /12345678 / ok.0
```

U.

```
: U. ( u -- )
0 D. ;
```
affiche le nombre u dans la base courante (utilisé pour les nombres 16 bits non signés, qui sinon seraient compris négatifs entre 32768 et 65535).
```
50000 DUP . U. <RETURN>  -15536 50000 ok.0
```

H.

```
: H. ( u -- )
BASE @ HEX SWAP U. BASE ! ;
```
affiche le nombre u en hexadécimal ; utilisé surtout pour les adresses mémoire.
```
DECIMAL 32768 H. <RETURN>  8000 ok.0
```

.R

```
: .R ( n c -- )
>R S->D R> D.R ;
```
homologue 16 bits de D.R
```
." /" 1234 6 .R ." /" <RETURN>  /  1234/ ok.0
```

.

```
: . ( n -- )
S->D D. ;
```
homologue 16 bits de D. (prononcer "dot", "point" ou "print")
```
." /" 1234 . ." /" <RETURN>  /1234 / ok.0
```

?

```
: ? ( adr -- )
@ . ;
```
imprime le mot 16 bits situé à adr ; utile surtout en intéractif.
```
123 PAD ! PAD ? <RETURN>  123 ok.0
```

10. LA MACHINE VIRTUELLE FORTH

FORTH est un des premiers langages à avoir utilisé le concept de "machine virtuelle". Une machine virtuelle, c'est un modèle de fonctionnement sans support physique immédiat ; l'intérêt d'un tel concept est de pouvoir décrire le fonctionnement du langage en terme d'instructions de la machine virtuelle unique : vous pourrez constater à travers ce manuel que FORTH est en majorité défini en FORTH ! Seules les "primitives" de base, proches des instructions communes à toutes les "machines réelles" (le 6502 pour le TELESTRAT), sont à implémenter en code machine. D'où la grande facilité de portage du FORTH et sa disponibilité sur un grand nombre de machines.

La machine virtuelle FORTH est constituée de 5 registres et d'une unité arithmétique et logique travaillant sur le sommet de la pile de données.
- RP est le pointeur d'instructions
- W est le registre de décodage
- RP est le pointeur de pile d'adresses de retours
- SP est le pointeur de pile données
- UP est le pointeur de zone utilisateur

La machine virtuelle FORTH manipule principalement des

adresses ; si l'on considère les instructions de la machine réelle comme les microinstructions de la machine virtuelle, alors chaque instruction de la machine virtuelle est traitée comme un pointeur à une structure mémoire dont le premier mot est l'adresse d'un micro-programme ; pour cette raison, Ce premier mot est appelé CFA ("Code Field Address") car c'est un champ qui contient l'adresse du micro-code à exécuter, le mot suivant est appelé PFA ("Parameter Field Address") car c'est là que sont stockés tous les paramètres de la structure.

Pratiquement, l'automate micro-programmé de la machine virtuelle (exécuté à la fin de chaque micro-programme) réalise les opérations suivantes :
1° W:=(IP)+
charger W avec l'instruction pointée par IP et incrémenter IP
2° mip:=(W)
charger le pointeur de micro-instructions avec l'adresse pointée par W

Pour le 6502, IP W et UP sont en mémoire en page 0, RP est dans le registre S (pointeur de pile 6502 en page 1), et SP est dans le registre d'index X.

NOOP

CODE NOOP (--)
L: NEXT \ point d'arrivée de la macro de saut NEXT,
1 # LDY, IP)Y LDA, W 1+ STA, DEY, IP)Y LDA, W STA, CLC, IP LDA, 2 # ADC, IP STA, CS IF, IP 1+ INC, THEN, W 1- JMP, ;C
réalise l'opération 1 de l'automate ; l'opération 2 est exécutée en RAM (n'oublions pas qu'il s'agit d'un FORTH en ROM), au moyen d'une simple instruction de saut indirect placée à l'adresse W-1.
Notons au passage que le registre Y est laissé nul par l'automate, donc sera toujours nul à l'entrée de toutes les primitives, et que W pointe toujours sur le CFA.

Dans le cas d'une primitive FORTH, le PFA contient généralement le code machine de la primitive, et le CFA pointe donc sur le PFA, juste deux adresses plus loin ; toute primitive se terminera comme il se doit en sautant à NEXT.

Dans le cas d'une secondaire, appelée ainsi par opposition à "primitive", le PFA contient des instructions virtuelles, que l'automate devra exécuter en séquence ; le CFA pointe alors sur du code qui fera l'équivalent d'un appel de sous-programme : (voir la partie ;CODE de :)
1° -(RP):=IP
empilage du pointeur d'instructions sur la pile de retour
2° IP:=W+2
initialiser le pointeur d'instruction pointant sur le PFA
3° NEXT sauter à l'automate

La dernière instruction virtuelle d'une secondaire devra faire l'équivalent d'un retour de sous-programme :
1° IP:=(RP)+
dépilage du pointeur d'instruction en attente sur la pile de retour

2° NEXT sauter à l'automate

;S

CODE ;S (--)
PLA, IP STA, PLA, IP 1+ STA, NEXT, ;C
c'est le "return" FORTH compilé par "point-virgule" à la fin de chaque définition

Pour rompre le séquencement linéaire de l'exécution d'une secondaire, ii suffit d'utiliser des primitives qui manipulent le pointeur d'instruction (voir le chapitre "Les Structures de Contrôle") : comme dans les machines réelles, l'adresse de la prochaine instruction à exécuter est trouvée juste après l'instruction de branchement ; TELE-FORTH n'utilise que des sauts absolus, plus efficaces sur 6502 que les sauts relatifs habituellement implémentés en fig-FORTH. Puisque FORTH a aussi accès à la pile de retour (voir R> et >R au chapitre "Opérateurs de Manipulation des Piles"), une secondaire peut parfaitement modifier le déroulement de celle qui l'a appelée, en allant modifier son pointeur d'instruction en attente au sommet de la pile retour (voilà pourquoi on vous dit de faire attention en utilisant la pile de retour) : c'est ce que fait (OF"), la seule structure de contrôle trop complexe pour être écrite en code machine.

Puisqu'il suffit de connaître le CFA d'un mot (primitive ou secondaire) pour lancer son exécution, il n'est pas nécessaire que ce soit par l'intermédiaire du PFA d'une secondaire, on peut aussi passer par la pile de calcul :
1° W:=(SP)+
charger W avec la valeur dépilée de la pile de calcul
2° mip:=(W)
même seconde étape que pour l'automate

EXECUTE

CODE EXECUTE (cfa --)
1L LDA, W STA, 1H LDA, W 1+ STA, INX, INX, W 1- JMP, ;C
c'est l'exécuteur du FORTH, utilisé par l'interprète

La plupart des autres structures ont des données dans leur PFA : pour une constante, sa valeur ; pour une variable en ROM, l'adresse de sa valeur en RAM ; etc...

Le chapitre suivant présente l'organisation mémoire de TELE-FORTH

11. ORGANISATION MEMOIRE

Voici l'occupation mémoire globale sur le TELESTRAT :

0000-008B réservé (moniteur)
008C-00BD pile de données (50 octets) (registre X pointeur de pile données)
00C0-00C7 N pour sous-programme SETUP (8 octets)
00C8-00C9 IP pointeur instructions FORTH
00CA saut indirect (6C)
00CB-00CC W registre de décodage
00CD-00CE UP pointeur zone USER
00CF XSAVE sauvegarde temporaire pointeur de pile

00D0-00FF réserve (TELEMATIC ou application)
0100-.... Terminal Input Buffer (TIB)
....-01FF pile retour (registre S pointeur de pile retour)
0200-02FF réservé (moniteur)
0300-037F réservé (entrées/sorties internes et extensions)
0380-03FF réservé (entrées/sorties utilisateur)
0400-055F réservé (moniteur et DOS)
0560-058F langage (FORTH utilise les premiers octets : voir interface avec moniteur)
0590-05FF réservé (éditeur basic)
0600-07FF réservé (langage ou application)
0800-083F zone USER
0840-0FFF tampons DOS pour 2 fichiers à accès direct
1000-1403 tampon FORTH pour 1 écran d'un bloc de 1 Ko
1404-.... variables initialisées en EPROM
    -.... espace libre pour dictionnaire FORTH
3FFF HIMEM en TELEMATIC (4000-B3FF arborescence)
97FF HIMEM en HIRES (9800-9FFF caractères, A000-BFDF écran)
B3FF HIMEM en TEXT (B400-BB7F caractères, BB80-BFDF écran)
C000-FFFF banque 0 : RAM interne (DOS)
banques 1-3 : port cartouche droit
banque 3 : ROM application (TELEMATIC)
banques 4-7 : port cartouche gauche
banque 6 : ROM langage (FORTH)
banque 7 : ROM reset (moniteur)

Table de démarrage :

C000 NOP, CLD JMP,
C004 NOP, WRM JMP,
C008 numéros de version
C010 adresse de la zone USER
C012 adresse de la base de la pile données (S0)
C014 adresse de la base de la pile retours (R0)
C016 adresse du tampon d'entrée (TIB)
C018 nombre maxi de caractères pour les noms (WIDTH)
C01A valeur initiale de WARNING (voir ERROR et MESSAGE)
C01C valeur initiale de FENCE (voir FORGET)
C01E valeur initiale de DP (fin du dicionnaire)
C020 valeur initiale de VOC-LINK (voir VOCABULARY)

Organisation du dictionnaire FORTH pour chaque entrée :

NFA
1 octet d'entête :
- bit 7=1 marque l'origine du NFA pour TRAVERSE
- bit 6 : 0=normal 1=IMMEDIATE
- bit 5 : 0=normal 1=SMUDGE
- bit 4-0 : nombre de caractères du nom
suivi de la chaîne de caractères du nom, dont dernier caractère : bit 7=1 pour marquer la fin du NFA

LFA
2 octets : pointeur sur le NFA de la définition précédente dans le même vocabulaire

CFA
2 octets : pointeur au code machine à exécuter (+2 octets si défini par DOES>)

PFA

zone paramètres

Contenu du CFA et PFA en fonction de la méthode de définition :

CODE... ;C
"primitives"
CFA pointe sur PFA
PFA : instructions machine

CREATE... ;CODE
(CFA est dans W, PFA atteint par 2 # LDY, W)Y LDA, ...)
CFA pointe après ;CODE du générateur
PFA : paramètres

<BUILDS... DOES>
(PFA sur la pile)
CFA pointe après ;CODE de DOES>
CFA+2 pointe après DOES> du générateur
PFA : paramètres

Contenu du CFA et du PFA pour les générateurs standard :

: (secondaires)
CFA pointe après ;CODE de :
PFA : instructions virtuelles (qui pointent sur CFA) terminées par ;S

CONSTANT
CFA pointe après ;CODE de CONSTANT
PFA : 2 octets valeur de la constante

VARIABLE
CFA pointe après ;CODE de VARIABLE
PFA : adresse RAM de la valeur de la variable (=PFA+2 en RAM)
RAM : 2 octets valeur de la variable

USER
CFA pointe après ;CODE de USER
PFA : 2 octets dont premier=index dans zone USER

VOCABULARY
CFA pointe après ;CODE de DOES>
CFA+2 : pointe après DOES> de VOCABULARY
PFA : adresse RAM structure vocabulaire (=PFA+4 en RAM)
PFA+2 : pointeur au vocabulaire précédent
RAM : 2 octets mot blanc (81 et A0)
RAM+2 : pointeur au NFA du dernier mot défini dans ce vocabulaire

12. L'INTERPRETEUR

On entend par là l'ensemble des mots qui lancent et font tourner la machine FORTH.

```
L: CLD1 ] COLD [ ASSEMBLER HEX
L: CLD 6C # LDA, W 1- STA, \ pour l'opération 2 de l'automate
10 +ORIGIN LDA, UP STA, \ init pointeur zone user
```

```
11 +ORIGIN LDA, UP 1+ STA,
14 +ORIGIN LDX, TSX, \ init pointeur pile retour
12 +ORIGIN LDX, \ init pointeur pile données
CLD1 0 100 U/ \ init pointeur instructions
# LDA, IP 1+ STA, # LDA, IP STA,
NEXT, FORTH \ exécuter COLD
```
c'est ici que tout commence pour FORTH, lancé pour un "démarrage à froid" par le moniteur

```
L: WRM1 ] WARM [ ASSEMBLER HEX
L: WRM
WRM1 0 100 U/ # LDA, IP 1+ STA, # LDA, IP STA,
' RP! JMP, FORTH \ lancer RP!, puis WARM
```
c'est ici que le moniteur relance FORTH lorsque vous appuyez sur le bouton RESET.

COLD

```
: COLD ( -- )
INIT-RAM DUP>R 2+ RAM-START R> @ CMOVE EMPTY-BUFFERS (
FIRST DUP USE ! PREV ! ) 12 +ORIGIN UP @ 6 + 010 CMOVE
ABORT ;
```
initialise la partie RAM utilisée par les variables et vocabulaires créés en ROM, vide les buffers FORTH, initialise la zone user, et enfin exécute ABORT ; COLD est la procédure de "démarrage à froid" du FORTH, lancée par le boot à CLD

WARM

```
: WARM ( -- )
EMPTY-BUFFERS ABORT ;
```
vide les buffers FORTH et exécute ABORT ; WARM est la procédure de redémarrage à chaud du FORTH, lancé par le boot WRM

ABORT

```
DEFER ABORT ' (ABORT) IS ABORT
```
c'est le point de passage obligé pour tout redémarrage ; ce vecteur, initialisé à (ABORT) peut être remplacé par une autre procédure que (ABORT), par exemple pour blinder une application relogeable contre les RESET indiscrets.

(ABORT)

```
: (ABORT) ( -- )
SP! DECIMAL TERMINAL ( BRK, fun RTS, ) 60 0 560 2! CR ."
TELE-FORTH v1.2 " HIMEM HERE - U. ." octets libres"
[COMPILE] FORTH DEFINITIONS STARTUP QUIT ;
```
réinitialise la pile, la base en décimal, les canaux d'entrée/sortie, signe en donnant l'espace mémoire disponible, installe FORTH comme vocabulaire CONTEXT et CURRENT, appelle STARTUP qui effectue les initialisations propres au TELESTRAT et enfin exécute QUIT

QUIT

```
: QUIT ( -- )
0 BLK ! [COMPILE] [ BEGIN CR RP! QUERY INTERPRET STATE @
0= IF PROMPT THEN AGAIN ;
```
réinitialise le flot d'entrée au TIB, passe en exécution, et boucle infiniment : initialise la pile retour, saisit une ligne au clavier, l'interprète, et affiche le prompt si en exécution ; QUIT est la boucle de base qui s'occupe de vos entrées et de tout remettre en ordre en cas

d'erreur (voir ERROR)

PROMPT

DEFER PROMPT ' OK IS PROMPT
message d'invite utilisé par QUIT, redéfinissable par l'utilisateur

OK

: OK (--)
." Ok" BASE @ CASE 0A OF ASCII . ENDOF (decimal : Ok.0) 10 OF ASCII # ENDOF (hexa : Ok#0) 2 OF ASCII % ENDOF (binaire : Ok%0) (autres bases : Ok%0) ASCII * SWAP ENDCASE EMIT DEPTH H. ;
message d'invite initial : affiche la base utilisée et le nombre de valeurs dans la pile

QUERY

: QUERY (--)
TIB @ 050 EXPECT 0 IN ! ;
fait saisir par EXPECT un maximum de 80 caractères dans le Terminal Input Buffer

EXPECT

: EXPECT (ad cnt --)
OVER + OVER DO KEY DUP I C! I 1+ OFF BL OVER > OVER 7E > OR IF CASE 7F OF DUP I = DUP R> 2- + >R IF 7 ELSE 8 DUP EMIT BL EMIT THEN ENDOF 0D OF 0 I C! LEAVE BL ENDOF 0 I C! R> 1- >R 10 OVER > 0VER 17 > OR IF 7 SWAP ELSE DUP THEN ENDCASE THEN EMIT LOOP DROP ;
c'est l'unique procédure de saisie de ligne au clavier : accepte et renvoie en écho à l'écran les caractères de code ASCII 32 à 126, c'est-à-dire les caractères imprimables et ceux de contrôle de 16 à 23 permettant de contrôler certains périphériques par le clavier du TELESTRAT ; la touche DEL génère le code ASCII 127=7Fh et provoque l'effacement du dernier caractère saisi ou un bip s'il n'en reste plus ; la touche RETURN génère le code ASCII 13=0Dh et provoque la fin de la saisie ; la saisie est également interrompue si le nombre de caractères dépasse cnt ; enfin les caractères saisis sont stockés à ad suivis de deux codes ASCII nuls pour que X soit exécuté pour sortir de la boucle infinie de INTERPRET

INTERPRET

: INTERPRET (--)
BEGIN -FIND IF STATE @ < IF CFA, ELSE CFA EXECUTE THEN ELSE HERE NUMBER DPL @ 1+ IF [COMPILE] DLITERAL ELSE DROP [COMPILE] LITERAL THEN THEN ?STACK AGAIN ;
c'est l'interprète à proprement parler : chaque mot (isolé par WORD), est recherché dans le vocabulaire CONTEXT puis CURRENT, et soit exécuté soit compilé (en fonction de l'état de la variable STATE) s'il est trouvé, sinon convertit en nombre (16, ou 32 bits si un point décimal est trouvé en cours de conversion) et soit compilé soit empilé (en fonction de l'état de la variable STATE) si la conversion réussit dans la base courante, sinon une erreur "inconnu dans ce vocabulaire" est générée ; l'état de la pile est testé à chaque mot ; la sortie de la boucle est effectuée par le "mot nul" qui est déclenché par l'interprétation des deux octets nuls toujours présent à la fin du buffer d'entrée (soit positionnés par EXPECT

soit présent à la suite du tampon disque).

<mot nul>

: <mot nul> (--)
BLK @ IF 1 BLK +! 0 IN ! ?EXEC THEN R>DROP ; IMMEDIATE
(pour B/SCR>1: BLK @ IF 1 BLK +! 0 IN ! BLK @ B/SCR 1-
AND 0= IF ?EXEC R>DROP THEN ELSE R>DROP THEN ; IMMEDIATE)
l'identificateur de ce mot est composé d'un seul caractère de code ASCII nul (C1 80) trouvé par (FIND) à la fin du flot d'entrée (toujours terminé par deux ASCII nuls) ; lorsque le flot d'entrée provient du clavier, la variable BLK est nulle, sinon elle est égale au numéro du bloc en cours d'interprétation sur le fichier courant ; dans ce dernier cas, le "mot nul" positionne le pointeur d'interprétation IN au début du bloc suivant, et dépile l'adresse de retour dans INTERPRET pour provoquer une sortie de sa boucle infinie.

STARTUP

: STARTUP (--)
060 0 0560 2! (L: BRK BRK, fun RTS,) 0321 C@ 7 AND 02F4 C! 4 +ORIGIN 02F5 ! (WARM sur reset) FLGTEL 1 CTST IF 88 MESSAGE ELSE (DOS non résident) 0840 07C0 ERASE 2 0549 C! 0840 0542 ! 1 FICNUM C! 0FFFF 0544 ! 0321 C@ 7 AND 054C C! FRW 054D ! (init XFIELD) LIT" FORTH.DAT" ?FILE IF 8 FTYPE C! XOPEN DOS 0 BLK ! [COMPILE] [RP! 1 LOAD QUIT THEN THEN ;
initialisations spécifiques au TELESTRAT : mise en place du sous-programme de break variable (pour MON), de l'adresse de reset (déclenche WARM quand on appuie sur le bouton de reset situé sur le côté droit du TELESTRAT) ; annonce l'absence éventuelle du DOS (s'il n'y a pas de lecteur connecté), sinon initialise et déclare deux blocs de contrôle de fichiers (deux pour permettre les copies de fichier à fichier), initialise le numéro du fichier par défaut à 1 (XFILE le fera passer de 1 à 2 ou inversement), installe le vectcur de traitement de fichiers à accès direct (voir le chapitre "STRATSED le DOS") et interprète l'écran 1 du fichier de démarrage "FORTH.DAT' s'il existe.

13. LES VARIABLES USER

Elles sont utilisées par l'interprète ; voir USER

```
S0        06 USER S0        \ base de la pile données
R0        08 USER R0        \ base de la pile retour
TIB       0A USER TIB       \ base Terminal Input Buffer
WIDTH     0C USER WIDTH     \ nombre max de caractères pour
NFA
WARNING   0E USER WARNING   \ paramétrage erreurs, voir
ERROR
FENCE     10 USER FENCE     \ limite pour FORGET, voir aussi
V<
DP        12 USER DP        \ pointeur de fin de
dictionnaire
VOC-LINK  14 USER VOC-LINK  \ lien des vocabulaires
BLK       16 USER BLK       \ numéro du bloc interprété
IN        18 USER IN        \ offset dans tampon d'entrée
OUT       1A USER OUT       \ colonne du curseur
SCR       1C USER SCR       \ numéro de l'écran édité
```

```
CONTEXT   20 USER CONTEXT   \ pointeur vocabulaire de
contexte
CURRENT  22 USER CURRENT  \ pointeur vocabulaire courant
STATE     24  USER  STATE     \  0=interprétation,  sinon
compilation
BASE     26 USER BASE     \ base de conversion numérique
DPL      28 USER DPL      \ nombre de décimales, -1=entier
CSP      2C USER CSP       \  repère  pour  pointeur  pile
données
R#       2E USER R#       \ position curseur écran édité
HLD      30 USER HLD      \ pointeur reculons dans PAD
```

14. LA GESTION DU DICTIONNAIRE

La variable DP pointe à la fin du dictionnaire, où seront ajoutées les nouvelles définitions. La variable CURRENT pointe sur le vocabulaire auquel seront liées les nouvelles définitions. La variable CONTEXT pointe sur le vocabulaire à partir duquel démarrera toute recherche ; si la recherche échoue dans le vocabulaire pointé par CONTEXT, elle est poursuivie à partir du vocabulaire pointé par CURRENT, si la recherche échoue encore, l'interprète tente une conversion numérique dans la base courante ; si la conversion échoue, l'interprète génère une erreur "inconnu dans ce vocabulaire".

HERE

```
: HERE ( -- adr )
DP @ ;
```

retourne l'adresse de fin du dictionnaire.

ALLOT

```
: ALLOT ( n -- )
DP +! ;
```

réserve n octets dans le dictionnaire (par ex. pour un tampon ou un tableau).

C,

```
: C, ( c -- )
HERE C! 1 ALLOT ;
```

ajoute l'octet c à la fin du dictionnaire.

,

```
: , ( n -- )
HERE ! 2 ALLOT ;
```

ajoute le mot n à la fin du dictionnaire.

COMPILE

```
: COMPILE ( -- )
?COMP R> WCOUNT , >R ;
```

ajoute à la définition en cours de compilation le CFA qui suit COMPILE dans la définition en cours d'exécution ; généralement utilisé dans les mots immédiats (voir la définition de LITERAL par exemple)

[COMPILE]

```
: [COMPILE] ( -- )
-FIND 0= 5 ?ERROR DROP CFA , ; IMMEDIATE
```

force la compilation de mots immédiats (voir la définition de DLITERAL par exemple)

LIT

CODE LIT (-- n)
IP)Y LDA, PHA, IP INC, 0= IF, IP 1+ INC, THEN, IP)Y LDA, IP INC, 0= IF, IP 1+ INC, THEN, PUSH, ;C
retourne sur la pile la valeur qui suit LIT dans la définition en cours d'exécution ; est compilé par LITERAL pour chaque constante numérique dans les définitions

LITERAL

: LITERAL (n --)
STATE @ IF COMPILE LIT , THEN ; IMMEDIATE
sans effet en exécution, LITERAL est appelé par INTERPRET pour compiler dans une définition une constante numérique

DLITERAL

: DLITERAL (d --)
STATE @ IF SWAP [COMPILE] LITERAL [COMPILE] LITERAL THEN ; IMMEDIATE
même rôle que LITERAL, mais pour des constantes numériques sur 32 bits

(LIT")

: (LIT") (-- ad)
R> DUP COUNT + >R ;
retourne l'adresse de la chaîne compilée par LIT"

LIT"

: LIT" (--)
?COMP COMPILE (LIT") ASCII " WORD HERE C@ 1+ ALLOT ; IMMEDIATE
compile comme ." la chaîne de caractères qui suit

TRAVERSE

: TRAVERSE (nfa 1 -- fin-du-nfa / fin-du-nfa -1 -- nfa)
SWAP BEGIN OVER + DUP 080 CTST UNTIL NIP ;
permet de traverser le "name field" dans les deux sens ; voir PFA et NFA.

PFA

: PFA (nfa -- pfa)
1 TRAVERSE 5 + ;
permet de passer du NFA au PFA.

NFA

: NFA (pfa -- nfa)
5 - -1 TRAVERSE ;
permet de passer du PFA au NFA.

CFA

: CFA (pfa -- cfa)
2- ;
permet de passer du PFA au CFA.

LFA

: LFA (pfa -- lfa)
4 - ;
permet de passer du PFA au LFA.

LATEST

: LATEST (-- nfa)

CURRENT @ @ ;
retourne l'adresse du nom du dernier mot enregistré dans le vocabulaire CURRENT.

DEFINITIONS
: DEFINITIONS (--)
CONTEXT @ CURRENT ! ;
rend le vocabulaire CURRENT égal au vocabulaire CONTEXT, pour que les nouvelles définitions soient enregistrées dans le vocabulaire CONTEXT.

IMMEDIATE
: IMMEDIATE (--)
LATEST 040 TOGGLE ;
inverse l'état "immédiat" du dernier mot enregistré dans le dictionnaire ; les mots "immédiats" sont exécutés quel que soit l'état du FORTH (interprétation ou compilation).

SMUDGE
: SMUDGE (--)
LATEST 020 TOGGLE ;
inverse l'état "smudgé" du dernier mot enregistré dans le dictionnaire ; les mots "smudgés" ne seront pas trouvés lors d'une recherche lexicale ; un mot n'est "désmudgé" que lorsque la compilation de sa définition s'est conclue correctement.

(FIND)
CODE (FIND) (refadr nfa1 -- pfa nfabyte 1 /-- 0)
2 SETUP, X!, BEGIN, 0 # LDY, N)Y LDA, N 2+)Y EOR, 3F # AND, 0= IF, (1) BEGIN, INY, N)Y LDA, N 2+)Y EOR, .A ASL, 0= IF, (2) 2SWAP CS UNTIL, X@, DEX, DEX, DEX, DEX, CLC, TYA, 5 # ADC, N ADC, 2L STA, 0 # LDY, TYA, N 1+ ADC, 2H STA, 1H STY, N)Y LDA, 1L STA, 1 # LDA, PHA, PUSH, (ok: -- 1) THEN, (2) CS NOT IF, (3) (dernier car.nfa trouvé) 2SWAP THEN, (1) (recherche fin nfa) BEGIN, INY, N)Y LDA, 0< UNTIL, THEN, (3) INY, N)Y LDA, TAX, INY, N)Y LDA, N 1+ STA, N STX, N ORA, (N = nfa) 0= UNTIL, X@, 0 # LDA, PHA, PUSH, (-- 0) ;C
primitive de recherche lexicale : la chaîne dont on cherche l'entrée dans le dictionnaire est située à refadr ; la recherche commence à nfa1 : si les longueurs des chaînes sont égales (à nfa1, seuls les 6 bits de poids faible sont significatifs), les caractères sont comparés un à un : si différence, on passe au nfa suivant (par le LFA situé après la fin du nfa, dont le dernier caractère a son bit 7 mis à un, voir CREATE) ; si l'entrée correspondante est trouvée, sont retournés son PFA, l'octet trouvé à son NFA et un drapeau vrai ; sinon seul un drapeau faux est retourné.

-FIND
: -FIND (-- pfa nfabyte 1 /-- 0)
BL WORD HERE CONTEXT @ @ (FIND) DUP 0= IF CONTEXT @ CURRENT @ - IF DROP HERE LATEST (FIND) THEN THEN ;
routine de recherche lexicale : utilise (FIND) pour chercher le mot (isolé par WORD dans le buffer d'entrée) d'abord dans le vocabulaire CONTEXT puis si échec dans le vocabulaire CURRENT s'il est différent de CONTEXT.

'

: ' (-- pfa)
-FIND 0= 5 ?ERROR DROP [COMPILE] LITERAL ; IMMEDIATE
retourne le PFA du mot suivant dans le buffer d'entrée ; si la recherche lexicale échoue, l'erreur 0 ("XXX inconnu") est déclenchée ; ' (prononcer "tick") fonctionne également si FORTH est en mode compilation : le PFA du mot trouvé est compilé dans le dictionnaire en tant que litéral ; voir LITERAL.

CREATE

: CREATE (--)
(redefini?:) -FIND IF DROP NFA ID. 4 MESSAGE SPACE THEN HERE (-- here | nfa:) DUP C@ WIDTH @ MIN 1+ ALLOT (pour éviter saut de page 6502:) DP C@ 0FD = ALLOT (marquage extrémités nfa:) DUP 0A0 TOGGLE HERE 1- 080 TOGGLE (here -- | lfa:) LATEST , CURRENT @ ! (cfa:) HERE 2+ , ;
crée une nouvelle entrée dans le dictionnaire ; un message "XXX (redéfini)" signale si une entrée du même nom existait déjà ; la longueur du nom est limitée à WIDTH caractères (normalement 15) ; le bit 7 des deux extrémités du nom sont mis à 1 pour la routine de recherche lexicale (voir (FIND) et TRAVERSE) ; la nouvelle entrée est "smudgée", c'est-à-dire qu'elle ne sera pas trouvée par (FIND) tant qu'elle ne sera pas "désmudgée" ; le LFA pointe sur le nom précédent défini dans le même vocabulaire ; le CFA pointe sur le PFA, comme pour les primitives.

RECURSE

: RECURSE (--)
?COMP LATEST PFA CFA , ; IMMEDIATE
en FORTH, on ne peut pas utiliser le nom d'un mot en cours de définition dans sa propre définition, car son entête est rendu introuvable lors de sa création (CREATE "smudge" l'entête, qui ne sera "désmudgé" par ; que si la compilation se termine sans erreur) ; pour permettre les définitions récursives (c'est-à-dire qui font appel à elles-même), il faut employer RECURSE à la place du nom du mot en cours de définition ; par exemple, ROLL aurait pu être défini récursivement ainsi :
: ROLL 1- -DUP IF SWAP >R RECURSE R> SWAP THEN ;
ATTENTION : la profondeur de la pile de données de cette implémentation de FORTH sur TELESTRAT est limitée à 25 (sa taille n'est que de 50 octets, le seul bloc contigü en page 0 non utilisé par le système) ; il faut donc éviter d'écrire des algorithmes récursifs trop gourmands en pile, sinon celle-ci viendra écraser les registres de la machine virtuelle FORTH qui n'y retrouvera plus ses petits...

:

: : (--)
?EXEC !CSP CURRENT @ CONTEXT ! CREATE] ;CODE IP 1+ LDA, PHA, IP LDA, PHA, CLC, 2 # LDA, W AC, IP STA, TYA, W 1+ ADC, IP 1+ STA, NEXT, ;C IMMEDIATE
"deux-points" définit un sous-programme ; crée une nouvelle entrée dans le vocabulaire CURRENT et passe en mode compilation ; CONTEXT prend la valeur de CURRENT pour que les recherches ne se fassent que dans le vocabulaire CURRENT ; la nouvelle entrée étant "smudgée" par CREATE, elle ne sera validée que si la définition est close sans erreur par "point-virgule" ou par ;CODE

;

: ; (--)
?CSP COMPILE ;S SMUDGE [COMPILE] [; IMMEDIATE
"point-virgule" clot une définition de sous-programme ouverte par "deux-point" ; génère une erreur "définition incomplète" si la profondeur de la pile a changé entre "deux-points" et "point-virgule", sinon "désmudge" la dernière entrée du vocabulaire (créée par "deux-points") pour la valider et repasse en mode exécution

[

: [(--)
0 STATE ! : IMMEDIATE
force l'interprète à passer en mode exécution

]

:] (--)
0C0 STATE ! ;
force l'interprète à passer en mode compilation

!CSP

: !CSP (--)
SP@ CSP ! ;
sauvegarde dans la variable CSP la profondeur de la pile

?CSP

: ?CSP (--)
SP@ CSP @ - 0E ?ERROR ;
génère une erreur "définition incomplète" si la profondeur de la pile est différente de celle mémorisée dans CSP

CONSTANT

: CONSTANT (n -- | -- n)
CREATE SMUDGE , ; CODE (pfa -- | @)
2 # LDY, W)Y LDA, PHA, INY, W)Y LDA, PUSH, ;C
crée une constante de valeur n ; une constante retourne sa valeur sur la pile
5 CONSTANT CINQ CINQ . <RETURN> 5 ok.0

VARIABLE

: VARIABLE (n -- | -- adr)
CREATE SMUDGE HERE 2+ , , ;CODE (pfa -- adr | @)
2 # LDY, W)Y LDA, PHA, INY, W)Y LDA, PUSH, ;C
crée une variable initialisée à n ; une variable retourne sur la pile l'adresse à laquelle se trouve sa valeur (pour ! et @ par exemple) ; (voir en introduction le paragraphe sur les différences d'implémentations du FORTH en RAM et en ROM)
79 VARIABLE STANDARD STANDARD ? <RETURN> 19 ok.0
83 STANDARD ! STANDARD ? <RETURN> 83 ok.0

USER

: USER (n -- | -- adr)
CONSTANT ;CODE (pfa -- adr | C@ UP +)
CLC, 2 # LDY, W)Y LDA, UP ADC, PHA, 0 # LDA, UP 1+ ADC, PUSH, ;C
crée une "variable-utilisateur" ; les variables-utilisateur s'utilisent de la même manière que les autres variables ; (voir en introduction le paragraphe sur les implémentations multitâche de FORTH)

+ORIGIN

: +ORIGIN (n -- adr)
0C000 + ;
retourne l'adresse du n-ième octet du dictionnaire ; utilisé par COLD pour initialiser les "user variables".

DEFER

: DEFER (--)
CREATE SMUDGE HERE 2 ALLOT HERE 2 ALLOT ' NOOP CFA OVER ! SWAP ! ;CODE (pfa -- | @ @ EXECUTE) 3 # LDY, W)Y LDA, N 1+ STA, DEY, W)Y LDA, N STA, DEY, N)Y LDA, W 1+ STA, DEY, N)Y LDA, W STA, W 1- JMP, ;C
définit un "vecteur", c'est-à-dire un mot dont l'action est modifiable (de même qu'une variable contient une valeur modifiable) ; la valeur initiale d'un vecteur est le mot NOOP qui ne fait rien (et qui sert principalement à initialiser les vecteurs) ; IS permet de modifier la valeur d'un vecteur ; un exemple de vecteur caractéristique est celui de EMIT qui permet d'imprimer des caractères soit en mode TEXT (avec CEMIT) soit en mode HIRES (avec HEMIT) ; l'exemple qui suit montre un fonctionnement équivalent, bien que moins efficace et plus lourd d'emploi, en utilisant une variable :
DEFER VECTEUR <RETURN> ok.0 ' 1+ IS VECTEUR <RETURN> ok.0 3 VECTEUR . <RETURN> 4 ok.0
' NOOP CFA VARIABLE WECTEUR <RETURN> ok.0 ' 1+ CFA WECTEUR ! <RETURN> ok.0 3 WECTEUR @ EXECUTE . <RETURN> 4 ok.0

(IS)

: (IS) (pfa pfa --)
SWAP CFA SWAP @ ! ;
compile dans une définition lors de l'emploi de IS en mode compilation

IS

: IS (pfa --)
[COMPILE] ' STATE @ IF COMPILE (IS) ELSE (IS) THEN ; IMMEDIATE
permet de changer la valeur d'un vecteur ; voir DEFER

<BUILDS

: <BUILDS (--)
0 CONSTANT ;
voir DOES>

DOES>

: DOES> (-- | -- pfa)
R> LATEST PFA ! ;CODE (-- pfa) IP 1+ LDA, PHA, IP LDA, PHA, 2 # LDY, W)Y LDA, IP STA, INY, W)Y LDA, IP 1+ STA, CLC, 4 # LDA, W ADC, PHA, 0 # LDA, W 1+ ADC, PUSH, ;C
<BUILDS et DOES> s'emploient de paire dans une définition qui permet de créer de nouvelles familles de mots FORTH ; deux exemples montreront mieux leur emploi :
Exemple 1 : on pourrait définir CONSTANT de la manière suivante :
: CSTE (n -- | -- adr) <BUILDS , DOES> @ ;
reprenons l'exemple d'emploi donné pour CONSTANT :
5 CSTE CINQ CINQ . <RETURN> 5 ok.0
au moment de l'exécution de CSTE, 5 est sur la pile ; de

la définition de CSTE, seule la partie "construction", comprise entre <BUILDS et DOES> va être exécutée : <BUILDS va créer un nouvel entête de nom CINQ dans le dictionnaire, puis "virgule" va prendre la valeur 5 sur la pile et la compiler à la suite de l'entête, enfin DOES> va terminer le travail (en faisant pointer le CFA de CINQ sur la première instruction machine suivant ;CODE dans la définition de DOES>, et le premier mot du PFA de CINQ, réservé par <BUILDS, sur le premier mot suivant DOES> dans la définition de CSTE) ; au moment de l'exécution de CINQ, ce sont les instructions machine qui suivent ;CODE dans la définition de DOES> qui vont être exécutées : elles vont laisser sur la pile l'adresse de la valeur 5 (suivant le mot réservé par <BUILDS après l'entête de CINQ), et provoquer l'exécution de la partie "action" de CSTE (qui suit DOES> dans la définition de CSTE) : @ va remplacer l'adresse sur la pile par son contenu, c'est-à-dire la valeur 5.

Exemple 2 : définissons une famille de tableaux (à une seule dimension) de variable :

```
: ARRAY ( dim -- | n -- adr )
<BUILDS 2* HERE OVER ERASE ALLOT DOES> SWAP 2* + ;
```

ARRAY crée une nouvelle entrée, d'entête TABLE par exemple, dans le dictionnaire, et réserve dim mots initialisés à 0 ; quand TABLE est exécuté, l'indice n au sommet de la pile est transformé en l'adresse du nième élément du tableau :

```
4 ARRAY TABLE 0 TABLE ? 2 TABLE ? <RETURN>  0 0 ok.0
13 3 TABLE ! 10 0 TABLE ! 0 TABLE ? 3 TABLE ? <RETURN>  10
13 ok.0
```

(;CODE)

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

compile par ;CODE qui n'est disponible qu'avec l'extension assembleur ; ;CODE permet d'écrire en code machine la partie "action" de la définition d'une nouvelle famille de mots FORTH (là où DOES> permet de l'écrire en FORTH)

VOCABULARY

```
: VOCABULARY ( -- )
<BUILDS HERE 4 + , HERE VOC-LINK @ , VOC-LINK !  A081  ,
CURRENT @ CFA , DOES> @ 2+ CONTEXT ! ;
```

définit un nouveau vocabulaire (dans le vocabulaire CURRENT) ; lors de son exécution, un mot défini par VOCABULARY devient le vocabulaire CONTEXT (la variable CONTEXT pointe dessus)

FORTH

```
VOCABULARY FORTH IMMEDIATE
```

vocabulaire racine contenant tous les mots de base du FORTH

ID.

```
: ID.  ( nfa -- )
COUNT 01F AND DUP 0> IF 0 DO COUNT 07F AND EMIT LOOP SPACE
DROP ELSE 2DROP THEN ;
```

affiche l'identificateur d'un mot dont on donne le NFA

VLIST

```
: VLIST ( -- )
```

```
CR CONTEXT @ @ BEGIN 028 OUT @ - OVER C@ 01F AND <  IF  CR
THEN  DUP  ID.  OUT  @ MINUS 7 AND SPACES PFA LFA @ DUP 0=
?TERMSTOP OR UNTIL DROP ;
```
liste les identificateurs des mots du vocabulaire CONTEXT dans l'ordre de recherche lexicale.

VOC-LIST

```
: VOC-LIST ( -- )
VOC-LINK BEGIN @ -DUP WHILE DUP 4 - NFA ID. REPEAT ;
```
liste les identificateurs de tous les vocabulaires du dictionnaire

V<

```
: V< ( ad2 ad1 -- b )
0C000 - SWAP 0C000 - SWAP U< ;
```
compare des adresses dans l'ordre historique du dictionnaire (voir FORGET et U<)

FORGET

```
: FORGET ( -- )
CURRENT  @  CONTEXT @ - 018 ?ERROR [COMPILE] ' DUP FENCE @
V< 015 ?ERROR NFA >R VOC-LINK BEGIN @ DUP R V<  UNTIL  DUP
VOC-LINK  !   BEGIN  DUP  2-  @ 2+ DUP BEGIN @ DUP R V< 0=
WHILE PFA LFA REPEAT SWAP ! @ DUP 0= UNTIL DROP R> DP ! ;
```
efface tout ce que contient le dictionnaire à partir du mot (inclus) dont l'identificateur suit FORGET dans le flot d'entrée ; FORGET prend soin de remettre en place tous les liens de vocabulaires.

A titre d'exercice sur les liens de vocabulaires, voici comment a été créé l'index ; DICO crée une table de tous les NFA du dictionnaire, par exploration systématique des liens de tous les vocabulaires ; SORT les trie par ordre alphabétique (la méthode de tri, dite "à bulle" est de loin la plus lente, mais la plus simple à programmer) ; DLIST affiche le résultat.

```
: DICO ( -- org end )
HERE  >R PAD DUP DP ! VOC-LINK BEGIN @ -DUP WHILE DUP 2- @
2+ BEGIN @ DUP 0= OVER 0A081 = OR NOT WHILE DUP , PFA  LFA
REPEAT DROP REPEAT HERE R> DP ! ;
: $- ( ad2 ad1 -- n | "soustraction" de chaînes )
COUNT  01F  AND  ROT  COUNT 01F AND ROT 2DUP - >R MIN >R 0
BEGIN DROP SWAP COUNT 07F AND ROT COUNT 07F AND  -  R>  1-
DUP>R 0= OVER OR UNTIL R>DROP R> OVER IF SWAP THEN NIP NIP
NIP ;
: SORT ( org end -- | tri bulle par ordre alphabetique )
OVER >R DO 0 I J DO I 2@ $- 0< IF 1+ I 2@ SWAP I 2! THEN 2
+LOOP ?LEAVE -2 +LOOP R>DROP ;
: DLIST ( org end -- )
SWAP DO I @ ID. ?TERMSTOP ?LEAVE 2 +LOOP ;
: WORDS ( -- )
DICO 2DUP SORT DLIST ;
```

15. UN PETIT DECOMPILATEUR

Les cinq mots qui suivent n'ont pas d'autre prétention que de faciliter la "décompilation manuelle" de définitions FORTH ; ce sont de petits outils utiles pour vérifier par exemple le fonctionnement de nouveaux mots immédiats dans

le cas de nouvelles structures de contrôle, ou de nouveaux générateurs de familles de mots ; en donnant les sources de tous les mots présents dans l'EPROM du TELEFORTH, nous espérons avoir déjà satisfait la curiosité de ceux qui veulent toujours en savoir plus, mais nous ne doutons pas de l'insatiabilité de certains, qui se feront un plaisir d'utiliser, et nous imaginons facilement, de développer cette boîte à outils de campagne.

\C

```
: \C ( adr -- adr+2 )
DUP U. WCOUNT 2+ NFA ID. ;
```

"Cfa" : décompile le CFA situé à l'adresse adr ; retourne l'adresse suivante

\L

```
: \L ( adr -- adr+2 )
DUP U. WCOUNT . ;
```

"Lit" : affiche le mot situé à adr (après un LIT) ; retourne l'adresse suivante

\S

```
: \S ( adr -- adr+cnt+1 )
DUP U. COUNT ASCII " EMIT 2DUP TYPE ASCII " EMIT BL EMIT +
;
```

"String" : affiche la chaîne de cnt caractères, commençant par un octet de compte, située à adr ; retourne l'octet suivant la chaîne de caractères

\J

```
: \J ( adr -- adr+2 )
DUP U. WCOUNT DUP . @ 2+ NFA ID. ;
```

"Jump" : décompile un saut (après BRANCH 0BRANCH (LOOP) (+LOOP) (OF), voir les structures de contrôle) en affichant son adresse et en décompilant le CFA du mot situé au bout du saut

\D

```
: \D ( adr cnt -- adr+cnt )
OVER U. 0 DO COUNT 3 .R LOOP SPACE ;
```

"Dump" : affiche le contenu des cnt octets situés à l'adresse adr ; retourne l'adresse du premier octet non "dumpé"

Un exemple, en supposant que HERE soit en 1000 :

```
: TEST 3 0 DO IF ." VRAI" ELSE ." FAUX" THEN LOOP ;
<RETURN> ok#0
' TEST <RETURN>  ok#1 \ la définition démarre au PFA de
TEST
\C <RETURN>  1008 LIT ok#1 \ un LIT, utilisons \L
\L <RETURN>  100A 3 ok#1 \ qui vaut 3
\C <RETURN>  100C 0 ok#1 \ 0 est une constante, donc pas
de LIT
\C <RETURN>  100E (DO) ok#1 \ DO est immédiat et a compilé
(DO)
\C <RETURN>  1010 0BRANCH ok#1 \  IF a compilé 0BRANCH,
utilisons \J
\J <RETURN>  1012 101F (.") ok#1 \ il saute après ELSE sur
(.")
\C <RETURN>  1014 (.") ok#1 \ ." a compilé (."), utilisons
\S
```

```
\S  <RETURN>   1016 "VRAI" ok#1 \ 4 caractères +1 octet de
compte = 5
\C <RETURN>  101B BRANCH ok#1 \ 5  octets  après,  ELSE  a
compilé BRANCH
\J  <RETURN>  101D 1026 (LOOP) ok#1 \ qui saute après THEN
sur (LOOP)
\C <RETURN>  101F (.") ok#1 \ ." a compilé (."), utilisons
\S
\S  <RETURN>   1021 "FAUX" ok#1 \ 4 caractères +1 octet de
compte = 5
\C <RETURN>  1026 (LOOP) ok#1 \ 5  octets  après,  LOOP  a
compilé (LOOP)
\J  <RETURN>   1028 1010 0BRANCH ok#1 \ qui saute après DO
sur 0BRANCH =IF
\C <RETURN>  102A ;S ok#1 \ ; a  compilé  ;S  et  clos  la
définition
. <RETURN>  102C ok#0
```

16. LES STRUCTURES DE CONTROLE

FORTH est un langage structuré : le programmeur ignore les GOTO, c'est le compilateur qui les calcule à partir des structures de contrôle.
FORTH supporte en standard les structures de contrôle suivantes :

```
<test> IF <corps vrai> THEN ( THEN et ENDIF synonymes )
<test> IF <corps vrai> ELSE <corps faux> THEN
BEGIN <corps de boucle infinie> AGAIN
BEGIN <corps de boucle> <test fin> UNTIL (  UNTIL  et  END
synonymes )
BEGIN <test encore> WHILE <corps de boucle> REPEAT
<fin+1> <ini> DO <corps de boucle incrément unitaire> LOOP
<fin+1> <ini> DO <corps de boucle> <incrément arith> +LOOP
```

TELE-FORTH supporte en plus la construction CASE, avec un (OF) en code machine :

```
CASE <sélecteur>
<choix1> OF <corps1> ENDOF
<choixN> OF <corpsN> ENDOF
<corps par défaut> ENDCASE
```

FORTH est un des rares langages qui permette au programmeur de définir ses propres structures de contrôle : c'est ce qui a été fait avec OF" pour les applications TELEMATIC (voir SERVEUR dans le chapitre "Gestion des Canaux d'Entrées/Sorties"). TELE-FORTH supporte également une structure de test interprétée (les précédentes ne peuvent s'utiliser qu'en compilation, c'est-à-dire à l'intérieur d'une définition), utilisée principalement pour la compilation conditionnelle :

```
<test> IF( <corps vrai> )ELSE( <corps faux> )ENDIF
```

MRK>

```
: MRK> ( -- ad )
HERE 0 , ;
```

laisse sur la pile l'adresse à laquelle est réservé un mot pour installation ultérieure par RES> d'une adresse de saut

RES>

```
: RES> ( ad -- )
```

```
      HERE SWAP ! ;
      résoud un branchement avant marqué par MRK>

MRK<
      : MRK< ( -- ad )
      HERE ;
      laisse sur la pile l'adresse à laquelle devra se faire  un
      saut installé par RES<

RES<
      : RES< ( ad -- )
      , ;
      résoud un branchement arrière marqué par MRK<

IF
      : IF ( -- ad 1 )
      ?COMP COMPILE 0BRANCH MRK> 1 ; IMMEDIATE

ELSE
      : ELSE ( ad 1 -- ad' 1 )
      1 ?PAIRS COMPILE BRANCH MRK> SWAP RES> 1; IMMEDIATE

ENDIF
      : ENDIF ( ad 1 -- )
      1 ?PAIRS RES> ; IMMEDIATE

THEN
      : THEN ( ad 1 -- )
      [COMPILE] ENDIF ; IMMEDIATE

BEGIN
      : BEGIN ( -- ad 2 )
      ?COMP MRK< 2 ; IMMEDIATE

AGAIN
      : AGAIN ( ad 2 -- )
      2 ?PAIRS COMPILE BRANCH RES< ; IMMEDIATE

UNTIL
      : UNTIL ( ad 2 -- )
      2 ?PAIRS COMPILE 0BRANCH RES< ; IMMEDIATE

END
      : END ( ad -- 2 )
      [COMPILE] UNTIL ; IMMEDIATE

WHILE
      : WHILE ( ad 2 -- ad ad' 3 )
      2 ?PAIRS COMPILE 0BRANCH MRK> 3 ; IMMEDIATE

REPEAT
      : REPEAT ( ad ad' 3 -- )
      3 ?PAIRS COMPILE BRANCH SWAP RES< RES> ; IMMEDIATE

DO
      : DO ( -- ad 4 )
      ?COMP COMPILE (DO) MRK< 4 ; IMMEDIATE

LOOP
      : LOOP ( ad 4 -- )
      4 ?PAIRS COMPILE (LOOP) RES< ; IMMEDIATE
```

```
+LOOP
      : +LOOP ( ad 4 -- )
      4 ?PAIRS COMPILE (+LOOP) RES< ; IMMEDIATE

CASE
      : CASE ( -- n 5 )
      ?COMP CSP @ !CSP 5 ; IMMEDIATE

OF
      : OF ( 5 -- ad 6 )
      5 ?PAIRS COMPILE (OF) MRK> 6 ; IMMEDIATE

OF"
      : OF" ( 5 -- ad 6 )
      5  ?PAIRS COMPILE (OF") ASCII " WORD HERE C@ 1+ ALLOT MRK>
      6 ; IMMEDIATE

ENDOF
      : ENDOF ( ad 6 -- ad' 5 )
      6 ?PAIRS COMPILE BRANCH MRK> SWAP RES> 5 ; IMMEDIATE

ENDCASE
      : ENDCASE ( n adN ... ad1 5 -- )
      5 ?PAIRS COMPILE DROP BEGIN SP@ CSP @ - WHILE RES>  REPEAT
      CSP ! ; IMMEDIATE
```

Voici les bases sur lesquelles reposent toutes les structures de contrôle FORTH :

```
ASSEMBLER

L: NLOOP \ point d'entrée pour (+LOOP) ( incrément -- )
\  [(RP):=(RP)+inc]  si  incrément<0   [si   (RP)<=(RP+2)]
sinon...
1H  LDA,  PHA, PHA, 1L LDA, INX, INX, X=RP, INX, INX, CLC,
1L ADC, 1L STA, PLA, 1H ADC, 1H STA, PLA, 0< IF,  CLC,  1L
LDA, 2L SBC, 1H LDA, 2H SBC, SKIP,

L: 1LOOP \ point d'entrée pour (LOOP) ( -- )
\  [(RP):=(RP)+1]  ...[si  (RP+2)<=(RP)]:[RP:=RP+4]  sinon
branch
X=RP, 1L INC, 0= IF, 1H INC, THEN, 2SWAP  THEN,  X=RP,  1J
INC, 0= IF, AH INC, THEN, 2SWAP THEN, CLC, 2L LDA, 1L SBC,
2H LDA, 1H SBC, THEN, X=SP, .A ASL, CS IF, ( 1 ) PLA, PLA,
PLA, PLA, NOJUMP,

L: <>JUMP  point d'entrée pour (OF) ( clé sélecteur -- clé
/-- )
\ branchement si clé<>sélecteur
INX, INX, 0L LDA, 1L CMP, 0= IF, ( 2 ) 0H LDA, 1H CMP,  0=
IF, ( 3 ) INX, INX, NOJUMP,

L: 0JUMP \ point d'entrée pour 0BRANCH ( b -- )
\ branchement si b=0
INX, INX, 0L LDA, 0H ORA, 0= IF, ( 4 )

L: NOJUMP \ point d'arrivée pour la macro de saut NOJUMP,
\ [IP:=IP+2] continuer en séquence sans branchement
CLC,  IP  LDA,  2  #  ADC, IP STA, CS IF, IP 1+ INC, THEN,
NEXT, THEN, ( 4 ) THEN, ( 3 ) THEN, ( 2 ) THEN, ( 1 )
```

```
      L: JUMP \ point d'entrée pour BRANCH ( -- )
      \ point d'arrivée pour la macro de saut JUMP,
      \  [IP:=(IP)] branchement  absolu  (plus  rapide  que
      branchement relatif)
      IP  )Y  LDA,  PHA, INY, IP )Y LDA, IP 1+ STA, PLA, IP STA,
      NEXT 2+ JMP, ( car Y=1 )

      FORTH

(+LOOP)
      CODE (+LOOP) ( n -- )
      NLOOP HERE 2- ! ;C
      primitive de test de fin de boucle à incrément variable

(LOOP)
      CODE (LOOP) ( -- )
      1LOOP HERE 2 - ! ;C
      primitive de test de fin de boucle à incrément unitaire

(OF)
      CODE (OF) ( clé sélecteur -- clé /-- )
      <>JUMP HEPE 2- ! ;C
      primitive de test à choix multiples sur valeur numérique

(OF")
      (OF") ( ad -- ad /-- )
      DUP R> DUP C@ 1+ 2DUP + >R ( ad ad ad' cnt ) S= IF DROP R>
      2+ ELSE R> @ THEN >R ;
      primitive  de  test à choix multiples sur constante chaîne
      de caractères

0BRANCH
      CODE 0BRANCH ( b -- )
      0JUMP HERE 2- ! ;C
      primitive de branchement conditionnel

BRANCH
      CODE BRANCH ( -- )
      JUMP HERE 2- ! ;C
      primitive de branchement inconditionnel

(DO)
      CODE (DO) ( fin+1 ini -- )
      2H LDA, PHA, 2L LDA, PHA, \ valeur finale de l'index
      1H LDA, PHA, 1L LDA, PHA, \ valeur initiale de l'index
      L: POPTWO \ point d'arrivée de la macro de saut POPTWO,
      INX, INX,
      L: POP \ point d'arrivée de la macro de saut POP,
      INX, INX, NEXT, ;C
      primitive d'initialisation de boucle à incrément

?LEAVE
      CODE ?LEAVE ( b -- | IF LEAVE THEN )
      INX, INX, 0L LDA, 0H ORA, 0= NOT IF,
      L: ABORTLOOP \ point d'entrée de LEAVE
      X=RP, 1L LDA, 2L STA, 1H LDA, 2H STA, X=SP, THEN, NEXT, ;C
      primitive d'abandon de boucle conditionnel

LEAVE
      : LEAVE ( -- )
```

ABORTLOOP HERE 2- ! ;
primitive d'abandon de boucle inconditionnel

I
CODE I (-- index)
X=RP, 1L LDA, PHA, 1H LDA, X=SP, PUSH, ;C
retourne l'index de la boucle courante

J
CODE J (-- index)
X=RP, 3L LDA, PHA, 3H LDA, X=SP, PUSH, ;C
retourne l'index de la boucle englobant la boucle courante
: BOUCLES 2 0 DO 3 0 DO J . I . ." , " LOOP SPACE LOOP ;
<RETURN> ok.0
B0UCLES <RETURN> 0 0 , 0 1 , 0 2 , 1 0 , 1 1 , 1 2 , 2 0 , 2 1 , 2 2 , ok.0

IF(
IF((b --)
0= IF BEGIN BL WORD LIT")ELSE(" HERE DUP C@ 1+ S= LIT")ENDIF" HERE DUP C@ 1+ S= OR UNTIL THEN ; IMMEDIATE
s'utilise avec)ELSE(et/ou)ENDIF pour autoriser la compilation conditionnelle de certaines parties de source ; en utilisation à l'intérieur d'une définition, utiliser une constante IMMEDIATE pour la valeur boolénne

)ELSE(
:)ELSE((--)
BEGIN BL WORD LIT")ENDIF" HERE DUP C@ 1+ S= UNTIL ; IMMEDIATE
voir IF(

)ENDIF
:)ENDIF (--)
; IMMEDIATE
n'est là que pour IF(et)ELSE(

17. GESTION DES ERREURS

WARNING
0E USER WARNING
cette variable permet de contrôler la gestion des erreurs :
nulle, seul le numéro de l'erreur est affiché ;
positive, le message d'erreur est affiché ;
négative, déclenche ABORT ;
sa valeur initiale au démarrage du FORTH est positive.

MESSAGE
: MESSAGE (n --)
CR 00FF AND >R MSG BEGIN COUNT R < WHILE COUNT + REPEAT 1- COUNT R = WARNING @ AND IF COUNT TYPE SPACE ELSE DROP ." Msg:" R . THEN R>DROP ;
n étant le numéro de message, affiche son libellé s'il existe et si WARNING est positive, sinon affiche seulement le numéro de message ; les libellés des messages sont à l'adresse MSG, sous forme de chaînes de caractères avec octet de compte précédé d'un octet de numéro de message ; tous les messages sont concaténés dans l'ordre des numéros de message, dont le dernier a le numéro 255 ; pour le

libellé et la signification exacte de chaque message, voir l'annexe "Messages d'Erreur".

ERROR

```
: ERROR ( n -- )
WARNING @ 0< IF ABORT THEN HERE COUNT TYPE ."  ? " MESSAGE
SP! BLK @ -DUP IF IN @ SWAP WHERE THEN QUIT ;
```

appelle ABORT si WARNING est négative, sinon affiche le dernier mot isolé dans le flot d'entrée, suivi du message d'erreur correspondant ; si le flot d'entrée était en provenance du disque, appelle WHERE.

?ERROR

```
: ?ERROR ( b n -- )
SWAP IF ERROR ELSE DROP THEN ;
```

génère l'erreur n si b est vrai.

?STACK

```
: ?STACK ( -- )
DEPTH DUP 0< 1 ?ERROR 032 > 7 ?ERROR ;
```

génère l'erreur "pile vide" ou "pile pleine" s'il y a débordement de capacité de pile

?COMP

```
: ?COMP ( -- )
STATE @ 0= 011 ?ERROR ;
```

génère l'erreur "en définition seulement" si FORTH est en interprétation

?EXEC

```
: ?EXEC ( -- )
STATE @ 012 ?ERROR ;
```

génère l'erreur "hors définition seulement" si FORTH est en compilation

?PAIRS

```
: ?PAIRS ( n2 n1 -- )
- 013 ?ERROR ;
```

génère l'erreur "contrôles mal appairés" si n2 et n1, positionnés par les mots de construction des structures de contrôle, sont différents.

?LOADING

```
: ?LOADING ( -- )
BLK @ 0= 016 ?ERROR ;
```

génère l'erreur "sur disque seulement" si le flot d'entrée n'est pas un fichier.

18. LES ENTREES/SORTIES SERIE

Les entrées/sorties série standard se font à travers trois primitives vectorisées : ?TERMINAL KEY et EMIT.

?TERMINAL

```
DEFER ?TERMINAL ( -- b )
' ?TERM IS TERMINAL
```

vecteur testant généralement la présence d'un caractère à traiter en entrée ; il est initialisé à ?TERM (voir le vocabulaire IOS du chapitre "Interface avec le Moniteur TELESTRAT")

?TERMSTOP

```
: ?TERMSTOP ( -- b )
?TERMINAL DUP IF 2 0 DO DROP KEY DUP 3 = SWAP 01B = OR DUP
?LEAVE LOOP THEN ;
```

KEY

```
DEFER KEY ( -- c )
' KEY0 IS KEY
```

vecteur retournant généralement le premier caractère disponible en entrée ; il est initialisé à CKEY (voir le vocabulaire IOS du chapitre "Interface avec le Moniteur TELESTRAT")

EMIT

```
DEFER EMIT ( c -- )
' CEMIT IS EMIT
```

vecteur envoyant généralement un caractère en sortie ; il est initialisé à CEMIT (voir le vocabulaire IOS du chapitre "Interface avec le Moniteur TELESTRAT")

CR

```
: CR ( -- )
0D EMIT 0A EMIT ;
```

passe à la ligne (0D=carriage-return, 0A=line-feed).

CLS

```
: CLS ( -- )
0C EMIT ;
```

efface la (les) fenêtre(s) affectée(s) au canal de sortie courant

GOTOXY

```
: GOTOXY ( Xcolonne Yligne -- )
01F EMIT 040 OR EMIT 040 OR EMIT ;
```

positionne le curseur en (X, Y) dans la fenêtre affectée au canal courant

SPACE

```
: SPACE ( -- )
BL EMIT ;
```

envoie un espace sur la sortie courante

SPACES

```
: SPACES ( n -- )
0 MAX -DUP IF 0 DO SPACE LOOP THEN ;
```

envoie n espaces sur la sortie courante

-TRAILING

```
-TRAILING ( adr cnt -- adr cnt' )
DUP 0 DO 2DUP + 1- C@ BL - IF LEAVE ELSE 1- THEN LOOP ;
```

pour une chaîne de cnt caractères située à adr, retourne adr inchangée et cnt' la longueur de la chaîne sans ses espaces terminaux ("trailing spaces")

TYPE

```
: TYPE ( adr cnt -- )
DUP IF 0 DO COUNT EMIT LOOP ELSE 2DROP THEN ;
```

envoie sur la sortie courante par appels successifs à EMIT la chaîne de cnt caractères située à adr ; si cnt est nul, TYPE est sans effet.

(.")

: (.") (dans une définition seulement)
R> COUNT 2DUP + >R TYPE ;
lorsque ." est utilisé dans une définition, (.") est compilé suivi de la chaîne de caractères avec octet de compte ; (.") ne doit jamais être utilisé en direct.

."

: ." (--)
ASCII " STATE @ IF COMPILE (.") WORD HERE C@ 1+ ALLOT ELSE WORD HERE COUNT TYPE THEN ; IMMEDIATE
en mode interprétation, imprime la chaîne de caractères suivant ", délimitée par le premier caractère " (guillemet) trouvé ; en mode compilation, la chaîne de caractères est compilée dans le dictionnaire précédée de (."), qui imprimera la chaîne lors de l'exécution de la définition compilée ; attention : ." doit être suivi d'un espace, qui ne fera pas partie de la chaîne imprimée.

ENCLOSE

CODE ENCLOSE (adr c -- adr ndébut nfin ndernier)
2 SETUP, TXA, SEC, 8 # SBC, TAX, (place pour 4 val) 2H STY, 1H STY, N LDA, (trouver le premier non-délimiteur) BEGIN, BEGIN, N 2+)Y CMP, 0= IF, INY, 2SWAP 0= UNTIL, N 3 + INC, 3H INC, 2H INC, 1H INC, 2SWAP DROP JMP, (AGAIN,) THEN, 3L STY, BEGIN, 2L STY, N 2+)Y LDA, 0= IF, (délimiteur nul) 1L STY, TYA, 3L CMP, 0= IF, 2L INC, 0= IF, 2H INC, THEN, THEN, (pour X, le mot nul) NEXT, THEN, (sinon:) INY, 0= IF, N 3 + INC, 2H INC, 1H INC, THEN, N CMP, 0= UNTIL, 1L STY, NEXT, ;C
primitive de balayage : cherche à isoler une chaîne de caractères encadrée par le caractère délimiteur c, en commençant à l'adresse adr ; le caractère de code ASCII nul (toujours placé à la fin de chaque bloc ou du TIB) est toujours considéré comme délimiteur par défaut ; retourne adr inchangée, et trois déplacements par rapport à adr (offsets) : ndébut pour le caractère suivant le premier délimiteur c trouvé, nfin pour le second délimiteur c trouvé ; ndernier est égal à nfin+1 si le second délimiteur est égal au caractère c, sinon nfin si le second délimiteur est le caractère de code ASCII nul ; pratiquement, la chaîne se trouve de adr+ndébut inclue à adr+nfin exclue.

WORD : WORD (--)

BLK @ IF BLK @ BLOCK ELSE TIB @ THEN IN @ + SWAP ENCLOSE HERE 34 BLANKS IN +! OVER - DUP>R HERE C! + HERE 1+ R> CMOVE ;
transfère à HERE le mot suivant (délimité par des espaces avec ENCLOSE) du flot d'entrée courant : le tampon d'entrée série courant ("Terminal Input Buffer" ou TIB) si BLK est nul, sinon le bloc BLK du fichier courant.

ASCII

ASCII (-- c)
BL WORD HERE 1+ C@ [COMPILE] LITERAL ; IMMEDIATE
retourne le code ASCII du premier caractère du mot suivant ASCII (normalement un mot d'un seul caractère bien sûr), et le compile comme constante numérique si dans une définition en cours de compilation (voir LITERAL)

(

((--)
ASCII) WORD ; IMMEDIATE
balaye le flot d'entrée en ignorant son contenu jusqu'à trouver une parenthèse fermante ; sert à introduire des commentaires dans les sources

19. INTERFACE AVEC LE MONITEUR TELESTRAT

16 Koctets d'EPROM contiennent toutes les routines "moniteur" d'interface avec le "matériel" spécifique au TELESTRAT ; l'accès à ces routines a été normalisé : il suffit de passer les arguments soit par les registres du 6502, soit par des variables système d'adresse fixe en mémoire (la plupart du temps en page 0), et d'exécuter une instruction BRK suivie d'un numéro de fonction (pour des renseignements complets sur les numéros de fonction et les adresses système, se référer au "dossier développeur").
Pour les numéros de fonction calculés, trois instructions sont initialisées en RAM par le FORTH à l'adresse 560 hexa : une instruction BRK, un octet libre pour le numéro de fonction, et une instuction RTS ; ces instructions sont mises en place par STARTUP.

0560 LABEL BRK

AYX

0563 CONSTANT AYX

PIO

0566 CONSTANT PIO

MON

CODE MON (fun --)
1L LDA, BRK 1+ STA, INX, INX, X!, AYX LDA, AYX 1+ LDY, AYX 2+ LDX, PIO LSR, BRK JSR, AYX STA, AYX 1+ STY, AYX 2+ STX, PHP, PLA, PIO STA, X@, NEXT, ;C
prend le numéro de fonction sur la pile et passe les valeurs des registres dans la variable AYX (adresse 563 hexa, registre accumulateur en 563, registre Y en 564, registre X 565) et l'indicateur de retenue C dans le bit 0 de la variable PIO (adresse 566) ; après exécution de MON, les valeurs des registres sont retounées aux mêmes adresses, et l'indicateur d'état est retourné à l'adresse PIO (bit 7 à 0 : NV-BDIZC) ; pour exemples, voir les différentes interfaces avec le moniteur

MON:

: MON: (fun -- | --)
CREATE C, ;CODE X!, 2 # LDY, W)Y LDA, BRK 1+ STA, BRK JSR, X@, NEXT, ;C
définit une famille de mots appelant des fonctions moniteur sans argument ; pour les exemples, voir les différentes interfaces sans argument avec le moniteur

HRS:

: HRS: (narg fun -- | args ... -- | nargs+080 pour HIRES)
CREATE C, C, ;CODE 2 # LDY, W)Y LDA, BRK 1+ STA, INY, W

)Y LDA, .A ASL, TAY, BEGIN, 1L LDA, HRS1 2- ,Y STA, INX, 1L LDA, HRS1 1- ,Y STA, INX, DEY, DEY, 0= UNTIL, CS IF, FLGTEL BIT, 0< IF, VS NOT IF, 6 PICK 6 PICK THEN, X!, BRK JSR, X@, THEN, THEN, 2DROP NEXT, ;C
définit une famille de mots (graphiques et sonores) passant plusieurs (narg) mots arguments en HRS1 (adresse 4D) ; les mots graphiques ne devant être exécutés qu'en mode graphique (bit 7 à 1 de l'octet FLGTEL, adresse 20D) et ne devant pas être exécutés en mode "minitel" (bit 6 à 1 de l'octet FLGTEL), on les distingue des mots de gestion du PSG (synthétiseur) en mettant à 1 le bit 7 de narg

20. GESTION SONORE

Pour tout renseignement détaillé sur le générateur sonore, se procurer une documentation sur le circuit 8912

SOUNDS
FORTH DEFINITIONS VOCABULARY SOUNDS IMMEDIATE SOUNDS DEFINITIONS
vocabulaire contenant tous les mots de gestion sonore

PSG:
: PSG: (14params -- | --)
CREATE HERE 0E ALLOT HERE DO I 1- C! 1- +LOOP ;CODE X!, 2 # LDY, W)Y LDA, TAX, INY, W)Y LDA, TAY, 040 MON, X@, NEXT, ;C
définit des tables de 14 paramètres qui seront chargés à l'exécution dans les 14 registres du 8912

PSG!
: PSG! (n --)
AYX 2+ C! 041 MON ;
charge le registre A du 8912 avec la valeur n

les mots qui suivent se comportent de la même manière que leur homonyme BASIC ; se référer à la documentation BASIC du TELESTRAT pour plus de précision ; toutes les valeurs numériques sont données en hexadécimal

PLAY
4 43 HRS: PLAY

SOUND
3 44 HRS: SOUND

MUSIC
4 45 HRS: MUSIC

OUPS
42 MON: OUPS

ZAP
46 MON: ZAP

SHOOT
47 MON: SHOOT

EXPLODE
9C MON: EXPLODE

PING

9D MON: PING

21. GRAPHIQUES HAUTE RESOLUTION

Les instructions graphiques sont sans effet lorsque l'affichage n'est pas en mode haute résolution ou lorsque le TELESTRAT est en mode minitel (voir HRS: au chapitre Interface avec le Moniteur TELESTRAT)

GRAFX

FORTH DEFINITIONS HEX VOCABULARY GRAFX IMMEDIATE GRAFX DEFINITIONS
vocabulaire contenant tous les mots de gestion graphique

TEXT

19 MON: TEXT
force le mode texte : écran de 27 lignes (ordonnée Y de 0 à 26) de 40 caractères (abscisse X de 0 à 39)

HIRES

1A MON: HIRES
force le passage en mode haute résolution : écran de 200 lignes (ordonnée Y de 0 à 199) de 40 cellules de 6 pixels (abscisse X de 0 à 239), plus zone inférieure de 3 lignes de 40 caractères ; sert également à effacer l'écran haute résolution

PAPER

: PAPER (w couleur --)
080 OR SWAP AYX 2! 092 MON ;
change la couleur du papier ("background") ; les codes de couleur sont :
0 : noir ;
1 : rouge ;
2 : vert ;
3 : jaune ;
4 : bleu ;
5 : magenta ;
6 : cyan ;
7 : blanc.
voir également INK

INK

: INK (w couleur --)
080 OR SWAP AYX 2! 095 MON ;
change la couleur de l'encre ("foreground") ; lorsque PAPER ou INK sont utilisés dans une fenêtre, ses deux premières colonnes (les 12 premiers pixels en haute résolution) de chaque ligne sont occupés par les attributs de couleur, et donc inutilisables ; voir également PAPER

PATTERN

: PATTERN (motif --)
02AA C! ;
mémorise le motif de tracé des lignes en mode haute résolution ; voir la documentation BASIC

FB

CODE FB (fb --)
1L LDA, 6 # LDY, BEGIN, .A ASL, DEY, 0= UNTIL, 057 STA, POP, ;C
à la différence du BASIC qui demande systématiquement un paramètre FB (comme "Foreground" et "Background"), les instructions qui suivent prennent la dernière valeur fixée par le mot FB ; elles ont donc systématiquement un paramètre de moins que dans la documentation BASIC du TELESTRAT (voir également HRS: au chapitre "Interface avec le Moniteur TELESTRAT")

CURSET
82 90 HRS: CURSET (X Y --)
positionne le curseur aux coordonnées données

CURMOV
82 91 HRS: CURMOV (déplacementX déplacementY --)
déplace le curseur du nombre de pixels donné (par rapport à son ancienne position bien sûr)

ADRAW
84 8D HRS: ADRAW (départX départY arrivéeX arrivéeY --)
trace une ligne droite dont on donne les coordonnées absolues des extrémités

DRAW
82 8E HRS: DRAW (déplacementX déplacementY --)
trace une ligne droite, partant du point où se trouve le curseur, et dont on donne les coordonnées relatives du point d'arrivée par rapport au point de départ

CIRCLE
81 8F HRS: CIRCLE (rayon --)
trace un "cercle" (ovale si un téléviseur est utilisé en guise de moniteur car dans ce cas l'écran n'a pas le même nombre de pixels au centimètre en hauteur et en largeur) dont on donne le rayon, et dont le centre est le point où se trouve le curseur

BOX
82 94 HRS: BOX (dimensionX dimensionY --)
trace un rectangle, dont un coin se trouve à la position du curseur, et dont on donne les coordonnées relatives du coin opposé

ABOX
84 95 HRS: ABOX (Xcoin1 Ycoin1 Xcoin2 Ycoin2 --)
trace un rectangle dont on donne les coordonnées absolues de deux coins opposés

PAVE
83 96 HRS: PAVE (nbcell nblign motif --)
remplit nbcell cellules (de 6 pixels) sur nblig lignes avec motif, la première cellule étant celle où se trouve le curseur ; PAVE remplace l'instruction FILL du BASIC (pour éviter la confusion de l'utilisateur avec le FILL du FORTH servant à remplir une zone mémoire avec un caractère donné)

HEMIT
: HEMIT (c --)

AYX C! 097 MON ;
écrit le caractère de code ASCII c aux coordonnées courantes du curseur haute résolution

HTYPE
: HTYPE (adr cnt --)
SWAP AYX 2! 098 MON ;
écrit la chaîne de cnt caractères située à adr aux coordonnées courantes du curseur

ADCHAR
: ADCHAR (c -- adr)
AYX C! 01D MON AYX @ ;
retourne l'adresse de la définition matricielle (8 octets) du caractère de code ASCII c

CHAR:
: CHAR: (c 8octets -- | --)
<BUILDS HERE 9 ALLOT HERE DO I 1- C@ -1 +LOOP 0 C, DOES>
AYX ! 039 MON ;
définit une famille de mots de définition matricielle de caractères ; les 8 octets de la définition matricielle du caractère de code ASCII c sont pris sur la pile et stockés dans le dictionnaire ; à l'exécution, cette définition matricielle est mise en vigueur (pour tous les caractères de code ASCII c qui ont été ou seront affichés) ; utile pour faire varier commodément la forme d'un caractère (pour les fous du "packman")

?HIRES
: ?HIRES (-- b)
FLGTEL 080 CTST ;
b est vrai si le mode d'affichage est la haute résolution

HIMEM
: HIMEM (-- ad)
?HIRES IF 09800 ELSE 0B400 THEN ;
donne l'adresse limite que le vocabulaire FORTH ne doit pas dépasser

22. GESTION DES FENETRES

Le TELESTRAT permet l'utilisation simultanée de quatre "fenêtres" texte, qui peuvent être affectées à des canaux en sortie (comme tout autre driver de sortie série) ; une fenêtre est définie par son adresse de base (adresse mémoire écran du caractère qui correspondra à la ligne 0 et à la colonne 0, définissant un système de coordonnées local la fenêtre), et par ses quatre côtés (numéros de colonnes correspondant aux côtés gauche et droit, numéros de lignes correspondant aux côtés haut et bas, tous quatre en coordonnées locales à la fenêtre) ; chaque fenêtre gère son propre curseur et gère automatiquement le passage à la ligne en fin de ligne et le scroll en bas de page (uniquement pour les caractères à l'intérieur de la fenêtre bien sûr).

WINDOWS
FORTH DEFINITIONS HEX VOCABULARY WINDOWS IMMEDIATE WINDOWS DEFINITIONS

vocabulaire contenant tous les mots de gestion des fenêtres

WINDOW:

```
: WINDOW: ( dx fx dy fy -- | w -- )
<BUILDS ROT >R ROT C, R> C, ROT C, C, BB80 , DOES> AYX 2!
036 MON ;
```

définit une famille de mots mémorisant des définitions de fenêtres, ayant toutes pour adresse de base le caractère en haut à gauche de l'écran, chacune ayant pour première et dernière colonne dx et fx, et pour première et dernière ligne dy et fy ; l'exécution d'une fenêtre remplace la définition de la fenêtre dont le numéro est sur la pile

SCRW

```
DECIMAL 0 39 1 27 WINDOW: SCRW 0 SCRW
```

c'est la fenêtre 0 par défaut à l'initialisation du FORTH, prenant tout l'écran sauf la ligne 0 ; à l'initialisation du FORTH, le vecteur EMIT exécute CEMIT qui envoie les caractères sur le canal 0, le canal de sortie par défaut, qui a pour seul driver de sortie celui de la fenêtre 0, initialisée à SCRW

MSGW

```
DECIMAL 0 39 0 0 WINDOW: MSGW
```

c'est une fenêtre qui occupe toute la ligne 0 (pour l'envoi de messages minitel par exemple)

CURSOR

```
CODE CURSOR ( w b -- )
X!, 1L LSR, 1H LDA, TAX, CS IF, 035 MON, ELSE, 034 MON,
THEN, X@, POPTWO, ;C
```

si b est vrai affiche, sinon éteint, le curseur de la fenêtre w

SCROLL

```
CODE SCROLL ( ligde ligà b -- )
( vérifier si numéro de fenêtre = numéro de canal !)  PIO
2+  LDA, 028 STA, 3 SETUP, X!, N 4 + LDX, N 2+ STY, N LDA,
0= IF, 038 MON, ELSE, 037 MON, THEN, X@, NEXT, ;C
```

scrolle de la ligne ligde à la ligne ligà de la fenêtre du canal de sortie courant, d'une ligne vers le haut si b est vrai, sinon vers le bas

23. GESTION DE L'HORLOGE

Le TELESTRAT est équipé d'un VIA qui génère une interruption toutes les 100 millisecondes ; c'est à cette occasion qu'il va scruter le clavier, incrémenter l'horloge (et au besoin l'afficher à l'écran sur 6 caractères HH:MM:SS), et décrémenter deux minuteries, l'une en dixièmes de secondes, l'autre en secondes

```
FORTH DEFINITIONS HEX
```

CLOCK

```
: CLOCK
CASE  ( 0 -- | close ) 0 OF 03D MON ENDOF ( col lig 1 -- )
1 OF 028 * + BB80 + AYX ! 03E MON ENDOF ( h mn se 2 -- ) 2
OF 0211 C! 0212 C! 0213 C! ENDOF ENDCASE ;
```

remplit trois fonctions suivant la valeur au sommet de la pile :
2 initialise l'heure à h heures, mn minutes et se secondes ;
1 affiche l'horloge à l'écran sur 6 caractères à la colonne col et à la ligne lig ;
0 arrête le rafraîchissement de l'horloge à l'écran.

WAIT

```
: WAIT
CASE ( h mn 0 -- ) 0 OF SWAP BEGIN DUP  0213  C@  =  UNTIL
DROP  BEGIN DUP 0212 C@ = UNTIL DROP ENDOF ( sec  1 -- ) 1
OF 042 ! BEGIN 042 @ 0= UNTIL ENDOF ( dsec 2 -- ) 2 OF 044
! BEGIN 044 @ 0= UNTIL ENDOF ENDCASE ;
```

remplit trois fonctions suivant la valeur au sommet de la pile :
2 attend dsec dixièmes de secondes
1 attend sec secondes ;
0 attend la minute mn de l'heure h (ne pas oublier dans ce cas de régler d'abord l'horloge avec la fonction 2 de CLOCK).

24. GESTION DES CANAUX D'ENTREES/SORTIES

La gestion de tous les canaux d'entrée/sortie se fait par des appels au moniteur, qui passe à travers des "drivers".

TERMINAL

```
: TERMINAL ( -- )
WINDOWS 0 SCRW IOS 0 0 OPCH ' ?TERM IS ?TERMINAL ' CKEY IS
KEY 0 INPUT ' CEMIT IS EMIT 0 OUTPUT ;
```

(ré)initialise la fenêtre 0 à SCRW sur le canal 0, les vecteurs d'entrée/sortie, avec 0 comme numéro de canal courant d'entrée et de sortie.

IOS

```
FORTH DEFINITIONS HEX VOCABULARY IOS IOS DEFINITIONS
```

vocabulaire contenant tous les mots de gestion des canaux d'entrée/sortie

OPCH

```
: OPCH ( ch io -- )
080 OR AYX C! 3 AND MON ;
```

ouvre sur le canal ch le driver de l'entrée/sortie io

CLCH

```
: CLCH ( ch io -- )
080 OR AYX C! 3 AND 4 + MON ;
```

ferme sur le canal ch le driver de l'entrée/sortie io

?TERM

```
CODE ?TERM ( -- b )
X!, 0 # LDX, 56 MON,  CS  NOT  IF,  INY,  THEN,  TYA,  X@,
PUSH0A, ;C
```

retourne une valeur booléenne, vraie si un caractère est disponible dans le tampon d'entrée du clavier, faux si le tampon est vide.

CKEY

```
CODE CKEY ( -- c )
```

```
      PIO 1+ LDA, 0C # ORA, BRK 1+ STA, BRK JSR, PUSH0A, ;C
      retourne  c,  le premier caractère disponible sur le canal
      d'entrée courant, dont le numéro est stocké dans le second
      octet  de  la  variable  PIO  (I comme "Input") ; utilisez
      INPUT pour changer le numéro du canal d'entrée courant.

INPUT
      : INPUT ( ch -- )
      PIO 1+ C! ;
      change le numéro du canal d'entrée courant pour CKEY

CEMIT
      CODE CEMIT ( c -- )
      PIO 1+ LDA, TAY, 10 # ORA, BRK 1+ STA, 1L  LDA,  BRK  JSR,
      220 ,Y LDA, ' OUT LDY, UP )Y STA, POP, ;C
      envoie le caractère c sur le canal de sortie courant, dont
      le numéro  est  stocké  dans  le  troisième  octet  de  la
      variable  PIO  (O  comme  "Output") ; utilisez OUTPUT pour
      changer le numéro du canal de sortie courant.

OUTPUT
      : OUTPUT ( ch -- )
      PIO 2+ C! ;
      change le numéro du canal de sortie courant pour CEMIT

KBD:
      : KBD: ( n -- | -- )
      CREATE C, ;CODE X!, 2 # LDY, W )Y LDA, 052 MON, X@,  NEXT,
      ;C
      définit  une famille de mots permettant de changer le type
      du clavier

QWERTY
      0 KBD: QWERTY ( -- )

AZERTY
      1 KBD: AZERTY ( -- )

FRENCH
      2 KBD: FRENCH ( -- )

ACCENT
      4 KBD: ACCENT ( -- )

      ce sont les différents types de clavier  prédéfinis  ;  se
      reporter  à  la documentation BASIC ; ACCENT fait basculer
      entre les types de clavier avec ou sans accent

TSTLP
      01E MON: TSTLP ( -- )
      teste la présence de l'imprimante

PEMIT
      : PEMIT ( c -- )
      AYX C! 048 MON ;
      envoie le caractèrc c directement sur l'imprimante

VCOPY
      048 MON: VCOPY ( -- )
      hardcopy d'écran lorsque le TELESTRAT est en mode videotex
```

HCOPY

04C MON: HCOPY
hardcopy d'écran lorsque le TELESTRAT est en mode haute résolution

WCOPY

: WCOPY (w --)
028 C! 04A MON ;
hardcopy d'écran lorsque le TELESTRAT est en mode texte ;
w est le numéro de la fenêtre qui doit être imprimée

SEMIT

: SEMIT (c --)
AYX C! 067 MON ;
envoie un caractère directement sur la sortie série RS232

SDUMP

05C MON: SDUMP (--)
affiche à l'écran en hexadécimal les codes ASCII des caractères reçus par l'entrée RS232

CONSOLE

05D MON: CONSOLE (--)
transforme le TELESTRAT en terminal : les caractères saisis au clavier sont envoyés en sortie sur la RS232, les caractères entrant par la RS232 sont affichés à l'écran

SLOAD

: SLOAD (-- org fin)
PIO 1 CRST 05E MON DESALO @ FISALO @ ;

SLOADA

: SLOADA (org -- fin)
PIO 1 CSET DESALO ! 05E MON FISALO @ ;

SSAVE

: SSAVE (org fin --)
0 SALO! 05F MON ;

transfert de fichiers par la RS232 ;
SLOAD reçoit des fichiers avec un entête ayant la structure suivante :
- synchro : au moins 10 octets ASCII 16 hexa, puis 1 octet 24 hexa
- titre : une chaîne de caractères, terminée par un octet nul
- type : un octet 40 hexa pour chargement mémoire sans démarrage auto
- org, fin, boot : 3 adresses de 2 octets (poids faible en premier)
- réserve : 1 octet quelconque
- corps : les octets utiles (il doit y en avoir fin-org+1)
- réserve : 1 octet quelconque
SLOADA reçoit des fichiers sans entête, et charge le premier octet à org ; quitter en tapant <CTRL>+<C> ; SSAVE envoie le contenu d'une zone mémoire sur la RS232

MEMIT

MEMIT (c --)
AYX C! 067 MON ;
envoie un caractère directement sur la liaison minitel

MLOAD

```
: MLOAD ( -- org fin )
PIO 1 CRST 060 MON DESALO @ FISALO @ ;
```

MLOADA

```
: MLOADA ( org -- fin )
PIO 1 CSET DESALO ! 060 MON FISALO @ ;
```

MSAVE

```
: MSAVE ( org fin -- )
0 SALO! 061 MON ;
```

ce sont les homologues exacts dc SLOAD, SLOADA et SSAVE pour transfert de fichiers par liaison minitel

MNTL

```
CODE MNTL ( fun -- )
3 # LDA, BNKCIB STA, 1L LDA, BNKCIB 2- STA, DEY, BNKCIB 1-
STY, X!, AYX LDA, AYX 1+ LDY, AYX 2+ LDX, PIO LSR, 040C
JSR, AYX STA, AYX 1+ STY, AYX 2+ STX, PHP, PLA, PIO STA,
X@, POP, ;C
```

primitive d'appel de fonctions TELEMATIC (banque mémoire numéro 3)

MINITEL

```
DEFER MINITEL

L: MNTL0 ] MINITEL [ HERE 2+ , HERE 2+ , ASSEMBLER RTS,
L: MNTL1 X@, MNTL0 0 0100 U/ # LDA, IP 1+ STA, # LDA, IP
STA, NEXT, FORTH
```

SERVEUR

```
SERVEUR ( n -- )
04000 PAD U< 0C ?ERROR ( arborescence en 4000h ) 0321 C@ 7
AND 02FD C! MNTL1 02FE ! AYX 2+ C! 0C5 MNTL ;
```

L'interface avec l'application TELEMATIC est plus souple d'emploi en FORTH qu'en BASIC : l'initialisation du serveur et le traitement des "pages langage" (en rouge, ou sous-intensité sur un moniteur monochrome, dans l'arborescence) se font separément.

La partie initialisation peut être installée dans le fichier de démarrage BONJOUR.DAT : chargement en mémoire de l'arborescence et du programme de traitement des "pages langage" (attention : l'arborescence démarre en 4000 hexa), qui sera au mieux un "relogeable" dont le dernier mot (exécuté automatiquement au chargement) procèdera à l'installation du vecteur MINITEL (avec IOS ' <traitement-pages> IS MINITEL) et se terminera par l'appel à TELEMATIC par le mot SERVEUR.

Le traitement des pages se fait à travers le vecteur MINITEL ; la sélection de la page se fait aisément grâce une construction CASE avec un OF" spécialisé dans la comparaison de chaînes ; les écritures se font facilement soit sur l'écran de contrôle par le canal 0 (avec 0 OUTPUT), soit sur l'écran du minitel par le canal 1 (avec 1 OUTPUT) ; les saisies au clavier du minitel se font à travers TINPUT ; lorsque le traitement est terminé, le retour se fait automatiqucment à TELEMATIC.

PAGE

: PAGE (-- ad)
09CC1 7 -TRAILING HERE C! HERE COUNT CMOVE HERE ;
retourne l'adresse où se trouve le nom de la page à traiter (commençant par un octet de compte), pour utilisation par OF" (qui fonctionne comme le OF classique, mais en comparant des chaînes de caractères) ; la sélection de l'action en fonction du nom de la page est aussi facile que l'emploi de la construction CASE :
: TRAITEMENT-PAGES (--)
CASE PAGE OF" PAGE12" ... ENDOF OF" PAGE27" ... ENDOF ... ENDCASE ;
bien que parfaitement adaptée à cette fonction, l'instruction OF" est générale et attend sur la pile l'adresse d'une chaîne commençant par un octet de compte.

TINPUT
: TINPUT (n -- ad code)
060 C! 0BC MNTL 061 @ 00FF C@ ;
effectue la saisie d'un maximum de n caractères au clavier du minitel ; retourne l'adresse où se trouve la chaîne de caractèrcs saisie, ainsi que le code de la touche de fonction minitel qui a conclu la saisie ; voir le manuel TELEMATIC pour les codes des touches.

APPLIC
: APPLIC (n --)
04D ! 0BF MNTL ;
passe le paramètre n à l'application résidente en banque mémoire 3 ; dans le cas de la cartouche TELEMATIC, voir le manuel.

RING
062 MON: RING (--)

WCXFIN
063 MON: WCXFIN (--)

LIGNE
064 MON: LIGNE (--)

DECON
065 MON: DECON (--)

instructions contrôlant directement la ligne minitel ; voir les manuels BASIC et TELEMATIC ; permettent de générer des applications télématiques sans le support (commode mais pas universel) de la cartouche TELEMATIC.

En résumé, voici comment monter un micro-serveur FORTH+TELEMATIC, démarrant automatiquement à la mise sous tension du TELESTRAT (ou redémarrant après une coupure secteur) :

- créez une arborescence (voir manuel TELEMATIC)
- créez un mot pour chaque "page langage" à traiter par FORTH ; TINPUT permettra de gérer les entrées (au clavier du minitel connecté) ; 1 OUTPUT redirigera les sorties sur l'écran du minitel connecté, 0 OUTPUT redirigera les sorties sur l'écran du TELESTRAT (pour "tracer" le fonctionnement du serveur par exemple)
- créez un mot de "sélection de page" avec la construction

CASE+OF, appelant pour chaque nom de "page langage" le mot correspondant
- créez un dernier mot d'initialisation affectant au vecteur MINITEL le mot de sélection de page, puis appelant avec SERVEUR la cartouche TELEMATIC
- sauvegardez le tout sous forme d'un module relogeable (dont le dernier mot défini doit être celui d'initialisation)
- créez le fichier de démarrage FORTH.DAT (dont l'écran 1 sera interprété au démarrage du FORTH à la mise sous tension du TELESTRAT), qui doit charger l'arborescence puis le module relogeable, dont le dernier mot défini sera automatiquement exécuté, c'est-à-dire lancera le micro-serveur

A vos serveurs !

25. STRATSED le DOS

FORTH considère la mémoire de masse comme une mémoire virtuelle (avec gestion d'une batterie de tampons).
TELE-FORTH, comme la plupart des versions modernes de FORTH, passe par le DOS ("Disc Operating System" ou "Système d'Exploitation de Disque", d'où le nom STRATSED donné au DOS du TELESTRAT) pour accéder à la mémoire de masse ; les fichiers FORTH sont vus par le programmeur comme des fichiers à accès direct avec des enregistrements ("blocs" en FORTH) de 512 octets (voir BLOCK).

Les mnémoniques qui suivent ne sont donnés que dans un but de clarté :

```
06 EQU ADTMP ( adresse de travail )
08 EQU ADPS ( adresse table piste/secteur )
80 EQU BUFP ( pointeur sur tampon FORTH pour lecture/écriture )
417 EQU BNKCIB ( banque mémoire cible )
503 EQU RWBUF ( pointeur sur tampon DOS pour lecture/écriture )
512 EQU ERRNB ( numéro d'erreur DOS )
517 EQU BUFNOM ( numéro lecteur, nom fichier et extension )
528 EQU VSALO ( drapeaux pour lecture/écriture )
52C EQU FTYPE ( 8 pour les fichiers à accès direct )
52D EQU DESALO ( adresse début pour lecture/écriture )
52F EQU FISALO ( adresse fin pour lecture/écriture )
542 EQU TAMPFC ( tampon de travail )
548 EQU FICNUM ( numéro logique, 1 pour FORTH )
549 EQU NBFIC ( nombre de fichiers ouvrables à la fois )

1A EQU XOPEN ( ouverture de fichier )
1D EQU XCLOSE ( fermeture de fichier )

(DOS)
CODE (DOS) ( fun -- err )
FLGTEL LDA, .A LSR, CS IF, 08 # LDA, ELSE, ERRNB STY, BNKCIB DUP STY, 1L LDA, BNKCIB 2- STA, DEY, BNKCIB 1- STY, X!, PHP, TSX, DEX, DEX, DEX, DEX, 0513 STX, AYX LDA, AYX 1+ LDY, AYX 2+ LDX, PIO LSR, 040C JSR, AYX STA, AYX 1+ STY, AYX 2+ STX, PHP, PLA, PIO STA, PLP, X@, ERRNB LDA,
```

THEN, 1L STA, NEXT, ;C
primitive d'appel au DOS : le bit 0 de FLGTEL est à 0 si le DOS a été chargé (dans la banque mémoire 0) sinon il faut retourner un code d'erreur 8 ; pour d'obscures raisons, il faut préserver le registre d'état et conserver en 0513 la valeur du pointeur de pile, diminuée de 4, avant d'appeler dans la banque mémoire du DOS l'adresse de la fonction à exécuter, dont la forme est FFxx hexa, avec xx=fun (Y est toujours nul à l'entrée d'une primitive) ; un code différent de zéro est retourné dans ERRNB s'il y a eu erreur ; comme pour MON, les registres sont passés et retournés à travers les variables AYX et PIO

DOS
: DOS (fun --)
(DOS) DUP 080 OR ?ERROR ;
DOS traite l'éventuelle erreur retournée par (DOS) ; les numéros d'erreurs DOS commencent à 128 (voir ?ERROR et MESSAGE)

INIT
: INIT (--)
05C DOS ;
comme en BASIC, initialise une disquette

FILENAME
: FILENAME (ad --)
COUNT SWAP AYX 2! 024 MON ;
la chaîne de caractères située à ad est analysée comme un nom de fichier (avec lecteur et extension par défaut) et le résultat est stocké à BUFNOM (comme dans .FILE)

DIR
: DIR (--)
BL WORD HERE FILENAME 056 DOS ;
fonctionne comme en BASIC, mais il faut toujours préciser une sélection (DIR *.* pour lister le répertoire de tous les fichiers)

DEL
: DEL (--)
BL WORD HERE FILENAME 04D DOS ;
fonctionne comme en BASIC

COPY
: COPY (--)
BL WORD HERE FILENAME BUFNOM 0100 0D CMOVE BL WORD HERE FILENAME BUFNOM 010D 0D CMOVE 080 VSALO ! 038 DOS ;
homologue de son homonyme BASIC, dans sa seule version "Overwrite"
COPY TOTO.DAT TUTU.DAT

SALO!
: SALO! (DESALO FISALO VSALO --)
VSALO ! FISALO ! DESALO ! ;
n'a pas d'autre prétention que d'initialiser ces trois variables système

$SAVE
: $SAVE (org fin --)
0 SALO! 040 FTYPE C! 06B DOS ;

sauve le contenu de la mémoire des adresses org à fin (incluses), dans un fichier d'extension .COM dont le nom doit se trouver dans BUFNOM (après utilisation de FILENAME par exemple)

$LOAD

```
: $LOAD ( org -- fin )
0 08080 SALO! 062 DOS FISALO @ ;
```

charge le contenu d'un fichier en mémoire en partant de l'adresse org ; le nom du fichier doit être dans BUFNOM comme pour $SAVE ; l'adresse qui suit le dernier octet chargé est retounée

?FILE

```
: ?FILE ( ad -- b )
FILENAME  AYX 2+ 082 CTST 089 ?ERROR 07D DOS PIO 2 CTST 0=
;
```

si la chaîne de caractères située à ad correspond à un fichier existant, b est retourné vrai sinon faux (sans joker, sinon une erreur "nom de fichier incorrect" est générée)

.FILE

```
: .FILE ( -- )
CR ." Using " FICNUM C@ 1- 1 AND 0260 * 0B40 + COUNT ASCII
A + EMIT ." -" DUP 9 -TRAILING TYPE ." ." 9 + 3  -TRAILING
TYPE SPACE ;
```

affiche le nom complet du fichier dont le numéro logique (1 ou 2) est dans FICNUM

```
.FILE <RETURN>  Using A-TOTO.DAT ok.0
```

XFILE

```
: XFILE ( -- )
FLUSH FICNUM 3 TOGGLE ;
```

TELE-FORTH supporte deux blocs de contrôle de fichier : deux fichiers peuvent être ouverts en même temps (principalement pour faciliter la copie de blocs de fichier à fichier, voir SCRCOPY au chapitre "Editeur de Lignes") ; XFILE permet de basculer d'un bloc de contrôle à l'autre : il n'y a qu'un seul fichier courant à tout moment ; utiliscr .FILE pour savoir quel est le fichier courant

?OK

```
: ?OK ( -- b )
BEGIN ."  OK? " KEY CASE 0D OF 1 ;S ENDOF 1B OF 0 ;S ENDOF
3 OF QUIT ENDOF CR ."  RETURN=oui  ESC=non  CTRL+C=stop  "
ENDCASE AGAIN ;
```

n'autorise que trois réponses à l'invite "OK?" : <RETURN> retourne un booléen vrai, <ESC> un faux, et <CTRL>+<C> provoque l'abandon et le retour à la boucle de base de l'interprète ; voir USING ci-dessous et SCRCOPY au chapitre "Editeur de Lignes".

USING

```
: USING ( -- )
FLUSH  8  FTYPE C! BL WORD HERE ?FILE IF XCLOSE (DOS) DROP
XOPEN DOS ELSE 081 MESSAGE CR ." Creer"  HERE  COUNT  TYPE
?OK  IF  XCLOSE (DOS) DROP 0 DESALO ! B/BUF FISALO ! XOPEN
DOS FSTR ADTMP @ DUP RWBUF ! FLEN CMOVE ADPS @ 0C + @  AYX
! 08F DOS THEN THEN .FILE ;
```

permet de changer le fichier courant ; s'il n'existe pas, il peut être créé :
USING TUTU.DAT <RETURN> Using A-TUTU.DAT ok.0
USING TITI.DAT <RETURN> Fichier inexistant Creer TITI.DAT OK?
<ESC> Using A-TUTU.DAT ok.0

DRIVE

```
: DRIVE ( drv -- )
3 AND 0208 OVER + C@ 0= 08A ?ERROR 0500 C! ;
```

change le lecteur par défaut ; si le lecteur demandé n'est pas connecté, un message "lecteur absent" est généré

ST-R/W

```
: ST-R/W ( bufad secteur piste r/w -- )
>R 0501 C! 0502 C! RWBUF ! R> IF 0A1 ELSE 08C THEN DOS ;
```

homologue de R/W, mais à la place du numéro logique de bloc, on donne les numéros physiques de piste et de secteur, n'utilisez cet outil qu'en connaissance de cause...

ASSEMBLER HEX

```
L: FSTR 82 C, 80 C, 41 C, 24 C, 2020 , 2020 , 2020 , ( A$ ) 82 C, 80 C, 42 C, 24 C, 2020 , 2020 , 2020 , ( B$ ) 82 C, 80 C, 43 C, 24 C, 2020 , 2020 , 2020 , ( C$ ) 82 C, 80 C, 44 C, 24 C, 2020 , 2020 , 2020 , ( D$ ) FF C, HERE FSTR - EQU FLEN ( 41 octets de structure de champs )
L: FRW 80 # LDY, ( 128 octets par champ ) 51 LDA, 0< IF, ( write ) 60 STY, BUFP LDA, 61 STA, BUFP 1+ LDA, 62 STA, ELSE, ( read ) DEY, BEGIN, 61 )Y LDA, BUFP )Y STA, DEY, 0= UNTIL, THEN, CLC, BUFP LDA, 80 # ADC, BUFP STA, CS IF, BUFP 1+ INC, THEN, RTS, ( BUFP est prêt pour le champ suivant )
```

FORTH

primitive interne de lecture/écriture d'un enregistrement de 512 octets ; la conception du DOS a été entièrement orientée en fonction des besoins du BASIC : c'est la raison pour laquelle une structure de champ est nécessaire, bien que complètement artificielle pour FORTH (si vous observez la structure d'un fichier FORTH au moyen de l'utilitaire TELESTRAT de configuration de fichier, vous constaterez que les enregistrements sont constitués de quatre champs de 128 octets nommés A$, B$, C$ et D$) ; reportez-vous la documentation développeur pour de plus amples explications.

di

```
CODE di ( -- )
SEI, NEXT, ;C
```

ei

```
CODE ei ( -- )
CLI, NEXT, ;C
```

di ("Disable Interrupts") interdit les interruptions, ei ("Enable Interrupts") les autorise

B/BUF

1024 CONSTANT B/BUF
nombre d'octets par tampon ("Bytes/BUFfer")

B/SCR

1 CONSTANT B/SCR
nombre de tampons par écran ("Buffers/SCReen") de 1 Ko

R/W

: R/W (bufad bloc r/w --)
IF 020 ELSE 023 THEN >R OVER BUFP ! 1+ 2* 1+ DUP 2- di DO I DESALO ! J (DOS) 0F = IF DUP B/BUF BLANKS LEAVE THEN LOOP ei R>DROP DROP ;
interface de base pour lecture/écriture sur disque d'un enregistrement ; à bufad se trouve le tampon FORTH affecté à l'enregistrement numéro bloc ; si r/w est vrai, l'enregistrement doit être lu, sinon écrit ; si l'enregistrement n'existe pas sur disque, le tampon est rempli d'espaces.

FIRST

4096 CONSTANT FIRST
adresse de début de la zone des tampons

LIMIT

FIRST B/BUF 2+ 2+ + CONSTANT LIMIT
adresse de fin de la zone des tampons, chaque tampon est précédé de deux octets qui contiennent le numéro de l'enregistrement contenu dans le tampon (le bit 15 est mis à 1 si l'enregistrement, suite à une modification, doit être écrit sur disque avant de libérer le tampon), et suivi de deux octets nuls afin de repérer la fin du tampon (voir "le mot nul")

EMPTY-BUFFERS

: EMPTY-BUFFERS (--)
FIRST LIMIT OVER - ERASE ;
efface le contenu des tampons sans autre forne de procès ; ne pas abuser...

UPDATE

: UPDATE (--)
FIRST 1+ 080 CSET ;
marque le tampon "modifié"

FLUSH

: FLUSH (--)
FIRST @ 0< IF FIRST 1+ 080 CRST FIRST WCOUNT 0 R/W THEN ;
sauvegarde sur disque le tampon s'il est marqué "modifié"

BLOCK

: BLOCK (bloc -- ad)
FIRST @ OVER XOR 2* IF FLUSH DUP FIRST ! FIRST WCOUNT 1 R/W THEN DROP FIRST 2+ ;
retourne l'adresse ad du tampon contenant l'enregistrement bloc ; si l'unique tampon ne contenait pas déjà cet enregistrement, l'ancien enregistrement qu'il contenait est sauvegardé sur disque si marqué "modifié", puis l'enregistrement requis est chargé du disque dans le tampon.

LOAD

: LOAD (scr --)
BLK @ >R IN @ >R 0 IN ! B/SCR * BLK ! INTERPRET R> IN ! R> BLK ! ;
lance l'interpréteur avec pour flot d'entrée l'écran scr du fichier courant.

-->
: --> (--)
?LOADING 0 IN ! B/SCR BLK @ OVER MOD - BLK +! ; IMMEDIATE
l'interpréteur FORTH s'arrête normalement en fin de bloc (grâce aux deux octets nuls disposés à la suite de chaque tampon disque) ; --> permet de passer à l'interprétation du bloc suivant avant d'atteindre la fin du bloc.

THRU
: THRU (scr1 scrN --)
1 + SWAP DO I LOAD LOOP ;
charge les écrans scr1 à scrN par appels successifs à LOAD

LOAD-USING
: LOAD-USING (scr --)
XFILE USING LOAD XFILE .FILE ;
interprète l'écran scr du fichier dont le nom suit LOAD-USING dans le flot d'entrée, puis retourne au fichier courant (ex : 1 LOAD-USING TOTO.DAT)

THRU-USING
: THRU-USING (scr1 scrN --)
THRU-USING est à THRU ce que LOAD-USING est à LOAD

\
: \ (--)
?LOADING C/L IN @ OVER MOD - IN +! ; IMMEDIATE
en chargement, ignore la fin de la ligne ; un autre outil que les parenthèses pour introduire des commentaires dans les sources.

C/L
040 CONSTANT C/L
nombre de caractères par ligne, conventionnellement 64 ; permet de découper les écrans de 1 Ko en 16 lignes de 64 caractères.

(LINE)
: (LINE) (lig scr -- ad cnt)
>R C/L B/BUF */MOD R> B/SCR * + BLOCK + C/L ;
retourne un compte de 64 caractères et l'adresse de début de la ligne lig de l'écran scr

.LINE
: .LINE (lig scr --)
(LINE) -TRAILING TYPE ;
imprime la ligne lig de l'écran scr du fichier courant.

LIST
: LIST (scr --)
CR DUP SCR ! ." SCR #" . 010 0 DO CR I 2 .R SPACE I SCR @ .LINE ?TERMSTOP ?LEAVE LOOP CR ;
imprime l'écran scr sous forme de 16 lignes de 64 caractères, précédées de numéros de lignes ; le numéro de l'écran est mémorisé dans la variable SCR

TRIAD

```
: TRIAD ( scr -- )
CLS  3  /  3 * 3 OVER + SWAP DO CR I LIST ?TERMSTOP ?LEAVE
LOOP CR 0F MESSAGE CR ;
```

affiche trois écrans dont le premier a pour numéro le premier multiple de trois inférieur ou égal à scr ; correspond normalement à une page de listing imprimante.

INDEX

```
: INDEX ( prem dern -- )
CLS 1+ SWAP DO CR I 3 .R SPACE 0 I .LINE ?TERMSTOP  ?LEAVE
LOOP ;
```

imprime la ligne 0 de tous les écrans compris entre prem et dern inclus ; habituellement, la ligne 0 de chaque écran est utilisée pour décrire le contenu de l'écran.

26. GESTION DE MODULES RELOGEABLES

Un "relogeable" (ou "overlay") est un fichier sur disque qui contient du code FORTH déjà compilé qui peut être chargé à une adresse différente de celle où il a été créé ; les intérêts sont multiples :

- le temps nécessaire au chargement d'un relogeable est négligeable par rapport au temps qu'il faudrait pour compiler le code source équivalent
- il est facile de charger un relogeable artificiellement haut en mémoire pour pouvoir après utilisation le "déloger" et récupérer l'espace mémoire correspondant, sans avoir eu à consommer un octet de l'espace dictionnaire
- des applications peuvent être distribuées sous cette forme sans avoir à fournir les sources ; il est même possible d'empêcher l'emploi du décompilateur, car le dernier mot défini dans un relogeable est automatiquement exécuté lors du chargement du relogeable : il est alors possible de rentrer dans une boucle qui ne redonne jamais la main à l'interprète FORTH
- des programmes de très grosse taille peuvent être découpés en plusieurs relogeables se remplaçant mutuellement pour partager le même espace mémoire à des moments différents

Un exemple à la fin du chapitre montre comment générer puis utiliser un relogeable.

OV6502

```
: OV6502 ( -- )
HERE DUP MINUS 0FF AND DUP ALLOT BLANKS ;
```

sur le 6502, les relogeables ne peuvent être implémentés simplement qu'à condition d'avoir une adresse de base cadrée sur un début de page mémoire ; OV6502 est à exécuter avant la première définition devant faire partie du relogeable.

OVSAVE

```
: OVSAVE ( -- | OVSAVE <mot> <fichier> )
BL WORD HERE CONTEXT @ @ (FIND) 0= 5 ?ERROR DROP  DUP  NFA
HERE  ROT  LFA  @ (FIND) 0= 0A ?ERROR DROP NFA SWAP ( nfa'
nfa ) OVER 0FF AND OVER 0FF AND OR 0B  ?ERROR  HERE  DUP>R
OVER - ( len ) R OVER 8 / 1+ DUP ALLOT ERASE LATEST , 2DUP
```

```
( org len ) , , 0 DO COUNT ROT COUNT ROT XOR ( ad ad' b  ;
vrai: adresse haute )  IF J I 1- BMAP CSET 1+ SWAP 1+ 2
ELSE SWAP 1 THEN +LOOP 2DROP BL WORD HERE FILENAME HERE 2-
@ HERE $SAVE R> DP ! ;
```

pour créer un relogeable, il suffit de repérer les adresses dont le contenu dépend de l'adresse origine du module ; pour cela il suffit de compiler le même code source deux fois (erreurs courantes : "OVSAVE: compiler deux fois" ou "OVSAVE: utiliser OV6502"), et de noter les différences dans une "bitmap" (où chaque bit représente un octet), et de sauvegarder le tout avec l'adresse origine à laquelle a été compilé le module, sa longueur, et où se trouve le mot à exécuter au chargement

(OVLOAD)

```
: (OVLOAD) ( ad -- )
?FILE 0= 081 ?ERROR OV6502 HERE $LOAD ( début -- fin ) 6 -
WCOUNT SWAP WCOUNT SWAP @ ( latest' len' org' ) HERE  SWAP
- >R DUP HERE + OVER 0 DO DUP I BMAP CTST IF J HERE I + +!
2 ELSE 1 THEN +LOOP DROP LATEST HERE PFA LFA !  ( latest'
len' ) ALLOT R> + DUP CURRENT @ ! PFA CFA EXECUTE ;
```

le relogeable, dont le nom est à ad, est chargé à HERE (voir OV6502), puis la différence avec son ancienne adresse d'origine est ajoutée à tous les mots repérés dans la bitmap ; enfin, le premier et le dernier mot du relogeable sont liés au vocabulaire CURRENT, le pointeur de dictionnaire avancé (on se débarrasse de la bitmap), et le dernier mot est exécuté.

OVLOAD

```
: OVLOAD ( -- | OVLOAD <fichier> )
BL WORD HERE (OVLOAD) ;
```

c'est la forme la plus couramment utilisée, $OVLOAD ne servant que dans les définitions

(+OVLOAD)

```
: (+OVLOAD) ( ad -- )
DUP ?FILE 0= IF COUNT TYPE 081 MESSAGE QUIT THEN ." Here="
HERE 6 + U. ."  Himem=" HIMEM U.  CR ." Ou voulez-vous
reloger " DUP COUNT TYPE ." ? " #IN LATEST >R HERE >R SWAP
DP  ! (OVLOAD) R DP ! 0FF81 , LATEST , R> CURRENT @ ! R> ,
;
```

demande où charger le relogeable dont le nom est à ad ; crée à HERE une structure spéciale de 6 octets pour l'usage de UNLINK

+OVLOAD

```
: +OVLOAD ( -- )
BL WORD HERE (+OVLOAD) ;
```

+OVLOAD est à (+OVLOAD) ce que OVLOAD est à (OVLOAD)

UNLINK

```
: UNLINK ( -- )
07F01 SP@ CONTEXT @ @ (FIND) 0= 5 ?ERROR DROP NIP DUP  NFA
020 TOGGLE ( smudge ) 2- DUP @ SWAP 2- ! ;
```

"déloge" le dernier relogeable installé avec +OVLOAD ou (+OVLOAD)

EDIT

```
: EDIT ( scr -- )
LIT" A-EDITOR.COM" (+OVLOAD) ;
```

l'éditeur résident en ROM est un "éditeur de lignes" ; l'éditeur pleine page relogeable appelé par EDIT doit être sur le lecteur A, sous le nom EDITOR.COM ; le mot exécuté automatiquement au chargement de l'éditeur est également EDIT qui prend sur la pile le numéro d'écran à éditer ; ASSEMBLER, CODE et ;CODE relogent le module ASSEMBLER.COM, dont le dernier mot exécuté automatiquement prendra sur la pile 0 pour ouvrir le vocabulaire ASSEMBLER, 1 pour exécuter CODE, et 2 pour ;CODE

```
ASSEMBLER
      : ASSEMBLER 0 LIT" A-ASSEMBLER.COM" (+OVLOAD) ; IMMEDIATE

CODE
      : CODE 1 [COMPILE] ASSEMBLER ;

;CODE
      : ;CODE 2 [COMPILE] ASSEMBLER ;
```

Voici un petit exemple pratique de création et d'utilisation de relogeable :

```
OV6502  : BELL 7 EMIT ; : HELLO BELL ." SALUT " ; <RETURN>
ok.0
OV6502 : BELL 7 EMIT ; : HELLO BELL ." SALUT " ; <RETURN>
BELL (redefini) HELLO (redefini) ok.0
OVSAVE BELL SALUT <RETURN>  ok.0
FORGET BELL FORGET BELL <RETURN>  ok.0
OVLOAD SALUT <RETURN>  <tut !> SALUT  ok.0
```

Notre "application" se compose de 2 mots, BELL et HELLO, que nous compilons deux fois, précédée de l'exécution de OV6502 ; la deuxième fois, c'est normal, on vous prévient que tous les mots sont redéfinis ; il ne reste plus qu'à donner à OVSAVE le nom du premier mot de l'application et le nom du fichier à utiliser ; pour "voir si ça marche", on efface du dictionnaire les deux copies de l'application, et on charge le relogeable : le dernier mot est automatiquement exécuté.

Le seul moyen (fortement recommandé) d'être sûr de compiler deux fois la même chose, est d'éditer l'application sur disque (dans notre exemple, disons sur l'écran 1), puis d'éditer un écran spécial de génération de relogeable, dans notre exemple :

```
OV6502 1 LOAD OV6502 1 LOAD OVSAVE BELL SALUT ( génération
) FORGET BELL FORGET BELL OVLOAD SALUT ( test )
```

27. EDITEUR DE LIGNES

L'éditeur classique, "fig-FORTH portable editor, by William F. Ragsdale", est intégré à TELE-FORTH, avec quelques améliorations au niveau de la saisie des écrans.

Le numéro de l'écran courant et la position du curseur dans l'écran courant sont mémorisés dans les variables user SCR et R# ; la ligne courante est celle où se trouve le curseur ; la variable user SCR mémorise le dernier écran affiché par LIST.

```
FORTH DEFINITIONS HEX

TEXT
```

```
: TEXT ( c -- )
HERE C/L 1+ BLANKS WORD HERE PAD C/L 1+ CMOVE ;
```
fonctionne conune WORD, mais la chaîne se trouve à PAD

LINE

```
: LINE ( lig -- ad )
DUP 0FFF0 AND 017 ?ERROR SCR @ (LINE) DROP ;
```
retourne l'adresse de la ligne lig de l'écran courant ; génère une erreur "lignes de 0 à 15" si lig est hors de cet intervalle

EDITOR

```
VOCABULARY EDITOR IMMEDIATE EDITOR DEFINITIONS
```
le vocabulaire contenant tous les mots de l'éditeur de lignes

WHERE

```
: WHERE ( in blk -- )
DUP B/SCR / DUP SCR ! ." Ecran # " DECIMAL . SWAP C/L
/MOD C/L * ROT BLOCK + CR C/L TYPE CR HERE C@ - SPACES
ASCII ^ EMIT [COMPILE] EDITOR QUIT ;
```
en cas d'erreur pendant un chargement de fichier, WHERE est appelé pour afficher la ligne où s'est produite l'erreur ; une flèche est positionnée en dessous du mot qui a causé l'erreur.

Outils de base

#LOCATE

```
: #LOCATE ( -- col lig )
R# @ C/L /MOD ;
```
retourne la position courante du curseur

#LEAD

```
: #LEAD ( -- ligad col )
#LOCATE LINE SWAP ;
```
retourne l'adresse de la ligne courante, avec la position du curseur dans la ligne

#LAG

```
: #LAG ( -- curad cnteol )
#LEAD DUP>R + C/L R> - ;
```
retourne l'adresse du caractère sous le curseur, et le nombre de caractères jusqu'à la fin de la ligne courante

-MOVE

```
: -MOVE ( ad lig -- )
LINE C/L CMOVE UPDATE ;
```
remplace la ligne lig par les 64 caractères situés à ad ; le bloc est marqué modifié

MATCH

```
: MATCH ( curad cnteol ad cnt -- 1 cnteos /-- 0 cnteol )
FORTH >R >R 2DUP R> R 2SWAP R> - 0 MAX OVER + SWAP DO 2DUP
I SWAP S= IF I + >R 2DROP 0 R> ROT - 0 0 LEAVE THEN LOOP
2DROP >R 0= R> ;
```
recherche à partir de l'adresse curad pour cnteol caractères une chaîne identique à celle de cnt caractères située à ad ; si trouvée, le déplacement cnteos, de curad à la fin de la chaîne, est retourné avec un drapeau vrai,

sinon cnteol est retourné inchangé avec un drapeau faux

1LINE

```
: 1LINE ( -- b )
#LAG PAD COUNT MATCH R# +! ;
```

recherche la chaîne de caractères située à PAD de la position du curseur à la fin de la ligne courante ; si trouvée, le curseur est juste après et b est vrai, sinon le curseur est au début de la ligne suivante et b est faux

FIND

```
: FIND ( -- )
BEGIN 03FF R# @ < IF TOP PAD HERE C/L 1+ CMOVE 0 ERROR
THEN 1LINE UNTIL ;
```

recherche la chaîne de caractères située à PAD de la position du curseur à la fin de l'écran courant ; si trouvée, le curseur est juste après, sinon une erreur "pas trouve" est générée

DELETE

```
: DELETE ( n -- )
>R #LAG + FORTH R - ( ad pour BLANKS ) #LAG R MINUS R# +!
#LEAD + SWAP CMOVE ( recul curseur et texte ) R> BLANKS
UPDATE ; ( efface fin ligne )
```

efface n caractères à reculons à partir du curseur ; des espaces sont ajoutés en fin de ligne

nu

```
: nu ( lig cfa -- )
>R 010 0 DO CR FORTH I 3 .R SPACE I OVER - IF I SCR @
.LINE ELSE QUERY 1 TEXT TIB @ C@ IF I J EXECUTE 1+ ELSE
07F EMIT I SCR @ .LINE THEN THEN LOOP DROP R>DROP ;
```

affiche l'écran jusqu'à la ligne lig-1, puis saisit du texte et applique cfa (R pour remplacer ou I pour insérer une ligne), jusqu'à la fin de l'écran ou jusqu'à ce qu'une saisie soit vide (seulement un retour chariot) auquel cas les lignes restantes sont listées

Positionnement du curseur et affichage de la ligne courante

TOP

```
: TOP ( -- )
0 R# ! ;
```

positionne le curseur en haut et à gauche de l'écran

T

```
: T ( lig -- )
DUP C/L * R# ! H 0 M ;
```

"Type" : affiche la ligne lig ; le curseur est positionné au début de la ligne ; la ligne est copiée à PAD

M

```
: M ( n -- )
R# +! CR SPACE #LEAD TYPE 07E EMIT #LAG TYPE #LOCATE .
DROP ;
```

"Move" : déplace le curseur de n caractères ; le contenu et le numéro de la ligne courante sont affichés, avec un repère à la position du curseur

Echanges de ligne avec le PAD

H

```
: H ( lig -- )
LINE PAD 1+ C/L DUP PAD C! CMOVE ;
```

"Hold" : copie la ligne lig dans le PAD

R

```
: R ( lig -- )
PAD 1+ SWAP -MOVE ;
```

"Replace" : remplace la ligne lig par la ligne dans le PAD

Destructions de lignes

E

```
: E ( lig -- )
LINE C/L BLANKS UPDATE ;
```

"Erase" : efface la ligne lig avec des espaces

D

```
: D ( lig -- )
DUP H 0F DUP ROT DO I 1+ LINE I -MOVE LOOP E ;
```

"Delete" : détruit la ligne lig, décalant les suivantes vers le haut ; la ligne 15 est vierge

Insertions de lignes (la ligne 15 est perdue)

S

```
: S ( lig -- )
DUP 1- 0E DO I LINE I 1+ -MOVE -1 +LOOP E ;
```

"Split" : insère une ligne vide à lig, décalant les suivantes vers le bas

I

```
: I ( lig -- )
DUP S R ;
```

"Insert" : insère la ligne contenue dans le PAD à lig, décalant les suivantes vers le bas

Recherche et effacement de texte

F

```
: F ( -- )
1 TEXT N ;
```

"Find" : le texte qui suit F jusqu'au retour chariot est copié à PAD et recherché de la position courante du curseur à la fin de l'écran courant ; si trouvée, le curseur est positionné juste après et la ligne correspondante est affichée, sinon une erreur "pas trouve" est générée

N

```
: N ( -- )
FIND 0 M ;
```

"Next" : cherche l'occurence suivante d'une chaîne déjà trouvée une fois par F

B

```
: B ( -- )
PAD C@ MINUS M ;
```

"Back" : recule le curseur de la longueur de la chaîne dans le PAD, pour ramener le curseur en début de chaîne trouvée après un F ou un N

X

```
: X ( -- )
1 TEXT FIND PAD C@ DELETE 0 M ;
```

efface la première occurence du texte qui suit X jusqu'au retour chariot

TILL

```
: TILL ( -- )
#LEAD + 1 TEXT 1LINE 0= 0 ?ERROR #LEAD + SWAP - DELETE 0 M
;
```

efface de la position courante du curseur jusqu'à la fin de la première occurence sur la ligne courante du texte qui suit TILL jusqu'au retour chariot

Remplacement et ajouts de texte

C

```
: C ( -- )
1  TEXT  PAD COUNT #LAG ROT OVER MIN DUP>R R# +! FORTH R -
>R DUP HERE R CMOVE HERE #LEAD + R> CMOVE R> CMOVE  UPDATE
0 M ;
```

"Copy" : insère à la position du curseur le texte qui suit C jusqu'au retour chariot

P

```
: P ( lig -- )
1 TEXT R ;
```

"Put" : le texte qui suit P jusqu'au retour chariot remplace la ligne lig

NEW

```
: NEW ( lig -- )
[ EDITOR ' R CFA ] LITERAL nu ;
```

saisit et remplace à partir de la ligne lig ; voir l'outil nu

UNDER

```
: UNDER ( lig -- )
1+ [ EDITOR ' I CFA ] LITERAL nu ;
```

saisit et insère à partir de la ligne lig+1 ; voir l'outil nu

Manipulations d'écrans entiers

L

```
: L ( -- )
SCR @ LIST 0 M ;
```

affiche l'écran et la ligne courants ; utiliser LIST pour changer d'écran

CLEAR

```
: CLEAR ( scr -- )
```

SCR ! 010 0 DO FORTH I EDITOR E LOOP ;
effacé l'écran scr

SCRMOVE

: SCRMOVE (srce dest cnt --)
FORTH ABS >R 2DUP < IF R 1- DUP D+ 0 R> MINUS ELSE R> 0 THEN DO 2DUP COPY 1 1 I 0< IF DMINUS THEN D+ LOOP 2DROP ;
remplace les cnt écrans à partir de dest par les cnt écrans à partir de srce ; l'ordre de copie est déterminé automatiquement

SCRCOPY

: SCRCOPY (srce dest cnt --)
USING ." de " 3 PICK DUP . ." a " OVER + . XFILE ." vers" USING ." de " OVER DUP . ." a " OVER + . XFILE ." Copier" OK? IF 0 DO OVER I + BLOCK XFILE OVER I + SWAP 2- ! XFILE LOOP ELSE DROP THEN 2DROP ;
remplace les cnt écrans à partir de dest dans le fichier destinataire par les cnt écrans à partir de srce du fichier origine ; la procédure de copie peut être abandonnée en tapant <ESC> (voir OK?)

28. EDITEUR PLEINE PAGE

Il sera disponible séparément sur disquette, avec le macro-assembleur et l'extension de calcul en virgule flottante (à ma connaissance, cet éditeur n'a jamais été commercialisé. L'éditeur pleine écran du FORTH 83-STANDARD devrait pouvoir être porté sans trop de difficulté si nécessaire).
C'est un relogeable autochargeable (voir EDIT au chapitre "Gestion de Modules Relogeables"), qu'on peut ensuite déloger pour récupérer de l'espace mémoire.
C'est un éditeur pleine page ("what you see is what you get") adapté spécialement à l'écran du TELESTRAT.

29. MACRO-ASSEMBLEUR 6502

Il sera disponible séparément sur disquette, avec l'éditeur pleine page et l'extension de calcul en virgule flottante (à ma connaissance, ce macro-assembleur n'a jamais été commercialisé. Le macro-assembleur du FORTH 83-STANDARD devrait pouvoir être porté sans trop de difficulté si nécessaire).
C'est un relogeable qu'on peut soit intégrer au dictionnaire (pour générer un autre relogeable avec plus de macros par exemple), soit implanter ailleurs en mémoire pour pouvoir après son emploi récupérer de l'espace mémoire (voir OVLOAD et +OVLOAD au chapitre "Gestion de Modules Relogeables").
C'est un macro-assembleur dans l'optique FORTH, les arguments précédant les mnémoniques ; nombre de macros sont prédéfinies : adressage des deux piles, sauts aux principales adresses de la machine virtuelle FORTH, et surtout structuration des sauts intra-définition par IF, ELSE, THEN, BEGIN, AGAIN, UNTIL, WHILE, REPEAT, et création dynamique de "labels" pour sauts inter-définitions.
Un utilitaire de mesure de fréquence d'utilisation des

mots d'une application l'accompagne pour permettre un choix à bon escient des routines les plus employées à optimiser en assembleur (par contre, je ne connais pas d'équivalent à cet utilitaire de mesure de fréquence en FORTH 83-STANDARD). TELE-FORTH n'est pas lent (vous pourrez lui faire faire des concours de vitesse avec le BASIC, qui est pourtant d'une facture particulièrement performante ; ça vous donnera une idée...), mais l'écriture en assembleur des mots les plus utilisés dans une application peut apporter une amélioration de performances appréciable.

30. EXTENSION DE CALCUL EN VIRGULE FLOTTANTE

Elle sera disponible séparément sur disquette, avec l'éditeur pleine page et le macro-assembleur (à ma connaissance, cette extension de calcul en virgule flottante n'a jamais été commercialisée. L'extension de calcul en virgule flottante du FORTH 83-STANDARD devrait pouvoir être portée sans trop de difficulté si nécessaire).
C'est un relogeable utilisant les mêmes primitives de calcul que le BASIC, celles qui sont intégrées au moniteur du TELESTRAT.
Chaque "flottant" prend 5 octets en mémoire, 7 dans la pile, d'où l'utilisation d'une pile séparée pour les calculs en flottant. L'extension peut être activée ou désactivée, pour permettre la reconnaissance automatique des nombres flottants dans le flot d'entrée

31. LE JEU DE LA VIE

Ce programme, que vous avez peut-être vu en démonstration, s'adresse à tous les publics : pour les débutants, c'est un simple exemple de programmation mettant en oeuvre la plupart des structures de contrôle FORTH, pour les plus chevronnés, c'est une occasion d'aborder un problème d'efficacité algorithmique ; pour tous, c'est une invitation à fouiller les possibilités offertes par le TELESTRAT.

Règles :

Sur un espace quadrillé (un écran par exemple), apparaissent et disparaissent des pions. La survie d'un pion dépend du nombre de ses voisins dans les huit cases qui entourent celle qu'il occupe : moins de deux voisins, le pion meurt d'isolement, plus de trois voisins, il meurt de surpopulation. S'il y a des morts, il faut des naissances : dans chaque case vide qui a exactement trois voisins, un heureux évènement se produit, la naissance d'un pion.

Structures de données :

Essayez sur un échiquier, vous constaterez vite que pour s'y retrouver, il ne faut surtout toucher aucun pion tant que le compte des voisins n'a pas été fait pour toutes les cases ; vous constaterez aussi vite que les seules cases dignes d'intérêt sont celles occupées et leurs voisines.
Dans le cas d'un échiquier, il n'y a que 64 cases, aussi

peut-on se permettre d'étudier systématiquement toutes les cases : il faudra en tester 8*64=512 par génération. Dans le cas de l'écran texte du TELESTRAT, il faudra en tester 8*40*27=8640, et dans le cas de l'écran haute résolution 8*240*200=384000 : même à raison d'un pixel testé par milliseconde, il faudrait 6 minutes par génération !
D'où l'intérêt de choisir des structures de données adaptées :
- une première liste des coordonnées des pions vivants
- une autre liste de leurs cases voisines non occupées
- une première bitmap (un bit par case) pour l'ancienne génération en cours de comptage
- une seconde bitmap pour la nouvelle génération en cours de création
- une troisième bitmap de travail pour marquer les cases voisines déjà prises en compte

Les bitmaps servent à accéder directement aux cases voisines sans avoir à parcourir la liste des pions vivants ; dans le cas de l'écran haute résolution, chaque bitmap occupera 240*200/8=6000 octets ; on pourrait se contenter d'une seule bitmap, à chaque fois reconstruite à partir de la liste des pions vivants, ce qui prendrait forcément du temps, alors qu'on dispose de place mémoire ; par contre on peut générer la liste des cases voisines à la suite immédiate de la liste des pions vivants, car le nombre des anciens pions vivants ne peut que diminuer, et le nombre des nouveaux ne peut être supérieur au nombre des cases voisines.

Implantation mémoire :

L'application étant résidente en ROM, il faut décider au moment de son initialisation de l'implantation de ses structures de données ; la méthode classique consiste à utiliser la mémoire inoccupée au dessus du dictionnaire FORTH ; c'est d'ailleurs le cas du PAD qui utilise les 68 premiers octets pour les conversions numérique vers ASCII ; les trois bitmaps étant de taille fixe, elles peuvent être implantées au-dessus du PAD, suivies de la liste des coordonnées qui pourra s'étendre et se rétracter au fur et à mesure des besoins ; nous travaillerons donc avec 6 pointeurs, un pour le début de chaque bitmap (NEWm OLDm WRKm), un pour le début de la liste des coordonnées des pions vivants (org^), un pour la fin de cette même liste (mid^), c'est-à-dire pour le début de la liste des coordonnées des cases voisines, le dernier pour la fin de cette dernière liste (end^).

Fonctions d'accès aux structures de données :

Deux fonctions permettent de tester ou de changer l'état d'un bit d'une bitmap (m@ m!), une fonction permet d'ajouter les coordonnées d'une case dans une liste (^,) ; certains prétendent que FORTH est un langage illisible : ici nous avons associé les noms de fonctions avec les noms de pointeurs, par exemple pour tester un bit donné, on écrira xy NEWm m@, pour changer son état xy NEWm m!, et pour ajouter une case à une liste xy mid^ ^, ou xy end^ ^, : ce sont peu d'efforts pour une bonne lisibilité...

Algorithme :

Partant d'une liste de pions vivants (entre org^ et mid^) et d'une bitmap (NEWm) cohérentes entre elles, on va

commenccr par mettre les deux autres bitmaps (OLDm et WRKm) dans le même état, par vider la liste des cases voisines (mid^ @ end^ !), et préparer la réception des survivants (org^ @ mid^ !) ;
puis pour chaque pion vivant, on va compter les cases occupées autour (xy OLDm m@), et à chaque fois qu'on passe sur une case vide, si elle n'a pas déjà été prise en compte (xy WRKm m@), on l'ajoute à la liste des cases voisines (xy end^ ^,) et on la marque (xy WRKm m!) ; si le nombre de voisins est 2 ou 3, le pion survit, donc on l'ajoute à la liste des survivants (xy mid^ ^,) sinon on l'efface de la bitmap (xy NEWm m!) et de l'écran (xy 0 PAINT) ; jusqu'à épuisement de la liste des pions vivants ;
puis pour chaque case de la liste des cases voisines (entre mid^ et end^), on va compter les pions autour (xy OLDm m@) ; si le nombre de voisins est exactement 3, un pion nait, donc on l'ajoute à la liste des pions vivants (xy mid^ ^,) et à la bitmap (xy NEWm m!) et on l'affiche à l'écran (xy 1 PAINT) ; jusqu'à épuisement de la liste des cases voisines ; on peut alors passer à la génération suivante.

Représentation interne des coordonnées :
Les coordonnées sont toujours difficiles à manipuler par deux, c'est pourquoi généralement on les compacte en un seul nombre, somme de l'abscisse et du produit de l'ordonnée par la valeur maximum que peut prendre l'abscisse (le nombre de colonnes par ligne) ; ainsi les coordonnées des 8 cases voisines peuvent-elles être obtenues par la simple addition d'un déplacement précalculé (voir DXY et around) ; pour retrouver les coordonnées originales, il suffira de faire une division par le nombre de colonnes par ligne, donnant l'ordonnée pour quotient et l'abscisse pour reste.

Fonctions dépendantes de la machine :
Jusqu'à présent, nos réflexions étaient valables pour n'importe quel ordinateur ; il faut maintenant pouvoir prendre en compte le nombre réel de colonnes/ligne et de lignes/colonne, et pouvoir effacer et dessiner un pion à l'écran ;
par ailleurs pour pouvoir positionner les pions de la première génération, après avoir effacé l'écran, il est agréable de pouvoir utiliser le curseur à l'écran comme repère, et la barre d'espacement pour changer l'état d'une case ; on pourrait aussi aimer savoir après chaque calcul de quelle génération il s'agit et combien elle comprend de pions vivants ;
enfin, quand on a longtemps patienté pour obtenir de belles configurations après quelques centaines de générations, on appréciera de pouvoir sauvegarder sur disque une génération, et donc de pouvoir la recharger.

```
FORTH DEFINITIONS DECIMAL
VOCABULARY LIFE LIFE DEFINITIONS
```

R#

```
0 VARIABLE R# \ coordonnées curseur
```

DXY

```
      0 VARIABLE DXY \ pointeur table coord. relatives voisins

NEWm
      0 VARIABLE NEWm \ pointeur bitmap nouvelle génération

OLDm
      0 VARIABLE OLDm \ pointeur bitmap ancienne génération

WRKm
      0 VARIABLE WRKm \ pointeur bitmap marquage voisins

org^
      0 VARIA8LE org^ \ pointeur début liste

mid^
      0 VARIABLE mid^ \ pointeur milieu liste

end^
      0 VARIABLE end^ \ pointeur fin liste

      \ mots dépendants de la machine

C:L
      : C:L \ -- n ; colonnes/ligne
      ?HIRES IF 240 ELSE 40 THEN ;

L:C
      : L:C \ -- n ; lignes/colonne
      ?HIRES IF 200 ELSE 27 THEN ;

C*L
      : C*L \ -- u ; lignes*colonnes, pour efficacité, voir m#
      ?HIRES IF 48000 ELSE 1080 THEN ;

m#
      : m# \ xy -- xy ; modulo C*L pour quadrillages miroirs
      ?HIRES IF -241 OVER U< \ 48000=-17536 sur 16 bits !
      ELSE DUP 0< \ en texte il suffit de tester le signe
      THEN IF C*L + ELSE C*L 1- OVER U< IF C*L - THEN THEN ;

TOKEN
      ASCII o CONSTANT TOKEN \ en mode texte, pour efficacité,

TXTORG
      HEX  BBA8  CONSTANT  TXTORG DECIMAL \ accès direct mémoire
      écran

PAINT
      : PAINT \ xy b -- ; b faux = effacer
      ?HIRES IF GRAFX FB 0 C:L U/ CURSET ELSE IF TOKEN  ELSE  BL
      THEN SWAP TXTORG + C!  THEN ;

.CURSOR
      : .CURSOR \ -- ; afficher le curseur clignotant en R#
      R#  @ m# DUP R# ! DUP NEWm m@ BEGIN 2DUP 0= PAINT 1 2 WAIT
      2DUP PAINT 2 2 WAIT ?TERMINAL UNTIL 2DROP ;

.GENE
      : .GENE \ n -- ; afficher génération, nombre de vivants et
      d'étudiés
      0  0 GOTOXY ." Gene " . ." : " mid^ @ org^ @ - 2/ . ." / "
```

```
      end^ @ org^ @ - 2/ . ;

QFN
      : QFN \ b -- ; QueryFileName pour Save et Load
      0 0 GOTOXY IF ." Load:" ELSE ." Save:" THEN BLK @ >R IN  @
      >R QUERY BL WORD HERE FILENAME R> IN ! R> BLK ! ;

      \ fonctions d'accès aux structures de données

mLEN
      : mLEN \ -- n ; nb d'octets d'une bitmap
      C:L L:C 8 */ ;

m,
      : m, \ map -- ; implante et initialise une bitmap
      HERE DUP ROT ! mLEN DUP ALLOT ERASE ;

m@
      : m@ \ xy map -- b ; teste un bit d'une bitmap
      @ SWAP BMAP CTST ;

m!
      : m! \ xy map -- ; change un bit d'une bitmap
      @ SWAP BMAP TOGGLE ;

^,
      : ^, \ xy list^ -- ; ajoute une case à la liste
      DUP>R @ ! 2 R> +! ;

around
      : around \ -- 0 DXY+16 DXY ; pour les boucles voisins
      0 DXY @ DUP 16 + SWAP ;

INI
      : INI \ -- ; initialise tout
      HERE  PAD DP ! NEWm m, OLDm m, WRKm m, HERE DXY ! -1 , 1 ,
      C:L DUP DUP MINUS , , DUP 1- DUP MINUS , , 1+ DUP MINUS  ,
      , HERE DUP org^ ! mid^ ! C:L L:C 2 */ R# !  DP ! ?HIRES IF
      GRAFX HIRES ELSE CLS THEN ;

GENE
      GENE \ -- ; calcul d'une génération
      NEWm @ mLEN 2DUP \ copier NEWm
      OLDm @ SWAP CMOVE \ sur OLDm et
      WRKm @ SWAP CMOVE \ sur WRKm
      mid^ @ DUP \  conserver  une  copie  de  mid^  pour  liste
      voisins
      DUP end^ ! \ mettre end^ à mid^
      org^ @ \ pour I de org^ à mid^ = pour chaque vivant :
      DUP mid^ ! \ avec mid^ à org^
      DO  I  @  around  DO  OVER  I  @ + m# \ pour chacun de ses
      voisins :
      DUP>R OLDm m@ \ si vivant :
      IF 1+ \ le compter,
      ELSE R WRKm m@ 0= \ sinon, si nouveau voisin :
      IF R WRKm m! \ le marquer et
      R end^ ^, \ l'ajouter à la liste voisins,
      THEN \ sinon déjà vu.
      THEN R>DROP 2 +LOOP \ voisin suivant ;
      14 AND 2 = \ si 2 ou 3 voisins :
      IF mid^ ^, \ ajouté à la liste des pions survivants,
```

```
        ELSE DUP 0 PAINT \ sinon l'effacer de l'écran et
        NEWm m! \ de la bitmap.
        THEN 2 +LOOP \ vivant suivant ;
        end^ @ SWAP \ pour I de mid^ à end^ = pour chaque à voir
        DO I @ around DO OVER I @  +  m#  \  pour  chacun  de  ses
        voisins :
        OLDm m@ + 2 +LOOP \ le compter, voisin suivant ;
        3 = IF DUP mid^ ^, \ si 3 voisins : ajouté à liste vivants
        et
        DUP 1 PAINT \ affiché à l'écran et
        DUP NEWm m! \ ajouté à la bitmap.
        THEN DROP 2 +LOOP ; \ à voir suivant ; fin de  génération.

        FORTH DEFINITIONS DECIMAL

DEMO
        : DEMO \ -- ; dans le vocabulaire FORTH
        ASCII  I  BEGIN  CASE  ASCII I OF INI ENDOF 08 OF -1 R# +!
        ENDOF \ gauche
        09 OF 1 R# +! ENDOF \ droite
        10 OF C:L R# +! ENDOF \ bas
        11 OF C:L MINUS R# +! ENDOF \ haut
        BL OF R# @ \ inverser état pion
        DUP NEWm m! \ dans bitmap
        DUP NEWm m@ \ nouvel état
        2DUP PAINT \ l'afficher
        IF mid^ ^, \ si né : l'ajouter à liste,
        ELSE mid^ @ org^ @ \ sinon le chercher et
        DO DUP I @ = \ quand trouvé :
        IF mid^ @ @ I ! \ remplacer par dernier
        -2 mid^ +! \ et effacer dernier
        LEAVE \ fin recherche.
        THEN 2 +LOOP DROP \ suite recherche ;
        THEN ENDOF 13 OF 0 BEGIN org^ @ mid^ @ =  ?TERMSTOP  OR  \
        arrêt ?
        0=  WHILE GENE 1+ DUP .GENE REPEAT DROP ENDOF ASCII S OF 0
        QFN \ Save population
        org^ @ mid^ @ 1- $SAVE ENDOF ASCII L OF R# @ C:L L:C 2  */
        - \ Load population
        1 QFN mid^ @ DUP $LOAD 1+ SWAP \ ne créer
        DO DUP I @ + m# DUP NEWm m@ \ que les nouveaux
        IF  DROP ELSE DUP mid^ ^, DUP NEWm m! 1 PAINT THEN 2 +LOOP
        DROP ENDOF 03 OF ;S ENDOF 7 EMIT ENDCASE \ Sinon tut !
        .CURSOR KEY AGAIN ; \ afficher curseur

        \ That's all folks !
```

32. BIBLIOGRAPHIE

"Débuter en Forth" de Leo Brodie est sans conteste l'ouvrage le plus pédagogique pour les débutants ; si vos finances vous permettent d'acheter deux ouvrages, prenez également "Tours de Forth", pour ses nombreux exemples. Et si vous voulez disposer d'une boîte à outils très complète, "Dr Dobb's toolbook of Forth", si l'anglais ne vous répugne pas, évidemment.

Le Forth, McCabe & Harvey (Belin, collection Modulo 1985)

Forth : manuel de référence TO7-MO5, (Cedic-Nathan 1985)

Débuter en Forth, Leo Brodie (Eyrolles 1984)

Tours de Forth, Michel Rousseau et Marc Petremann (Eyrolles 1984)

Forth, Sallman, Tisserant & Toulout (Eyrolles 1984)

Forth : Manuel d'application, M.S. Ewing (Masson 1984)

Le concept Forth : langage et système, Pascal Courtois (Editests 1983)

Programmer en Forth, Alain Pinaud (PSI 1983)

Dr. Dobb's Toolbook of Forth, (Marlin Ouverson 1986)

Thinking Forth, Leo Brodie (Prentice Hall 1984)

Invitation to Forth, Harry KATZAN J.R (Petrocelli Books)

Exploring Forth, Owen Bishop (Granada 1984)

Forth for Micros, Steve Okay (Newness Programming Books 1984)

Forth fundamentals, Kevin et MacCabe (Dilithium Press 1983)

The Complete Forth, Alan Winfield (SIGMA Technical Press 1983)

Discover Forth, Tom Hogan (Mac GrawHill 1982

FORTH Handbuch, Floegel (Hofacker 1982)

33. MESSAGES D'ERREUR

0 pas trouve
ce message n'est pratiquement utilisé que par l'éditeur lors de la recherche d'une chaîne de caractères

1 pile vide
lorsqu'il n'y a pas assez de valeurs sur la pile pour une fonction, le pointeur de pile déborde en dehors de la zone autorisée ; l'interpréteur teste le pointeur de pile après l'interprétation de chaque mot du flot d'entrée

2 dictionnaire plein
la taille mémoire restante est testée à la création de chaque entrée dans le dictionnaire ; s'il reste moins de 80 octets, cette erreur est générée

3 mode d'adressage incorrect
ce message n'est pratiquement utilisé que par l'assembleur

4 (redefini)
avant de créer une nouvelle entrée dans le dictionnaire, FORTH cherche s'il n'en existe pas déjà une du même nom, et vous prévient qu'elle ne sera plus accessible (puisque la nouvelle en cours de définition sera trouvée la première) ; ce n'est pas une condition d'erreur, juste un message d'avertissement (dans le cas des relogeables, la deuxième compilation produit ce message pour tous les mots définis)

5 inconnu dans ce vocabulaire
FORTH génère cette erreur quand il ne sait plus quoi faire d'un mot que vous lui avez donné à interpréter : l'interprète l'a d'abord cherché dans le vocabulaire CONTEXT, puis dans le vocabulaire CURRENT, puis a essayé de le convertir en nombre en fonction de la base courante, mais a échoué partout ; à moins d'une erreur de frappe toujours possible, il se peut que le mot se trouve dans un autre vocabulaire dans lequel FORTH n'a pas cherché.

7 pile pleine
la pile de TELE-FORTH est limitée à 25 valeurs (50 octets) ; si vous en mettez plus vous génèrerez cette erreur, voir aussi message 1

10=0Ah OVSAVE: compiler deux fois
avant de sauvegarder un relogeable, il faut compiler deux fois le même code ; voir le chapitre sur les relogeables

11=0Bh OVSAVE: utiliser OV6502
à cause de certaines particularités du 6502, le microprocesseur du TELESTRAT, les relogeables doivent commencer sur une adresse de début de page mémoire ; OV6502 assure cette condition ; voir le chapitre sur les relogeables

13=0Dh la base doit être decimale
vous n'êtes pas en base décimale, et l'opération que vous tentez le réclame ; tapez DECIMAL et recommencez votre opération

15=0Fh TeleFORTH rom (c) ORIC International
c'est le texte qui apparait au bas de chaque page imprimée en utilisant TRIAD

17=11h en definition seulement
le mot qui a généré cette erreur ne peut être utilisé qu'à l'intérieur d'une définition

18=12h hors definition seulement
le mot qui a généré cette erreur ne peut pas être utilisé dans une définition

19=13h controles mal appaires
les structures de contrôle ne sont pas correctement structurées : on peut les imbriquer les unes dans les autres, mais elles doivent toujours rester balancées au même niveau

20=14h definition incomplete
la profondeur de la pile en fin de définition n'est pas la même qu'en début de définition ; la pile est utilisée pour construire les structures de contrôle en cours de définition : il est fort probable qu'une structure de contrôle n'ait pas été complétée

21=15h dictionnaire protege
vous avez tenté d'utiliser FORGET avec un mot qui est en zone dictionnaire protégée ; l'adresse limite est contenue dans la variable FENCE : ne jouez pas avec sans savoir ce que vous faites (après tout vous ne risquez rien, TELE-FORTH est en ROM) !

22=16h sur disque seulement
le mot qui a causé l'erreur ne peut être employé que dans des sources sur disque

23=17h lignes 0 a 15
les numéros de ligne des écrans (blocs de 1 Ko de source divisé en 16 lignes de 64 caractères) doivent être entre 0 et 15 ; vous avez dû essayer une fonction de l'éditeur sans donner de numéro de ligne (tapez EDITOR TOP et réessayez)

24=18h declarer un vocabulaire
vous avez tenté d'utiliser FORGET alors que les vocabulaires CONTEXT et CURRENT ne sont pas identiques (tapez DEFINITIONS et réessayez)

129=81h Fichier inexistant
vous avez tenté d'ouvrir un fichier qui n'est pas sur la disquette du lecteur courant ; utilisez USING pour ouvrir un fichier : s'il n'existe pas, USING vous demandera si vous voulez qu'il le crée

130=82h Erreur disque
votre disquette commence peut-être à être fatiguée : faites quelques autres essais avant de décider qu'elle est morte ; essayez d'en faire une copie, vous pourrez peut-être ainsi récupérer sinon la totalité, du moins une partie de votre travail ; les disquettes craignent la poussière, les cendres et la fumée dc cigarette... prenez-

en soin et ne les laissez pas traîner hors de leurs pochettes ou de leur boite

131=83h Fichier existant
vous essayez d'écraser un fichier existant avec une nouvelle version, sans avoir précisé que vous ne voulez pas protéger l'ancien ; l'instruction COPY en ROM ne génèrera pas cette erreur

132=84h Disquette pleine
vous n'avez plus assez de place sur votre disquette pour l'opération en cours

133=85h Disquette protegee
remontez la languette de protection si vous voulez écrire sur votre disquette

134=86h Erreur de type
les fichiers source FORTH sont des fichiers à accès direct, donc de type 8 et d'extension .DAT ; les relogeables sont des fichiers contenant une image mémoire, donc de type 0 et d'extension .COM ; l'extension du fichier que vous voulez ouvrir ne correspond pas avec son type ; avec USING, il faut toujours préciser l'extension .DAT

135=87h Disquette non STRATSED
votre disquette n'a pas été formatée sur un TELESTRAT

136=88h Pas de STRATSED
votre disquette n'a pas le Système d'Exploitation de Disque du TELESTRAT

137=89h Nom de fichier incorrect
le nom d'un fichier doit avoir au plus 9 caractères avant le point, au plus 3 caractères après le point, ne doit contenir que des caractères alphabétiques ou numériques

138=8Ah Lecteur absent
le TELESTRAT peut supporter jusqu'à 4 lecteurs de disquettes, repérés dans l'ordre par les lettres A à D ; celui que vous voulez utiliser n'est pas connecté

254=FEh Cross Compiler MicroProcessor Eng Ltd
c'est la référence du "méta-générateur' avec lequel TELE-FORTH a été généré

255=FFh Chritophe LAVARENNE tel:45519287 PARIS
ce sont les coordonnées de l'auteur de ces lignes, créateur de TELE-FORTH

34. INDEX

Pour chaque entrée de l'index, sont donnés le libellé du mot, et le vocabulaire dans lequel le mot se trouve, FORTH étant le vocabulaire par défaut.

Le classement de l'index est fait par ordre alphabétique ASCII ; pour vous faciliter la recherche des noms aux initiales non alphabétiques, voici les caractères de code ASCII 32 (l'espace) à 126 (le tilde) :

```
  !  " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; <
= > ?  @ A B C D E F G H I J K L M N O P Q R S T U V W X Y
Z  [ \ ] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v
w x y z { | } ~
```

-MOVE EDITOR
-TRAILING
.
."
.CURSOR LIFE
.FILE
.GENE LIFE
.LINE
.R
.S
/
/MOD
0
0<
0=
0>
0BRANCH
1
1+
1-
1LINE EDITOR
2
2!
2*
2**
2+
2-
2/
2@
2DROP
2DUP
2SWAP
:
;
;CODE
;S
<
<#
<BUILDS
=
>
>R
?
?COMP
?CSP
?ERROR
?EXEC
?FILE
?HIRES
?LEAVE
?LOADING
?OK
?PAIRS
?STACK
?TERM IOS
?TERMINAL
?TERMSTOP
@
ABORT
ABOX GRAFX
ABS

ACCENT IOS
ADCHAR GRAFX
ADRAW GRAFX
AGAIN
ALLOT
AND
APPLIC IOS
ASCII
ASSEMBLER
AYX
AZERTY IOS
B EDITOR
B/BUF
B/SCR
BASE
BEGIN
BIN
BL
BLANKS
BLK
BLOCK
BMAP
BOX GRAFX
BRANCH
C EDITOR
C!
C*L LIFE
C,
C/L
C:L LIFE
C@
CASE
CEMIT IOS
CFA
CHAR: GRAFX
CIRCLE GRAFX
CKEY IOS
CLCH IOS
CLEAR EDITOR
CLOCK
CLS
CMOVE
CMOVE>
CODE
COLD
COMPILE
CONSOLE IOS
CONSTANT
CONTEXT
COPY
COUNT
CR
CREATE
CRST
CSET
CSP
CTST
CURMOV GRAFX
CURRENT
CURSET GRAFX
CURSOR WINDOWS

D EDITOR
D+
D+-
D.
D.R
DABS
DECIMAL
DECON IOS
DEFER
DEFINITIONS
DEL
DELETE EDITOR
DEMO
DEPTH
DIGIT
DIR
DLITERAL
DMINUS
DO
DOES>
DOS
DP
DPL
DRAW GRAFX
DRIVE
DROP
DUP
DUP>R
DXY LIFE
E EDITOR
EDIT
EDITOR
ELSE
EMIT
EMPTY-BUFFERS
ENCLOSE
END
ENDCASE
ENDIF
ENDOF
ERASE
ERROR
EXECUTE
EXPECT
EXPLODE SOUNDS
F EDITOR
FB GRAFX
FENCE
FILENAME
FILL
FIND EDITOR
FIRST
FLUSH
FORGET
FORTH
FRENCH IOS
GENE LIFE
GOTOXY
GRAFX
H EDITOR
H.

HCOPY IOS
HEMIT GRAFX
HERE
HEX
HIMEM
HIRES GRAFX
HLD
HOLD
HRS: GRAFX
HTYPE GRAFX
I
I EDITOR
ID.
IF
IF(
IMMEDIATE
IN
INDEX
INI LIFE
INIT
INK GRAFX
INPUT IOS
INTERPRET
IOS
IS
INUMBER
J
KBD: IOS
KEY
L EDITOR
L:C LIFE
LATEST
LEAVE
LFA
LIFE
LIGNE IOS
LIMIT
LINE EDITOR
LIST
LIT
LIT"
LITERAL
LOAD
LOAD-USING
LOOP
M EDITOR
M*
M/
M/MOD
MATCH EDITOR
MAX
MEMIT IOS
MESSAGE
MIN
MINITEL IOS
MINUS
MLOAD IOS
MLOADA IOS
MNTL IOS
MOD
MON

```
MON:
MOVE
MRK<
MRK>
MSAVE IOS
MSGW WINDOWS
MUSIC SOUNDS
N EDITOR
NEW EDITOR
NEWm LIFE
NFA
NIP
NOOP
NOT
NUMBER
OF
OF"
OFF
OK
OLDm LIFE
ON
OPCH IOS
OR
OUPS SOUNDS
OUT
OUTPUT IOS
OV6502
OVER
OVLOAD
OVSAVE
P EDITOR
PAD
PAGE IOS
PAINT LIFE
PAPER GRAFX
PATTERN GRAFX
PAVE GRAFX
PEMIT IOS
PFA
PICK
PING SOUNDS
PIO
PLAY SOUNDS
PROMPT
PSG! SOUNDS
PSG: SOUNDS
QFN LIFE
QUERY
QUIT
QWERTY IOS
R
R EDITOR
R#
R/W
R0
R>
R>DROP
RECURSE
REPEAT
RES<
RES>
```

RING IOS
ROLL
ROT
RP@
RP!
S EDITOR
S->D
S0
S=
SALO!
SCR
SCRCOPY EDITOR
SCRMOVE EDITOR
SCROLL WINDOWS
SCRW WINDOWS
SDUMP IOS
SEMIT IOS
SERVEUR IOS
SHOOT SOUNDS
SIGN
SLOAD IOS
SLOADA IOS
SMUDGE
SOUND SOUNDS
SOUNDS
SP!
SP@
SPACE
SPACES
SSAVE IOS
ST-R/W
STARTUP
STATE
SWAP
T EDITOR
TERMINAL
TEXT EDITOR
TEXT GRAFX
THEN
THRU
THRU-USING
TIB
TILL EDITOR
TINPUT IOS
TOGGLE
TOKEN LIFE
TOP EDITOR
TRAVERSE
TRIAD
TSTLP IOS
TXTORG LIFE
TYPE
U*
U.
U/
U<
UNDER EDITOR
UNLINK
UNTIL
UPDATE
USER

USING
V<
VARIABLE
VCOPY IOS
VLIST
VOC-LINK
VOC-LIST
VOCABULARY
WAIT
WARM
WARNING
WCOPY IOS
WCOUNT
WCXFI IOS
WHERE EDITOR
WHILE
WIDTH
WINDOW: WINDOWS
WINDOWS
WORD
WRKm LIFE
X EDITOR
XFILE
XOR
ZAP SOUNDS
[
[COMPILE]
\
\C
\D
\J
\L
\S
]
^, LIFE
around LIFE
di
ei
end^ LIFE
m! LIFE
m# LIFE
m, LIFE
m@ LIFE
mLEN LIFE
mid^ LIFE
nu EDITOR
org^ LIFE

1

6