ORIC FRANCE

FORTH POUR ORIC

SORACOM éditions

FORTH
POUR ORIC

ORIC FRANCE

FORTH POUR ORIC

DIFFUSION

ASN Z.I. «La Haie Griselle»
94470 Boissy-St-Léger

SOMMAIRE

INTRODUCTION

Le langage FORTH a été créé par M. Charles H. Moore en 1969 à l'Observatoire National de Radio Astronomie de Charlottesville, dans l'état de Virginie, aux U.S.A.
Il a été créé pour compenser les lacunes des outils de programmation disponibles, en particulier pour l'automatisation des observatoires.

M. Moore et quelques associés ont formé la société Forth Inc. en 1973, qui a pour but de soutenir le système d'exploitation du FORTH et son langage de programmation, et de founir des programmes d'application afin de répondre aux demandes des utilisateurs.
Cette version de FORTH est celle issue du Forth Interest Group (F.I.G.) qui se trouve dans le nord de la Californie. Ce groupe a été formé en 1978 par les programmeurs de FORTH pour encourager l'utilisation de ce langage, et échanger des idées au travers de séminaires et de publications.
F.I.G. a publié un modèle du langage et le code d'implémentation de ce langage sur un grand nombre de microprocesseurs. Cette publication est dans le domaine public, cependant il est nécessaire de la modifier pour qu'elle puisse tourner sur un système cible précis. Le programme ainsi remanié appartient à la personne qui l'a adapté, qui peut alors prendre des droits d'auteur sur sa version sur sa version particulière.

Ce manuel ne prétend pas être un livre exhaustif sur ce langage, mais plutôt une introduction à son usage, et une description générale de son travail interne. Les utilisateurs qui désirent de plus amples informations doivent écrire au Forth Interest Group dont l'adresse est : P.O. Box 1105, San Carlos, California 94070, U.S.A.

Le groupe F.I.G. vend un certain nombre de livres (certains étant disponibles par l'intermédiaire de libraires) et il sera heureux de vous envoyer un formulaire d'adhésion au F.I.G., ainsi que la liste des publications disponibles.
Un bon livre disponible dès maintenant se nomme « Starting Forth », l'auteur étant Léo Brodie, publié par Prentice Hall ; bien qu'il soit fondé sur le Poly-Forth (Ξ) qui comporte quelques différences avec le F.I.G. Forth.

(Ξ) Poly-Forth est une marque déposée par FORTH Inc.

Chapitre 1.

LA CASSETTE SOURCE

Compliments pour l'achat de ce langage FORTH pour votre ordinateur ORIC-1. Nous espérons que vous trouverez à la fois un grand plaisir et une meilleure connaissance des techniques de programmation en utilisant ce langage inhabituel.
La première chose à faire est de charger le programme FORTH dans votre ORIC. Remarquez que cela est simplement réalisé en entrant :

CLOAD « FORTH »,S

Le fichier sur la cassette distribuée étant enregistré en vitesse lente afin d'obtenir une relecture plus fiable. Si vous le désirez vous pouvez faire votre propre copie de FORTH en vitesse rapide, la méthode à utiliser est expliquée plus loin. Le programme FORTH prend environ dix minutes pour être lu.
Une fois que le programme a été entièrement chargé, vous pouvez lancer FORTH en tapant : CALL ≠ 400 (Return). Si vous tombez accidentellement dans le BASIC (par exemple en pressant le bouton de Reset), vous pouvez relancer FORTH sans perdre tout votre travail en tapant : CALL ≠ 404 (Return).
Lorsque vous entrez dans FORTH, vous êtes accueilli par le message : ORIC-FORTH V1 OK. Vous devez alors entrer la commande : EMPTY-BUFFERS (Return). Cette commande initialise le tampon pour la cassette (des explications sont données plus loin). L'oubli de cette commande bloquera les entrées/sorties.
Si vous tapez maintenant VLIST, Forth listera toutes les commandes (encore appelées mots) qu'il peut comprendre. Un Control-C permet de stopper cette édition.
Vous pouvez alors essayer la séquence suivante:2 3 * .(Return)
En mettant un espace entre chaque mot. La réponse 6 va

s'écrire à la droite du point. Ce que vous avez fait a été de donner à l'interpréteur Forth deux nombres, 2 et 3. Ceux-ci ont été empilés dans la pile au moment où ils ont été entrés. Le*(multiplication) est un opérateur qui multiplie les deux nombres entrés dans la pile et les remplace par le résultat qui se trouve alors au sommet de la pile. Enfin, la commande .(point) prend le nombre situé au sommet de la pile et l'imprime sur l'écran.
Si vous tapez maintenant : .(Return) vous obtiendrez le message d'erreur 'EMPTY STACK' (pile vide) puisqu'il n'y a plus rien à écrire.

Les entrées/sorties sur cassette.

FORTH a été conçu pour travailler avec des disques comme mémoire de masse, et le disque est géré de façon à fournir une mémoire virtuelle. Cela veut dire que le disque joue le rôle d'une extension de la mémoire normale. La méthode du FORTH consiste à imaginer le disque comme étant constitué de blocs de un kilo-octets numérotés de 1 à N, N étant fonction de la capacité du disque. Si l'utilisateur demande à ce que ce bloc 3 soit utilisé, alors ce bloc est chargé depuis le disque dans un tampon en mémoire vive. Si le tampon en mémoire vive contient déjà un bloc recopié du disque, ce tampon est recopié sur le disque avant que le nouveau bloc ne soit chargé en mémoire. (En fait il n'est recopié sur le disque que s'il a été modifié).

Avec cette façon de faire, la totalité du disque est accessible d'une manière directe, rapide et simple à utiliser. Si l'utilisateur a besoin d'un bloc, il apparaît dans le tampon. Les opérations de lecture et d'écriture sont automatiques et sont exécutées par un ensemble de mots FORTH connus globalement sous le nom de gérant de disque virtuel Forth.

Ces blocs de 1 kilo-octets sont également connus sous le nom d'écrans, parce qu'ils peuvent être affichés sur un écran normal en remplissant entièrement un écran de 16 lignes de 64 caractères. Cela n'est pas tout à fait vrai pour l'ORIC !

Adaptation pour cassette sur ORIC

Pour permettre un système similaire avec une cassette qui ne sacrifie pas la vitesse et la flexibilité de ce mécanisme, ni ne rende cette version de FORTH incompatible avec une utilisation d'un disque futur, la méthode suivante a été utilisée.
Un bloc de 7 kilo-octets a été réservé en mémoire vive pour simuler un micro-disque. Les sous-programmes normaux de gestion du disque considère ces 7 kilo-octets comme un disque et ils remplissent et mettent à jour des secteurs de 128 octets d'un tampon d'un disque ordinaire.
Les commandes de la cassette peuvent maintenant charger ce « microdisque » de 7 Koctets par unité d'écran de 1 Koctets, qui sont manipulés par Forth d'une manière tout à fait normale, sans restriction concernant son contenu, que ce soit du texte, des données, ou ce que l'utilisateur désire.

Commandes des cassettes

CLOAD charge des écrans de 1 Koctets depuis la cassette vers le tampon du « microdisque ». Le numéro de l'écran de début et celui de fin doivent être introduits dans la pile, par exemple :
1 3 CLOAD (Return) : charge les écrans 1, 2, 3 depuis la cassette.
1 3 CSAVE (Return) : stocke les mêmes écrans sur la cassette.
SPEED est une variable permettant de contrôler la vitesse de l'enregistrement ou de la lecture. Lorsqu'elle vaut 0, les transferts se font en mode FAST, lorsqu'elle est à 1, les transferts se font en mode SLOW. Par exemple :
O SPEED ! (Return) : les transferts sont en mode FAST.
Remarquez que CLOAD, CSAVE, SPEED sont comme toutes les commandes FORTH et peuvent être utilisées soit à partir du clavier, soit à l'intérieur d'un programme.
Le contenu complet de la cassette est décrit dans une annexe.

Chapitre 2.

LES QUATRE NOTIONS FONDAMENTALES

Ces quatres notions sont :
— les deux piles
— la notation postfixée
— le dictionnaire
— la mémoire virtuelle.
Avant de commencer d'étudier ce langage, il est essentiel d'avoir compris chacune de ces notions présentées ci-dessus. Si vous avez déjà utilisé une calculatrice Hewlett-Packard, les deux premiers points vous sembleront familiers.

Les piles.

FORTH utilise deux piles de nombres, ces piles sont du type dernier entré, premier sorti (LIFO). Elles peuvent être dessinées comme un casier de rangement vertical avec un ressort au fond. Lorsque vous poussez un nouvel élément dans le casier, vous poussez tous les éléments existant déjà vers le bas, et mettez le nouveau au sommet. Lorsque vous prenez l'élément situé au sommet du casier, tous les autres sont tirés et remontent pour que le prochain élément soit disponible.
Ceci est illustré dans la figure 1. Il est très important de remarquer que c'est le plus récent élément introduit qui est le premier à sortir du casier.

Du fait de la façon dont la plupart des microprocesseurs travaillent, à l'intérieur de la mémoire de la machine, les piles travaillent habituellement à l'inverse de cette description ; c'est-à-dire que les éléments sont ajoutés ou retirés à partir du fond de

la pile, comme le montre la figure 2. Remarquez que le pointeur« > » se déplace pour indiquer à tout moment où se trouve le sommet de la pile.

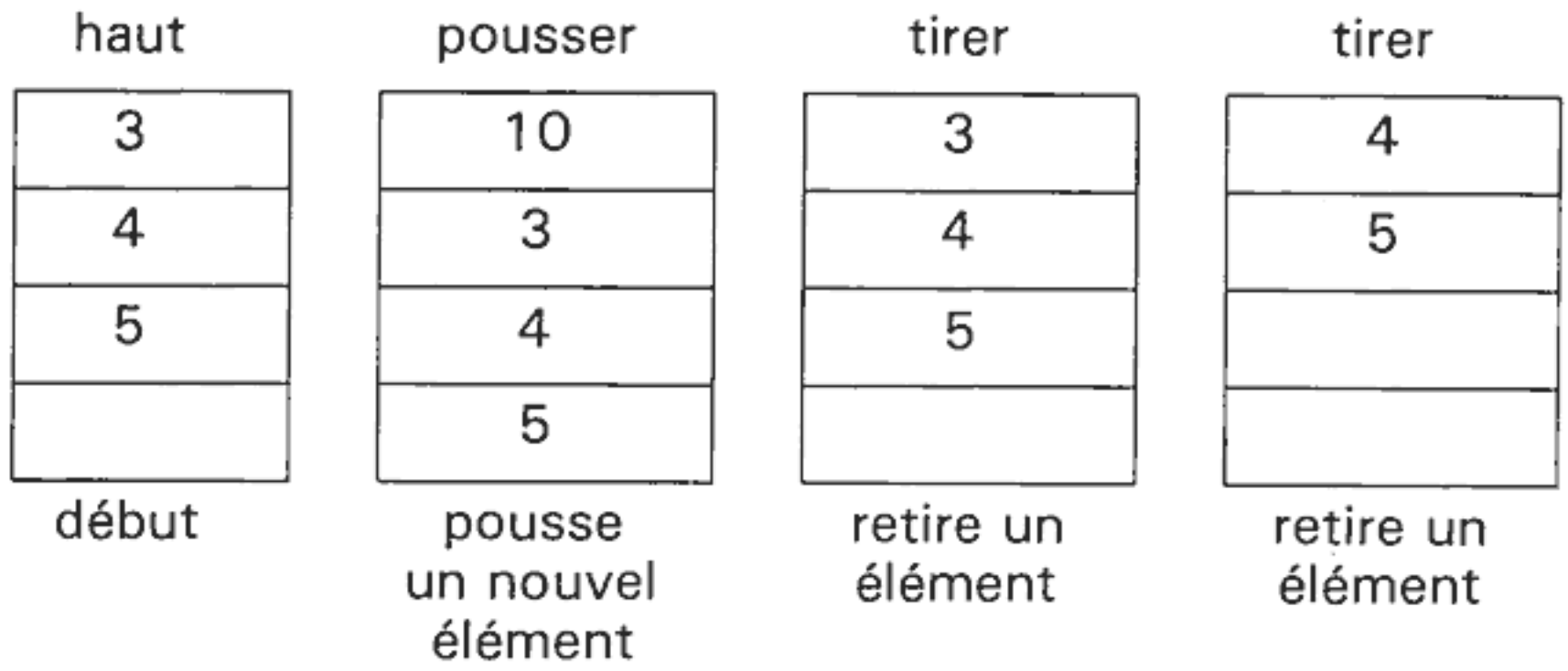


figure 1. une pile push-down

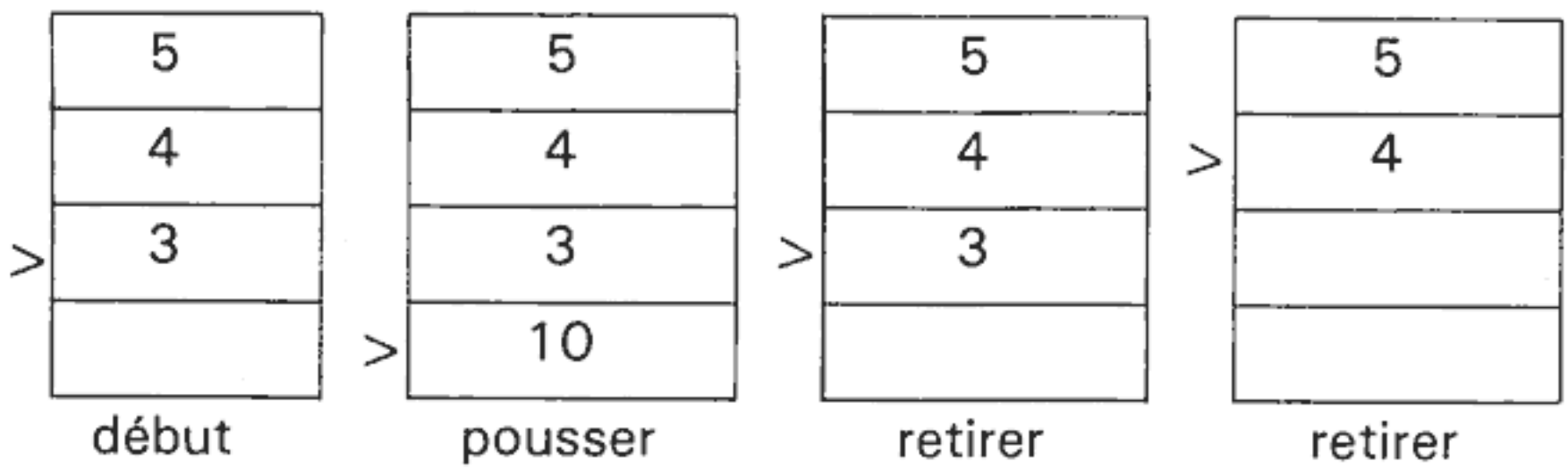


Figure 2.

La façon dont on visualise les piles n'a pas vraiment d'importance, la plupart des gens utilisent des schémas identiques à la figure 1.

La pile des retours.

C'est la pile conventionnelle de retour des sous-programmes, et pour le 6502 elle occupe les adresses depuis ≠ 1FF jusqu'à ≠ 100 (en hexadécimal). FORTH y stocke ces informations de liens d'une manière courante, et aussi d'autres éléments de temps à autre. Le registre S du microprocesseur effectue le travail du pointeur « > » dans l'exemple de la figure 2. Cette pile est toujours appelée par son nom entier : pile des retours.

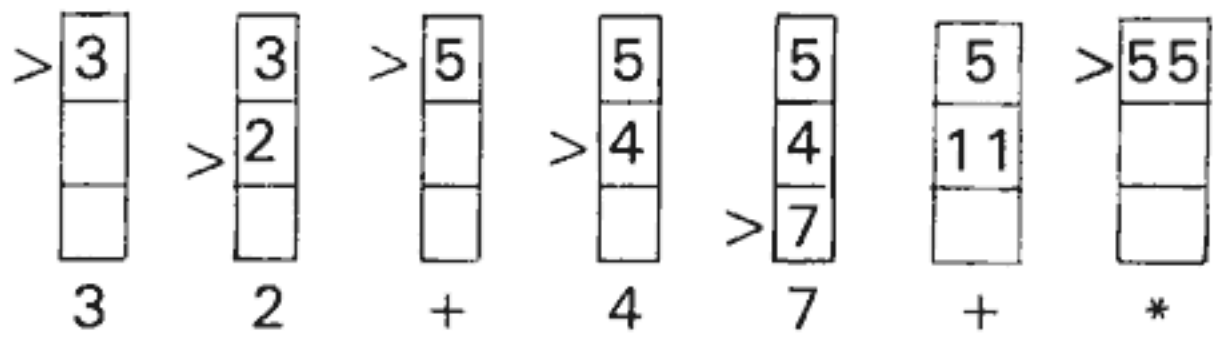
La pile des paramètres.

Tous les calculs et les opérations sont effectués sur les éléments situés au sommet ou derrière le sommet de cette pile. Les éléments manipulés sont presque toujours des entiers sur 16 bits, bien que la double précision (32 bits) puisse être aussi utilisée. Les manipulations sur un simple octet se feront dans un entier de 16 bits, où l'on aura forcé les 8 bits inutilisés à zéro.
Par exemple, l'opérateur + additionne deux entiers. Il retire les deux entiers les plus hauts dans la pile, les additionne, et met (pousse) un seul entier de 16 bits comme réponse dans la pile. Il est possible de transférer des nombres d'une pile à l'autre, (en prenant des précautions), et de manipuler les positions relatives des quatre premiers nombres de la pile.
La pile des paramètres occupe la plupart de la page zéro et utilise le registre X comme pointeur de pile.

Notation postfixée.

C'est le concept qui fait mettre les opérateurs arithmétiques APRES les paramètres. C'est plus facile à voir sur un exemple simple.

Considérez les deux instructions suivantes :
3 + 2 = notation algébrique
3 2 + notation postfixée (3 espace 2 espace +)
La première est l'instruction normale, la suivante peut être lue ainsi :
Lisant de gauche à droite, FORTH rencontre d'abord l'élément 3. Celui-ci, ainsi que tous les nombres explicites, est poussé dans la pile. Il en est de même pour le second élément, le 2. La prochaine chose est le « + » que FORTH interprète ainsi :
Prendre les deux nombres au sommet de la pile, les additionner, et retourner la réponse dans la pile.
C'est plus rapide que la forme algébrique, où la ligne doit être parcourue après l'opérateur (+ dans cet exemple) pour s'assurer que toutes les variables ou arguments ont bien été localisés.
La notation postfixée nécessite que les arguments existent (dans la pile) avant que l'opération soit appelée.
Essayez cet exemple : 3 2 + 4 7 + * qui laisse 55. Regardez les illustrations de la pile ci-dessous.



Pour obtenir l'affichage du résultat, il faut taper un point (.) suivi de < RTN >

Remarquez que les sommes intermédiaires 5 et 11 ont été laissées dans la pile et sont prêtes pour l'opérateur de multiplication sans aucune intervention.

Dans la facon normale (algébrique), cela aurait dû être écrit :
(3 + 2) * (4 + 7) =
Remarquez aussi que la notation postfixée ne nécessite pas de parenthèses. Cela parce que l'ordre dans lequel les opérations sont exécutées n'est fixé que par l'ordre dans lequel VOUS les demandez, et NON par quelque règle arbitraire de priorité.
J'espère que vous voyez maintenant comment cette notation

convient parfaitement à une machine fondée sur une pile, et comment elle accroît l'efficacité et la rapidité du programme. Il est surprenant que la plupart des langages sont en notation postfixée interne et passent beaucoup de temps à utiliser du code pour transformer la notation algébrique en notation postfixée, cela pour la prétendue commodité de l'utilisateur. Les compilateurs FORTRAN entrent dans cette catégorie.
L'utilisation de la pile permet également de diminuer l'utilisation de variables pour stocker les résultats intermédiaires, ou pour passer des arguments entre programmes. Les arguments pour les programmes sont passés dans la pile, bien sûr. Des variables ayant un nom peuvent être créées s'il est nécessaire, mais en général, moins il y en a, mieux c'est.

Le dictionnaire.

FORTH est un langage presque composé exclusivement de procédures ressemblant à des sous-programmes. Lorsqu'une procédure est exécutée, une adresse de retour est « poussée » dans la pile des retours et cette adresse est retirée lorsque l'on quitte la procédure.

En FORTH les procédures sont appelées MOTS (et de quoi donc est composé un langage ?), et chaque mot a un NOM distinct comportant jusqu'à 31 caractères (espace, retour-chariot, ligne-suivante, NULL sont exclus).
L'ensemble des mots composant ce langage est le DICTIONNAIRE constitué d'une liste chaînée.
Lorsqu'un mot est utilisé, le dictionnaire est parcouru depuis le haut (le mot le plus récent) jusqu'à la fin pour trouver le mot recherché. Ce qui est « trouvé » est l'adresse de départ du mot. Une fois trouvé le mot est alors exécuté, si l'on est en mode exécution, ou bien compilé si l'on est en mode compilation. De toute façon, l'adresse de départ est tout ce qu'il est nécessaire de connaître pour l'exécuter ou le compiler.

Le fichier FORTH contient environ 250 mots, quelques-uns en langage machine, mais la plupart sont écrits en FORTH. Oui, le langage est écrit en lui-même.

Le concept général de la programmation en FORTH est de construire de nouveaux mots qui consistent en une suite d'appels aux mots existants. La séquence d'appels est effectuée lorsque le nouveau mot est exécuté.
Bien sûr le nouveau mot peut être à son tour appelé par de futurs nouveaux mots, ce qui accroît la complexité de chaque procédure (mot) jusqu'à ce que toute votre application soit contenue dans un mot.
Chaque nouveau mot est introduit dans le dictionnaire, de telle façon que sa structure soit identique au reste du langage.
La programmation étend littéralement le langage (dictionnaire), pour générer un nouveau, dictionnaire étendu de mots qui peuvent effectuer la fonction désirée.
Il est possible de sauvegarder le nouveau dictionnaire étendu en tant que nouvelle version du langage qui peut être chargée et exécutée directement. C'est-à-dire que vous avez la possibilité de créer différents FORTH, pour des applications spécifiques.
Le contenu du dictionnaire et sa structure seront explicités avec plus de détails dans un prochain chapitre.

Mémoire virtuelle.

L'ORIC-FORTH possède un système d'exploitation de disque très simple, simulé en mémoire vive. Le disque est vu comme constitué de blocs numérotés, chaque bloc contenant 1 Koctets indépendamment de la taille réelle d'un secteur du disque.
Les blocs sont numérotés à partir de 0 jusqu'à la capacité du disque.
Si l'utilisateur veut accéder au bloc « n », une opération simple amènera ce bloc dans un buffer en mémoire vive, où il peut être manipulé. Si le bloc est modifié de quelque façon, la nouvelle version du bloc sera réécrite sur le disque automatiquement (uniquement s'il a été altéré).
Ce simple agencement permet que la totalité du disque puisse être adressée comme si c'était de la mémoire vive, l'adressage consistant en deux parties :
Le numéro du bloc.

L'adresse de l'octet (de 0 à 1 023) à l'intérieur du bloc.

Chapitre 3.

ALLONS-Y, AVEC QUELQUES EXEMPLES

Assurez-vous que vous avez bien chargé FORTH, et obtenu le message 'OK'. Ensuite tapez EMPTY-BUFFERS (Return) pour initialiser les tampons d'entrée/sortie. FAITES TOUJOURS CELA après un démarrage à froid à moins d'avoir des raisons spécifiques pour ne pas le faire.

1. Que répondre à OK.

Le système est en mode « entrée de données », et il attend une entrée au clavier, ainsi que l'indique la présence du curseur. Tout ce que vous tapez est tout d'abord stocké dans un tampon d'entrée (T.I.B. pour Terminal Input Buffer), de 80 caractères de long. Rien ne se passe réellement tant que vous n'avez pas fait un retour-chariot (Return). Si vous essayez de rentrer plus de 80 caractères, le programme d'entrée de données stoppera l'entrée de données en faisant un retour-chariot pour vous. Une fois que vous avez tapé le (Return), l'interpréteur du FORTH commence son travail en parcourant la ligne introduite telle que vous l'avez introduite.

L'interpréteur recherche des groupes de caractères *séparés par des espaces:*

— S'il trouve un mot FORTH, il est alors exécuté.

— S'il trouve un nombre, ce nombre est poussé au sommet de la pile.

— S'il ne peut reconnaître ce que vous avez entré, il arrête son travail, revient dans le mode « entrée de données » en affichant un « ? » précédé du mot qu'il n'a pas compris.

2. Arithmétique simple.

Si vous entrez 2 3 + . (Return) alors vous verrez 5 OK. Que s'est-il passé ? Les nombres '2' et '3' ont été poussés dans la pile. Le '+' signifie 'additionner les deux nombres au sommet de la pile et laisser le résultat au sommet de la pile'. Enfin '.' signifie 'imprimer le sommet de la pile en le considérant comme un nombre'.
Essayer : 45 + 67 + * . vous obtiendrez 117 OK

3. Base de numération.

FORTH peut travailler dans différentes bases de numération, qu'il peut changer n'importe quand. Vous pouvez l'utiliser comme un calculateur OCTAL/DECIMAL/HEX/BINAIRE. Lors du démarrage à froid, FORTH travaille en décimal. En tapant HEX (Return) vous le transformez en une machine hexadécimale. Un DECIMAL (Return) permet de revenir en décimal.

Dans ce qui suit les parties soulignées indiquent ce que vous devez frapper *(terminé par un Return)* ; FORTH finit toujours par OK pour montrer qu'il a fini de traiter la suite de commandes et qu'il est prêt pour d'autres demandes.

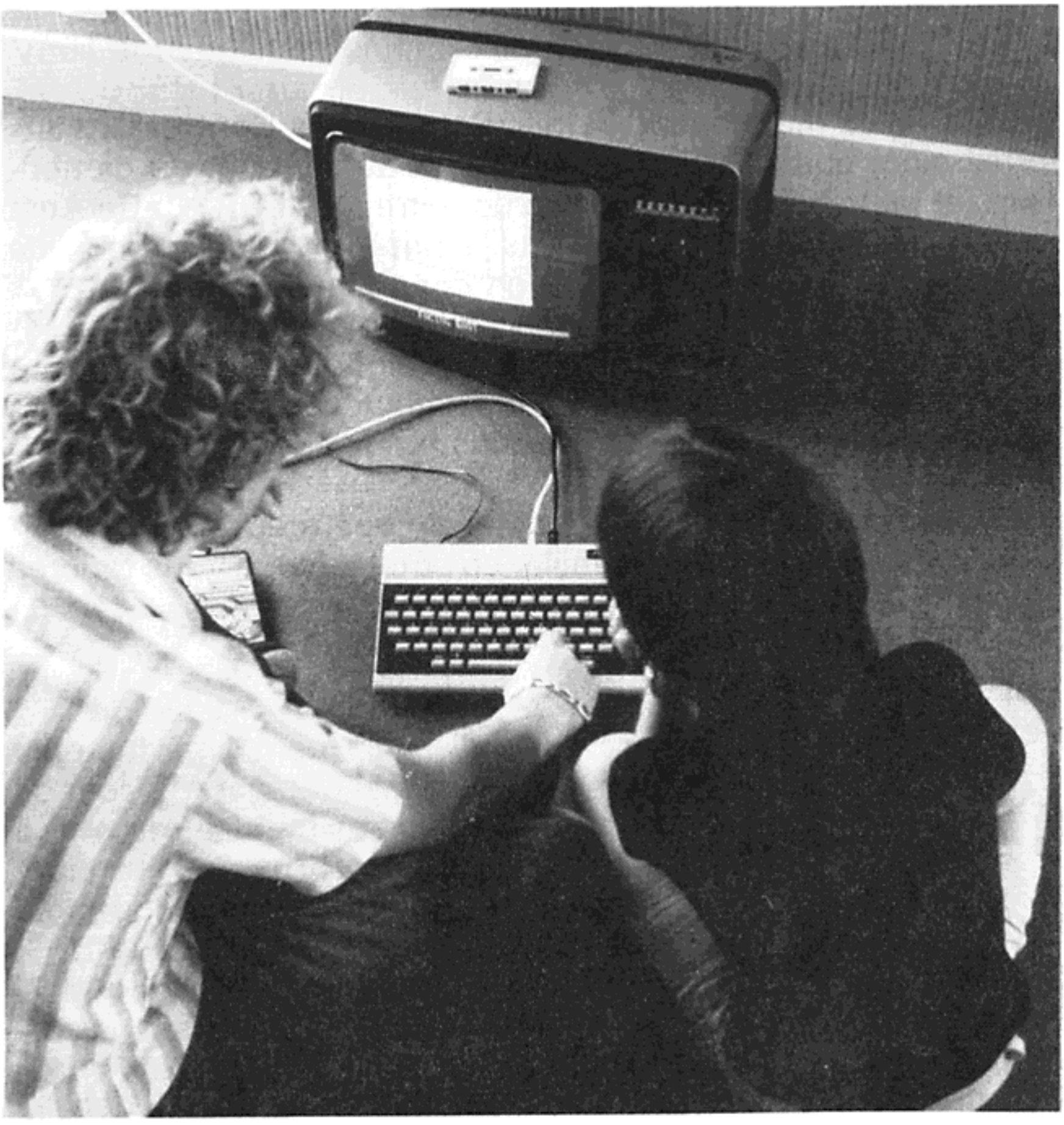
```
HEX OK                                  (addition hexadécimale)
3BE8 C8 + . 3CB0 OK                     (multiplication hexadécimale)
25 2F*. 6CB OK                          (décimal en hexadécimal)
DECIMAL 1348 HEX . 544 OK               (décimal en hexadécimal)
```

La base est modifiée en stockant la valeur correcte dans une variable dont le nom est BASE, aussi :

```
8 BASE ! OK                             (stocke 8 qui veut dire octal).
6 3 * . 22 OK                           (multiplication octale)
22 DECIMAL . 18 OK                      (octal en décimal)
```

4. Ajout d'un nouveau mot dans le dictionnaire.

Jusqu'à maintenant tout ce que vous avez tapé a été immédiatement exécuté après que vous ayez tapé (Return). Pour pouvoir ajouter un nouveau mot dans le dictionnaire, FORTH



doit être changer en mode « Compilation », de telle façon que la suite de la ligne soit compilée dans le dictionnaire au lieu d'être exécutée.

Supposez que vous voulez créer un nouveau mot qui prend un nombre au sommet de la pile et qui y retourne le cube de ce nombre, aussi nous pouvons essayer cela depuis notre clavier :

```
2 DUP DUP * * . 8 OK
```

Explication : '2' est poussé dans la pile, DUP DUP en fait deux copies qui sont également mises dans la pile, * * multiplie les trois nombres ensemble, laissant le cube au sommet de la pile, pour que '.' puisse l'imprimer. Pour que ceci fasse partie du dictionnaire, tapez ce qui suit :

```
: CUBE DUP DUP * * ; OKsuite
5 CUBE . 125 OK
```

Si vous tapez maintenant VLIST (Return) puis Control C, vous verrez alors que CUBE est maintenant au sommet du dictionnaire, et peut être utilisé comme un autre mot FORTH. Ceci est obtenu par le couple deux-points, point-virgule qui permet de compiler un nouveau mot dont on donne la définition.

: ABCDsignifie 'voici un nouveau mot appelé ABCD'. Le mot Forth qui suit est alors compilé dans le dictionnaire sous le nom ABCD. Enfin le ';' signifie 'c'est la fin du nouveau mot'. Rappelez vous l'exemple sur la base octale, vous pouvez maintenant faire :

: OCTAL 8 BASE ! ; OK pour définir un opérateur qui positionnera la base à 8, et ainsi travailler en octal lorsque vous taperez OCTAL (Return).

De même vous pouvez faire :

: BINAIRE 2 BASE ! ; OK

Veiller à laisser un espace après les « : ».

5. La boucle DO.....LOOP

C'est une simple boucle avec un indice de comptage (un peu comme la boucle FOR....NEXT en BASIC).

DO prend deux variables au sommet de la pile ; la valeur initiale de l'indice au sommet, et la valeur finale plus 1 dans le second élément.

Exemple : ('I' retourne l'indice courant)

```
DECIMAL OK                    (à écrire sur une seule ligne)
: 10-CUBES                    (imprime la table des cubes
                               de 0 à 9)
10 0 DO                       (initialise le début
                               et la fin de la boucle)
CR I . I CUBE .               (imprime le nombre et son
                               cube)
LOOP CR                       (fin de la boucle et
                               passage à la ligne suivante)
; OK                          (fin du nouveau mot)
```

Pour finir, n'oubliez pas 'Return'

Vous pouvez maintenant exécuter ce nouveau mot :

```
10-CUBES (Return)
0  0
1  1
2  8
3  27
4  64
5  125
6  216
7  343
8  512
9  729
```

6. Le test IF... ELSE... ENDIF

On peut également avoir IF...ELSE...THEN
Le mot 'IF' regarde le sommet de la pile (et le supprime).
Il interprète la valeur comme étant fausse (égale à zéro) ou vraie (différente de zéro), et exécute la partie appropriée de l'instruction conditionnelle comme l'indique ce schéma :
IF ... exécuté si sommet vrai ... ENDIF partie suivante
IF ... partie si vrai ... ELSE ... partie si faux ... ENDIF suite
Aussi voici un exemple qui retourne la valeur absolue du sommet de la pile. Remarquez que '0< est un test du sommet de la pile qui laisse une valeur vraie si le sommet de la pile est inférieur à zéro, c'est-à-dire négatif.

```
: ABS-VALUE
```

```
DUP O<   (copie le nombre et teste son signe)
IF MINUS ENDIF     (change le signe si négatif)
; OK
```

Vous pouvez maintenant essayer :

```
10 ABS-VALUE . 10 OK
-5 ABS-VALUE . 5 OK
```

7. La boucle BEGIN .... UNTIL

Cette boucle prend une valeur soit vraie, soit fausse comme argument, généralement calculée dans la boucle, qui est testée par UNTIL. Si la valeur est fausse, le programme revient au mot BEGIN. Si elle est vraie, le programme continue la partie se trouvant derrière le UNTIL.

Exemple :

```
: 10-CUBES                    (nom de ce mot)
O                             (initialise le compteur)
BEGIN                         (début de la boucle)
CR DUP . DUP CUBE .           (imprime le compteur et son
                               cube)
1 +                           (incrémente le compteur)
DUP 10 =                      (teste si le compteur a atteint
                               10)
UNTIL                         (fin de la boucle, sortie si vrai)
CR DROP                       (supprime l'index)
; OK                          (fin du mot)
```

On peut maintenant l'exécuter :

```
10-CUBES
0  0
1  1
2  8
3  27
4  64
5  125
6  216
7  343
8  512
9  729
OK
```

8. Entrée/sortie de texte.

La sortie de chaînes de caractères se fait généralement par l'intermédiaire du mot TYPE pour les chaînes générées d'une façon interne, ou .« TEXTE » pour imprimer 'TEXTE' qui est une chaîne ne pouvant être manipulée. Des opérateurs supplémentaires pour la sortie de texte sont :
-TRAILING
EMIT
SPACE
SPACES
et ERASE, FILL, BLANKS pour initialiser les zones de stockage des chaînes.
Remarquez qu'en FORTH, toutes les chaînes sont stockées avec leur premier octet rempli par leur longueur donc le maximum de caractères dans une chaîne est 255.
L'introduction de texte à partir du clavier fait appel aux mots : QUERY, EXPECT, et EDITOR avec TEXT qui peut transférer des chaînes introduites au clavier dans un tampon dont l'adresse est PAD (une variable).
La comparaison entre chaînes peut être faite en utilisant les mots de l'éditeur TEXT et MATCH.

9. Introduction et sortie de nombres.

Toutes les entrées/sorties sur des nombres sont faites dans la base courante, aussi assurez-vous de sa valeur lorsque vous programmez des entrées/sorties de nombres.
Le mot principal permettant l'introduction de nombre est NUMBER, qui prend une chaîne de caractères à une adresse donnée et essaye de la convertir en un entier en double précision.
Lorsque vous tapez 123 (Return), c'est le mot NUMBER qui convertit la chaîne '1' '2' '3' en binaire et pousse ce nombre au sommet de la pile. Remarquez que 123 (Return) génère un nombre entier en simple précision sur 16 bits.
Si vous tapez 123. (Return), le programme NUMBER reconnait

le point décimal comme une demande d'un entier en double précision sur 32 bits.
Remarquez que 1.23 sera aussi converti en 123, mais la variable DPL sera chargée par 2 pour indiquer que deux chiffres ont été trouvés derrière la virgule lors de la conversion.
Afin de rendre l'introduction des nombres un peu plus facile, un nouveau mot IN≠ existe sur la cassette extension écran 1, qui fait tout ce qui est nécessaire pour acquérir des entiers en simple précision à partir du clavier et mettre le résultat dans la pile.

Sortie des nombres.
Pour imprimer un nombre il doit tout d'abord être transformé en chaîne. Les opérateurs '.' '.R' 'D.' et 'D.R' font cela pour vous pour des sorties standard. Elles utilisent des opérateurs de conversion et de formattage qui sont disponibles pour des conversions spéciales. Ces opérateurs sont :

```
<# #S # HOLD SIGN #>
```

Vous pouvez remarquer que les nombres issus de la conversion prennent place de droite à gauche.
Ces primitives travaillent toujours sur des nombres en double précision.
La chaîne est générée en descendant à partir de l'adresse PAD.
L'exemple suivant vous montre un échantillon de ce que vous pouvez faire : il prend un entier sur 16 bits au sommet de la pile et l'imprime en heures, minutes et secondes..
Tout d'abord nous avons besoin d'un mot qui insère le caractère : dans la chaîne :
HEX : ':' 3A HOLD ; DECIMAL
Cela définit un nouveau mot appelé ':' qui fait cela. (En hexa le code du caractère : est $3A).
Maintenant il nous faut un opérateur pour convertir en unités de soixante (pour les secondes et les minutes).

```
: :00#6 BASE ! # `:' DECIMAL ;
```

Le mot :00 travaille ainsi :
convertit le chiffre le moins significatif en base 10
6 BASE ! positionne la base 6
convertit le chiffre suivant en base 6
':' insère le symbole :
DECIMAL restaure la base 10
Nous pouvons maintenant faire :

```
: TIME 0<#:00 :00### >TYPE SPACE ;
```

Le mot TIME attend une valeur au sommet de la pile. Il ajoute

un zéro à la pile pour convertir le nombre en double précision.
<#indique le début du programme de conversion des nombres
:00 convertit la partie basse du nombre en base 60 (pour les secondes)
:00 fait de même pour les minutes
#convertit les deux chiffres suivants (en décimal) pour l'heure
#>signifie fin de la conversion
TYPE imprime ensuite la chaîne résultat de la conversion
Ainsi 65 TIME (Return) imprimera 00:01:05
Remarquez que les opérateurs # #S SIGN HOLD ne peuvent être utilisés qu'entre les symboles<# et#>.

Conclusion.

Maintenant si vous tapez VLIST, vous verrez que vous avez ajouté quelques mots dans le dictionnaire. Vous pouvez faire une ou deux choses. Vous pouvez les oublier (FORGET) pour libérer l'espace en mémoire, où, si c'est un programme d'application, vous pouvez modifier le paramètre de chargement pour que les nouveaux mots introduits fassent partie intégralement du vocabulaire FORTH, ceci d'une façon permanente, vous pouvez alors sauvegarder le nouveau dictionnaire comme étant une nouvelle version du langage. (Plus de détails vous seront donnés dans une annexe).

Chapitre 4.

L'EDITION ET LA CREATION DE PROGRAMMES SOURCES

Nous venons de voir comment l'on pouvait donner des commandes au FORTH, et créer de nouveaux mots, nous allons maintenant voir comment faire un programme source particulier, en utilisant l'Editeur.
Tout d'abord l'Editeur doit être chargé depuis la cassette. Il existe dans les écrans de 1 à 7 sur la cassette source, par conséquent après avoir correctement positionné la cassette, et choisi la bonne vitesse, vous tapez 1 7 CLOAD (Return) et démarrez le magnétophone.
Les sept écrans sont maintenant chargés, et le message OK est apparu. Si vous souhaitez voir le texte du source, vous pouvez faire 1 LIST (Return), ce qui affichera les quatre premières lignes de l'écran 1. N'importe quelle touche sauf ↓ fera afficher les quatre lignes suivantes, ↓ jusqu'à la fin. Aussi 1 7 INDEX (Return) affichera les lignes de commentaires au début de chaque écran, de 1 à 7.

Le texte source peut maintenant être compilé dans le dictionnaire FORTH, simplement en faisant 1 LOAD (Return).
Les sept kilo-octets du texte source sont maintenant compilés (ce qui prend de 30 à 40 secondes) en 1,5 Koctets de code objet FORTH. Deux messages « XXXX ISN'T UNIQUE » sont générés (n'en tenez pas compte, et enfin le message « EDITOR LOADED » apparaîtra.
Pour utiliser l'éditeur, tapez maintenant EDITOR (Return). Vous pouvez alors choisir un bloc disponible pour y mettre votre nouveau programme (par exemple le bloc 4). Pour le vider complètement, vous pouvez taper :
4 CLEAR (Return) ou pour voir ce qu'il contient tapez
4 LIST (Return) ce qui affiche chaque bloc (également appelé

un écran) 4 lignes par 4 lignes (appuyez sur n'importe quelle touche pour passer au groupe de 4 lignes suivant ou ↓ pour aller jusqu'à la fin de l'écran).
Chaque « écran » est constitué par les lignes de 0 à 15, chacune pouvant contenir 64 caractères.
Pour entrer du texte en ligne 0, entrez la commande :
0 NEW (Return) qui « ouvre » la ligne 0, pour l'entrée de texte.
Tout ce que vous taperez avant le prochain (Return) sera mis dans cette ligne.
Par convention la ligne 0 de chaque écran, contient une ligne commentaire qui en décrit le contenu, aussi vous pouvez entrer :
(CET ECRAN SERT D'EXEMPLE) (Return)
L'éditeur vous invite alors à rentrer la ligne 1. Si vous voulez laisser cette ligne blanche tapez un espace suivi de (Return) et c'est la ligne 2 qui pourra être remplie. Supposons que vous voulez entrer la définition de CUBE du chapitre précédent :
: CUBE DUP DUP * * ; (n --- cube de n)
Si c'est tout ce que vous voulez entrer dans cet écran vous tapez (Return) directement à la demande d'introduction d'une nouvelle ligne. Vous pouvez maintenant essayer la commande L pour afficher votre nouvel écran, et essayer d'autres commandes de l'éditeur.
Vous pouvez maintenant charger votre nouvel écran, et additionner les mots qu'il contient dans le dictionnaire en faisant 4 LOAD.
Dans le paragraphe suivant quelques-unes des commandes de l'éditeur et leurs effets sont décrits.
Remarque : lorsque vous avez fini d'éditer un écran, entrez la commande FLUSH (Return) pour vous assurer que cet écran a bien été sauvegardé sur disque.

Edition d'écran.

Ces commandes opèrent sur un écran entier :
n LIST (Return) affiche l'écran n
n CLEAR (Return) efface l'écran n (remplissage par des blancs)
n1 n2 COPY (Return) recopie de l'écran n1 dans l'écran n2
L liste l'écran courant, et la ligne courante
FLUSH force l'écran modifié à être sauvegardé sur le disque.

Edition de lignes.

Ces commandes opèrent sur une ligne présélectionnée à l'intérieur de l'écran courant. Une zone tampon en mémoire vive appelée PAD (Bloc-notes) est alors utilisée. Cette zone PAD se trouve toujours 68 octets plus haut en mémoire que le sommet du dictionnaire.

n H (Return) copie la ligne n dans le tampon PAD

n D (Return) copie la ligne n dans le PAD et supprime cette ligne de l'écran courant. Les lignes de n + 1 à la fin de l'écran sont remontées, une nouvelle ligne 15 est ajoutée, vide.

n T (Return) édition de la ligne n sur le terminal, en la sauvant dans le PAD.

n R (Return) remplacement de la ligne n par celle contenue dans le PAD.

n I (Return) insère la ligne contenue dans le PAD à la ligne n, les lignes de n à 14 sont descendues et la ligne 15 perdue.

n E (Return) remplace la ligne n par des blancs (effacement)

n S (Return) les lignes n à 14 sont descendues, laissant la ligne n vide, la ligne 15 étant perdue.

Edition de chaîne et contrôle du curseur.

Les opérations d'édition de chaînes de caractères se font par référence à un curseur d'édition, affiché par un caractère '■'. Au départ, le curseur de l'éditeur est positionné en haut de l'écran. Pour revenir à notre exemple précédent, où la définition de CUBE a été introduite à la ligne 2, 2 T (Return) affiche la ligne 2, avec le curseur au début de la ligne.

■ : CUBE DUP DUP * * ;

Pour trouver la chaîne DUP, tapez F DUP (Return). L'écran est

parcouru en avant depuis la position du curseur pour trouver la position de la chaîne demandée, et lorsqu'elle est trouvée, l'affichage est ainsi :
: CUBE DUP ■ DUP * * ; (pas de blanc entre DUP et ■).
En faisant N (Return) on recherche la prochaine occurence de la chaîne (Next), on aura alors : CUBE DUP DUP ■ * * ;
En faisant B (Return) ont fait déplacer le curseur en arrière de la longueur de la chaîne trouvée :
: CUBE DUP ■ DUP * * ;
Remarquez que le curseur peut également être déplacé par la commande M précédée d'un nombre.
Lorsque vous avez localisé la partie de la ligne où vous voulez travailler, les commandes suivantes vous permettent de supprimer ou de modifier des chaînes de caractères.
X DUP (Return) recherche et détruit la chaîne DUP :
: CUBE DUP ■ * *
C DUP (Return) copie la chaîne qui suit à la position du curseur :
: CUBE DUP DUP ■ * * ;
D'autres commandes telles que TILL texte et n DELETE sont expliquées dans le glossaire.

Commandes d'introduction de texte.

Après avoir sélectionné l'écran pour l'édition (soit l'écran 4), en faisant 4 LIST (Return) ou 4 CLEAR (Return), les commandes suivantes sont disponibles pour l'insertion de texte. 1 P ce texte ira en ligne 1 (Return).
P signifie « introduire une nouvelle ligne » (Put) et le nombre précédent est le numéro de la ligne désirée (1 dans cet exemple), cette introduction écrasant le texte précédemment contenu par la ligne. La taille maximum d'une ligne est de 64 caractères. Attention de ne pas rentrer une ligne vide. Si vous faisiez par erreur 1 P (Return), un caractère 'null' serait mis dans cette ligne, ce qui génèrerait une erreur plus tard. Si cela vous arrivait, tapez 1 E (Return) pour effacer complètement la ligne.

n NEW (Return) sélectionne la ligne n pour une entrée de texte. L'écran est alors affiché, avec les numéros de lignes, jusqu'à la

ligne n, où il s'arrête et vous invite à introduire votre texte. Vous pouvez alors taper votre texte, en le terminant par un (Return), ce qui termine immédiatement l'introduction et le reste de l'écran est affiché.
n UNDER (Return) affiche l'écran jusqu'au début de la ligne n + 1 et attends votre entrée, comme avec NEW. L'ancienne ligne n + 1 et les lignes suivantes sont déplacées vers le bas, la ligne 15 étant perdue.

Chapitre 5.

UN EXEMPLE DE PROGRAMME DE DÉVELOPPEMENT

Un utilitaire d'impression

Posons ces spécifications de départ ainsi :
« Nous voulons imprimer des blocs contigus d'écrans, à raison de 3 blocs par page, avec une numérotation des pages, une ligne de titre et un message d'identification du système. »
FORTH est un langage structuré, c'est-à-dire qu'un problème est résolu plus facilement en partant du programme principal, et en travaillant en descendant, à le décomposer en différentes étapes, en allant de raffinement en raffinement.
Ainsi supposons que nous voulons obtenir une commande :
'début' 'fin' PRINT (Return)
La spécification ci-dessus précise que l'on en imprime 3 par page, mais s'il en reste moins de 3, seuls ceux restant seront imprimés.

Donc la première idée est de faire quelque chose comme cela :

```
: PRINT NUM_PAGE_1
PLUS_DE_3_ECRANS ?
IF  IMPR_PAGE_ENT ELSE IMPR_RESTANT
ENDIF
REPETER_JUSQU'A_FIN ;
```

Bien sûr cela ne marche pas, mais tous les éléments essentiels sont présents. Il nous faut définir une variable pour y mettre le numéro de page :

```
0 VARIABLE P≠
```

Alors NUM PAGE 1 devient 1 P≠ !

Nous devons également mettre l'imprimante en marche ou en arrêt.
Pour un premier essai, nous pouvons avoir :

```
: PRINT 1P≠ ! PR_ON (imprimante en marche)
DEBUT_BOUCLE BEGIN (début de la boucle)
≥3_A_FAIRE ? (y en a-t-il assez pour une page ?)
IF FAIRE_PAGE ELSE FAIRE_RESTE ENDIF
UNTIL (jusqu'à ce que tout soit fait)
PR_OFF (imprimante en arrêt)
```

Nous pouvons maintenant définir d'autres choses :

```
HEX
: PR_ON F57B DUP DUP 1BF4 ! 1C22 ! 1C1D ! ;
: PR_OFF CC12 DUP DUP 1BF4 ! 1C22 ! 1C1D ! ;
DECIMAL
```

Ce qui modifie les interrupteurs d'entrée/sortie pour le port de l'imprimante.
Maintenant comment faire le mot FAIRE_PAGE ? Il va prendre comme arguments le numéro de l'écran de début, le nombre à éditer et retourner les mêmes paramètres mis à jour, c'est-à-dire le début + 3 et nombre-3, ce qui laissera automatiquement les arguments corrects pour la prochaine boucle.
Avec un peu d'essais et d'erreurs, nous obtenons :

```
: FAIRE_PAGE 3 - SWAP 3 + SWAP OVER DUP 3 - PRINT_1 ;
```

Cette procédure ajuste le début et le nombre, et fournit également une valeur début et une valeur fin pour la procédure PRINT_1 qui fera réellement le travail.
De même, FAIRE RESTE devra également retourner les deux valeurs ajustées, sauf un nombre qui sera égal à 0.

```
: FAIRE_RESTE (début fin --- début 0)
>R Ø OVER DUP R> + SWAP PRINT 1 ;
```

Cela marche très bien parce que le nombre restant est au sommet de la pile et il n'est égal à 0 uniquement lorsque l'impression est terminée, il peut donc être utilisé pour tester la sortie de la boucle principale.
Nous devons également modifier les nombres 'début' et 'fin' fournis pour obtenir 'début' et 'nombre' nécessaire pour FAIRE PAGE et FAIRE_RESTE. Ce 'nombre' peut être testé pour voir s'il est supérieur à 2.
Nous avons donc :

```
: PRINT                 (début fin ----)
1 P≠ !                  (page = 1)
PR_ON CR                (imprimante marche,
```

```
                          saut à la ligne)
OVER -1 +                 (change début fin en début nombre)
BEGIN                     début boucle d'impression)
DUP 2>IF                  (test du nombre)
FAIRE_PAGE ELSE           (page entière si>2)
FAIRE_RESTE ENDIF         (sinon le reste)
CR                        (saut de ligne)
DUP 0= UNTIL              (boucle jusqu'à nombre = 0)
DROP DROP CR              (nettoyage variables inutiles)
PR_OFF ;                  (imprimante arrêt)
```

PRINT_1 est la procédure suivante à étudier, elle reçoit comme arguments les valeurs de début et de fin, qui peuvent être utilisées dans une boucle DO .... LOOP

```
: PRINT_1 (début fin ---- impression)
PR_ON DO I IMPRECR LOOP CR 15 MESSAGE CR CR CR CR PR
OFF ;
```

Ceci imprime une page entière ou seulement une partie, appelant IMPRECR pour imprimer un écran, et le message d'identification du système (numéro 15) à la fin.

Nous avons maintenant besoin de IMPRECR qui peut être emprunté entièrement à la fonction LIST :

```
: IMPRECR (n --- impression écran n)
DECIMAL CR DUP SCR ! . « SCREEN » . 16 0 DO
CR I 3 .R SPACE I SCR @ .LINE LOOP CR ;
```

qui acquiert les lignes dans une boucle DO ... LOOP. et les imprime avec .LINE.

C'est pratiquement terminé maintenant et l'utilitaire entièrement terminé est reproduit à la fin de ce chapitre, en utilisant les codes pour une imprimante OKI, qui peut faire des caractères en double hauteur. Regardez comment il fait pour aller chercher et imprimer l'entête avec le numéro de la page.

Un exemple de diagramme de la pile est également reproduit. Il est très utile pour visualiser ce qui arrive dans la pile, et son usage est hautement recommandé.

PRINT UTILITY PAGE 1

SCREEN 3

```
 Ø (UTILITAIRE D'IMPRESSION 1/3 WANB NOV 81)
 1 FORTH DEFINITIONS DECIMAL
```

```
2 Ø VARIABLE P#
3 (LES LIGNES 6 A 8 DEPENDENT DE L'IMPRIMANTE)
4 HEX : PR ON F57B DUP DUP 1BF4 ! 1C22 ! 1C1D ! ;
5 : PR_OFF CC12 DUP DUP 1BF4 ! 1C22 ! 1C1D ! ;
6 : DBLH 1F EMIT ;
7 : NOMH 1E EMIT ;
8 : TOF CR .« READY ? » KEY DROP ;
9 DECIMAL
 1Ø : IMPRECR (n --- imprime écran n)
 11 DECIMAL CR DUP SCR ! .« SCREEN ». 16 0 DO
 12 CR I 3 .R SPACE I SCR @ .LINE LOOP CR ;
 13⟶
14
15
```

```
SCREEN 4

 Ø (UTILITAIRE D'IMPRESSION 2/3   WANB NOV 81)
 1
 2 : IMPTETE (impression en tête de page)
 3 PAD C@ 38 MIN 1 MAX PAD C ! DBLH SPACE
 3PAD COUNT TYPE 31
 4 PAD C@ DUP Ø<IF CR DROP 32 ENDIF SPACES
 5 .« PAGE » P# @ 3 .R CR CR 1 P# + ! NOMH ;
 6
 7 : PRINT_1 (début fin ---- imprime ces écrans)
 8 PR_ON IMPTETE DO I IMPRECR LOOP
 9 CR 15 MESSAGE CR CR CR CR PR_OFF ;
10
11 : FAIRE_PAGE (début nombre -- début +3 nombre-3)
12 3 - SWAP 3 + SWAP OVER DUP 3 - PRINT - 1 ;
13⟶
14
15
```

```
SCREEN 5

 Ø (UTILITAIRE D'IMPRESSION 3/3  WANB NOV 81)
 1
 2 : FAIRE_RESTE (début fin - début Ø)
 3>R Ø OVER DUP R  + SWAP PRINT_1 ;
```

```
 4
 5
 6
 7 : PRINT (n m... impression des écrans n à m)
 8 CR .« ENTETE : » EDITOR ENTER
 9 1 P# !  PR_ON CR OVER - 1 +  BEGIN
10 DUP 2>IF FAIRE_PAGE ELSE FAIRE RESTE ENDIF
11 DUP Ø = TOF UNTIL
12 DROP DROP PR_ON CR CR CR CR PR_OFF ;
13 ; S
14
15
```

ORIC FIG-FORTH

VOIR TABLEAU A PART

Application : utilitaire d'impression

Mot : FAIRE PAGE

PILE					HAUT MOTS
			début	compte	(entrée avec 'début' et 'compte') : FAIRE-PAGE
		début	compte	3	3
			début	compte-3	—
			compte-3	début	SWAP
		compte-3	début	3	3
			compte-3	début + 3	+
			début + 3	compte-3	SWAP
		début + 3	compte-3	début + 3	OVER
	début + 3	compte-3	début + 3	début + 3	DUP
début + 3	compte-3	début + 3	début + 3	3	3
	début + 3	compte-3	début + 3	début	—
			début + 3	compte-3	PRINT-1
					;

Remarques : FAIRE-PAGE

Entrée : numéro d'écran de 'début' et 'compte' à imprimer
Puisqu'une page a 3 écrans, il génère les 'débuts' et 'compte' mis à jour, ainsi que les arguments nécessaires à PRINT-1, c'est-à-dire 'début + 3', 'début' soit la 'Fin' et le 'début' des numéros d'écrans pour une page de 3 écrans.

Chapitre 6.

LA STRUCTURE DU DICTIONNAIRE DE FORTH

Puisque 99 % de FORTH est dans son dictionnaire, cela vaut la peine de regarder sa structure, ce que nous allons faire en prenant comme référence notre exemple favori, la commande CUBE. Lorsque vous tapez :
: CUBE DUP DUP * * ; (Return)
Le nouveau mot CUBE est additionné au dictionnaire. La mémoire est semblable à ce qui suit, où chaque rectangle représente un octet de mémoire. (Voir schéma ci-contre)

Comme vous pouvez le voir, un élément du dictionnaire est constitué d'une section ENTETE et d'une section PARAMETRE. La première contient tout ce qui est nécessaire comme information pour décrire le nom de l'élément et son type ; la seconde partie contient une liste d'adresses qui pointent effectivement sur les mots composant le nouveau mot.
Le bloc entête est subdivisé ainsi :
Champ nom : il débute par un octet de longueur dont les 5 bits les moins significatifs indiquent la longueur de la chaîne ASCII constituant le nom. Le bit le plus significatif de cette longueur est positionné à 1 pour pouvoir l'identifier. Vient ensuite la chaîne ASCII du nom, dont le dernier caractère a également son bit de poids fort positionné à 1.
L'adresse de l'octet de longueur est généralement connue sous le nom d'adresse du champ nom (NFA pour Name Field Adress).
Champ lien : il contient le NFA du mot précédent dans le dictionnaire. Ce champ permet d'avoir un chaînage de tous les mots dans le dictionnaire, ce qui permet de le trouver en ne connaissant que l'adresse du mot le plus récent.
L'adresse de ce champ est l'adresse du champ lien (LFA pour Link Field Adress).
Champ code : c'est ce champ qui définit le « type » du mot et son adresse s'appelle CFA (Code Field Adress). Le CFA d'une définition est également l'adresse où l'exécution du mot commence ; le contenu du CFA est donc l'adresse du code machine réel et exécutable.
Dans cet exemple le CFA contient l'adresse de ' : ', une procédure FORTH qui permet de générer le code approprié à une définition par : tel que CUBE.
Tous les mots compilés par ' : ' auront cette adresse comme

#84	NFA : adresse du champ nom
C	
U	
B	
E +#80	
B lien	LFA : adresse du Champ lien
H lien	
B adresse de	CFA : adresse du champ code
H :	
B adresse de	PFA : adresse du champ paramètre.
H DUP	
B adresse de	
H DUP	
B adresse de	
H *	
B adresse de	
H *	
B adresse de	
H ;	
	Prochain octet libre.

En-tête du dictionnaire pour le mot « CUBE »

Bloc des paramètres de « CUBE » donnant la liste des opérations à effectuer.

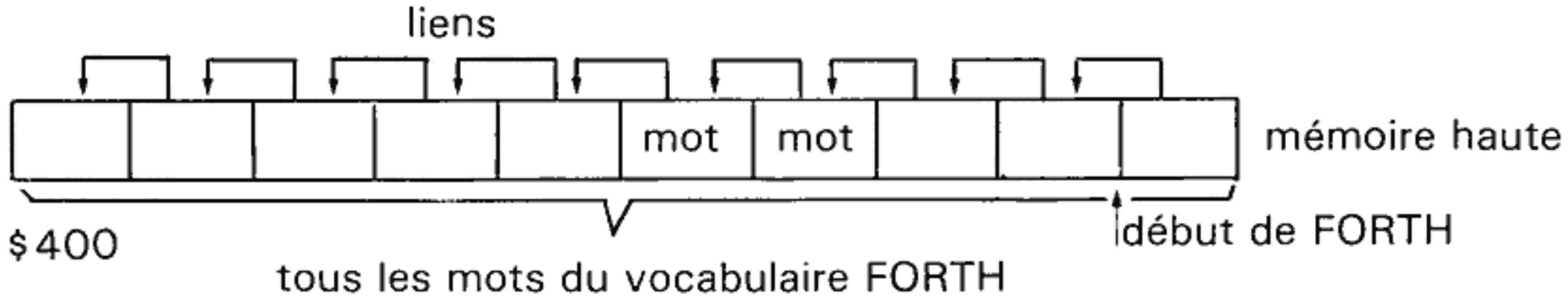
Mémoire haute.

CFA. Les autres mots auront d'autres adresses, qui dépendent de la « la classe » ou « type » du mot.
Le champ paramètre contient, dans cet exemple, une liste de CFA des mots qui constituent CUBE, et la première adresse est appelée PFA (Parameter Field Adress). Pour une définition commençant par deux-points, la liste se termine par l'adresse de ' ; ' qui a pour fonction la fin d'un sous-programme, un équivalent de RTS en assembleur. Le contenu du PFA varie en fonction du type du mot.

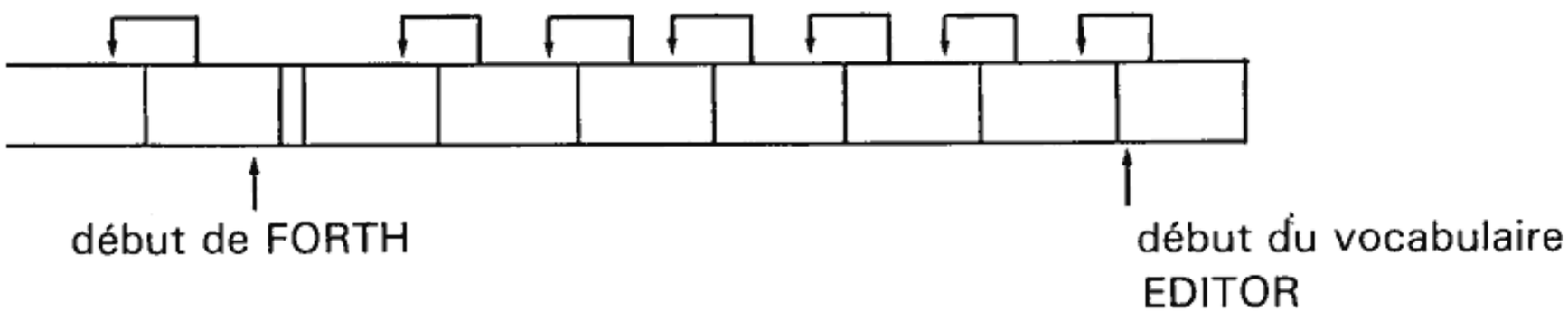
Vocabulaires

Les adresses des liens chaînent tous les mots du dictionnaire FORTH, et forment une longue liste, dans un premier lieu, qui est le VOCABULAIRE FORTH dans son entier. Pour plus de commodité, et pour une plus grande vitesse de recherche, vous pouvez séparer les nouveaux mots dans différents vocabulaires, un exemple est l'éditeur (EDITOR).
Les vocabulaires ont deux principaux effets : premièrement ils augmentent la vitesse de compilation, en permettant à la recherche de démarrer dans la bonne « zone ». Deuxièmement, vous pouvez avoir des mots qui ont le même nom dans différents vocabulaires sans risque de confusion (par exemple la commande R de l'éditeur et la commande R de FORTH).
Lorsque l'on prend un nouveau vocabulaire, l'adresse de chaînage est modifiée pour s'assurer que les nouveaux mots seront bien compilés dans ce vocabulaire.

Un exemple
1. Après avoir chargé FORTH, la structure du dictionnaire initiale est :



2. Après ajout de l'EDITOR, un deuxième vocabulaire existe :

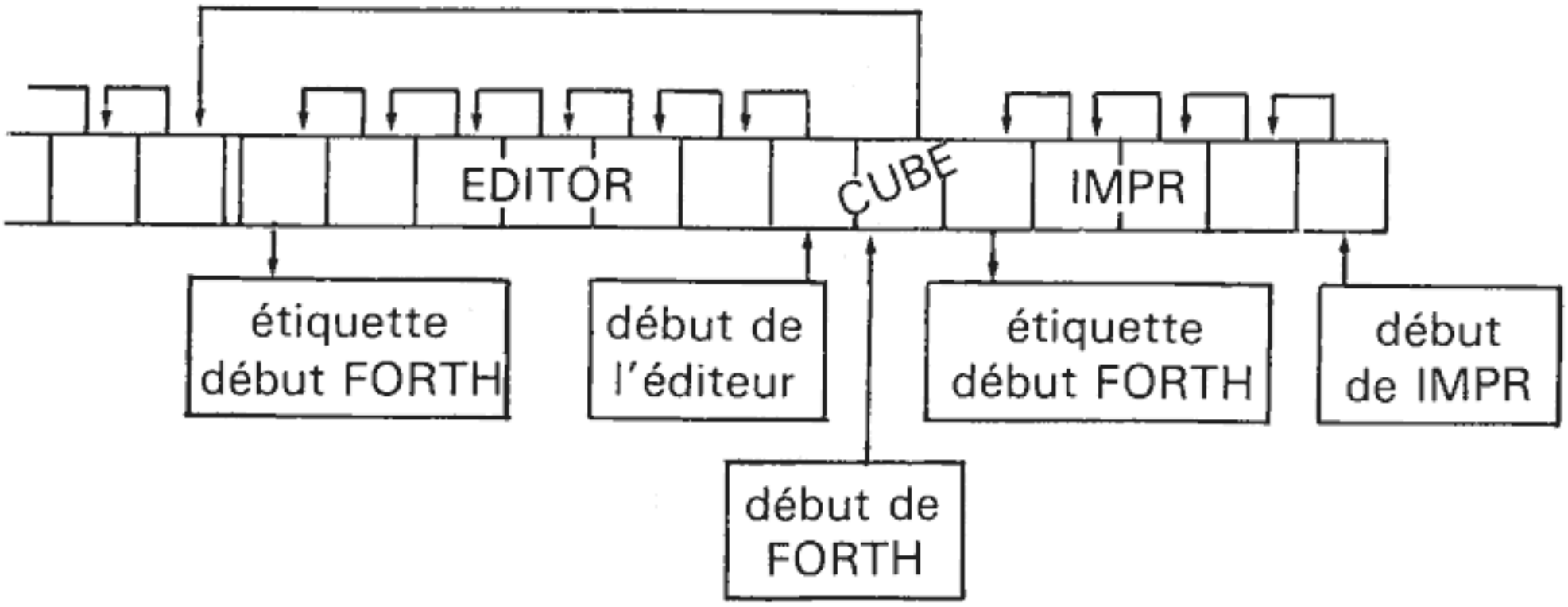


Lorsque vous tapez EDITOR (Return), cela dit à l'interpréteur que les recherches débutent au somment de vocabulaire de l'éditeur. En tapant FORTH (Return) on réinitialise le pointeur, appelé pointeur de contexte du vocabulaire, au somment de FORTH, l'éditeur est alors sauté.

3. Supposons que nous ajoutons CUBE au vocabulaire FORTH :

Remarquez que CUBE est ajouté à la fin du dictionnaire, mais est chaîné au dernier mot de FORTH, en sautant par-dessus l'éditeur.
Comme avant, la fin de EDITOR pointe au début de FORTH. La règle générale voulant que tous les vocabulaires pointent vers le vocabulaire principal FORTH, mais aucun ne pointe vers un autre vocabulaire.

4. Si nous ajoutons maintenant un nouveau vocabulaire appelé IMPR, nous aurons alors :



Nous avons donc trois vocabulaires : FORTH, EDITOR et IMPR. Par sécurité tous les nouveaux dictionnaires sont chaînés à FORTH, aucun n'est chaîné à un autre vocabulaire.

5. Pour se positionner dans un vocabulaire, l'instruction correcte est FORTH DEFINITIONS (mise de FORTH comme vocabulaire) VOCABULARY IMPR IMMEDIATE (déclare un nouveau vocabulaire appelé IMPR) puis IMPR DEFINITIONS positionne IMPR en tant que vocabulaire courant (c'est-à-dire que les nouveaux mots sont ajoutés à la liste du vocabulaire IMPR).
Enfin FORTH DEFINITIONS retourne au vocabulaire FORTH à la fin des additions au vocabulaire IMPR.
Pour utiliser les mots qui sont dans IMPR, tapez IMPR (Return).

Les autres 1 %

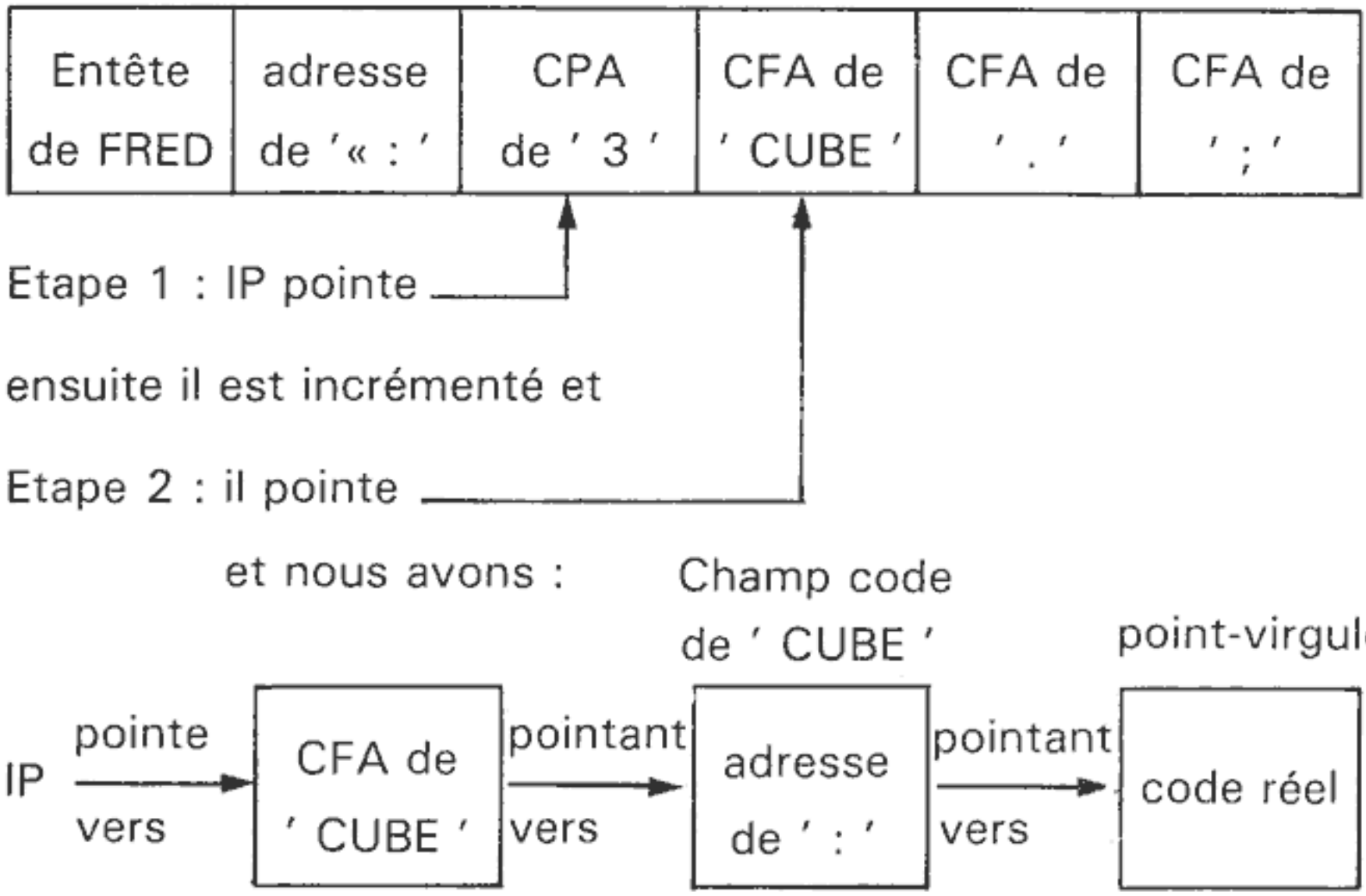
Nous avons dit que 99 % de FORTH dans le dictionnaire. Le pour cent restant (à part les piles) est la ZONE de TAMPON-CASSETTE et le BLOCK de VARIABLES UTILISATEUR.
Ils sont situés en haut de mémoire (regardez la carte de la mémoire).
Les variables utilisateur est un bloc de variables utilisées par le système, bien qu'elles soient accessibles aussi par vous. Elles sont appelées utilisateur parce qu'elles sont spécifiques à un utilisateur particulier. FORTH peut être multiutilisateur, et dans un tel cas, il y a un bloc de variables utilisateur pour chaque personne et seulement le pointeur d'utilisateur a besoin d'être modifié pour les référencer.
Leur contenu actuel, leur signification et leur emplacement sont donnés dans une annexe et dans le glossaire.
Le tampon cassette est une zone dans laquelle vous manipulez les informations de la cassette. C'est un tampon de 1 Koctets (1 028 octets exactement) qui est utilisé par le gérant de cassette. Ce tampon est composé ainsi :
2 octets : contiennent le numéro de bloc qui vient d'être stocké dans le tampon.
1 024 octets : zone de données, contenant un bloc disque.
2 octets : contenant zéro, pour marquer la fin du tampon.
Si vous voulez avoir une information sur le disque, les mots corrects en FORTH sont BLOCK, BUFFER, UPDATE et FLUSH. Regardez le glossaire.

CHAPITRE 7.

Le champ code

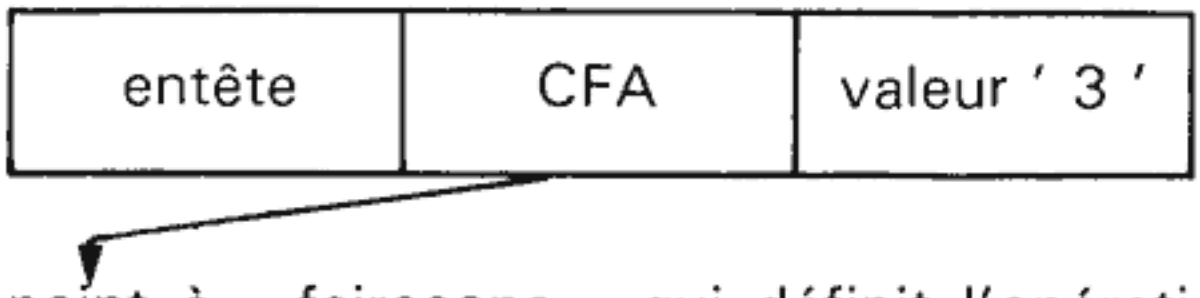
Dans chaque mot du dictionnaire FORTH, il y a un mot de deux octets qui s'appelle le champ code. Ce champ très important détermine le type du mot et comment il s'exécute.
Le contenu du CHAMP CODE est l'adresse d'un sous-programme en langage machine qui sera exécuté lorsque le mot sera appelé, c'est-à-dire que le CFA (Adresse du champ code) représente le début du mot.
Pour voir comment cela fonctionne, nous devons rentrer plus profondément dans le travail de FORTH.
Comme c'est décrit au chapitre 7, la plupart des mots FORTH sont constitués par une suite d'adresses d'autres mots FORTH à exécuter. Le compteur de programme Forth appelé IP (Interpretive Pointer) suit la liste élément après élément, travaillant d'une façon très similaire au compteur de programme du microprocesseur.

Jetons tout d'abord un regard sur les mots de type : c'est-à-dire les plus courants, avec notre exemple CUBE. Supposons que nous ayons un autre mot quelque part défini ainsi : FRED 3 CUBE. ; qui utilise le mot CUBE.
Lorsque ce mot est démarré, IP pointe sur le ' 3 ' qui est défini comme un mot FORTH qui met la valeur 3 au sommet de la pile. IP est alors incrémenté de deux et pointe sur le prochain élément de la liste, c'est-à-dire CUBE. Cet élément contient le CFA (Adresse du Champ Code) du mot CUBE. FORTH va maintenant effectuer un saut indirect. Cela signifie qu'il ne va pas à « CUBE » lui-même mais au code machine pointé par le champ code de « CUBE ». Cela semble très complexe, essayons de le dessiner :

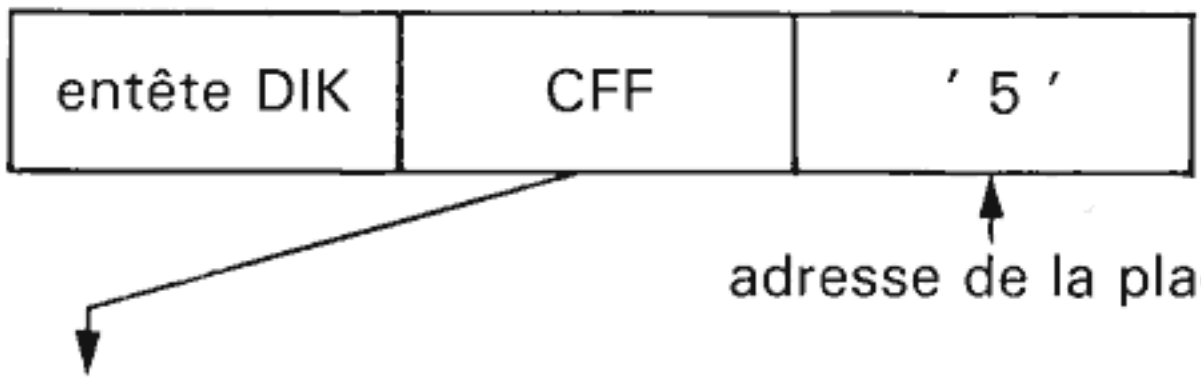


Il y a deux niveaux d'indirection dans FORTH, qui ont fait sa renommée (c'est-à-dire deux niveaux de pointeur pour accéder au code réel).
Le saut indirect (qui est une partie de l'interpréteur interne donne l'exécution directement à ' : '. Cette procédure travaille exactement comme un sous-programme, c'est-à-dire :
Afin de pouvoir exécuter le nouveau mot (CUBE dans ce cas), l'IP doit être modifié pour qu'il pointe sur la liste à l'intérieur de CUBE (son champ paramètre). Pour pouvoir faire cela nous devons d'abord sauvegarder l'ancien IP.
Et c'est ce que fait ' : '. Il sauvegarde tout d'abord IP en le poussant dans la pile des retours du microprocesseur, et il charge ensuite l'IP avec l'adresse de début de la liste de CUBE.
CUBE finit par un ;. C'est l'inverse de :, il va rechercher la valeur de l'IP dans la pile des retours et restaure le pointeur.
En résumé' : ' est une sorte de JSR Forth et ' ; ' l'équivalent de RTS.
' : ' est seulement approprié aux mots définis par le symbole :, ce qui veut seulement dire que le mot est constitué par une liste d'anciens mots. Par conséquent, si le champ code d'un mot contient l'adresse de ' : ' ce mot est « défini par ' : '.

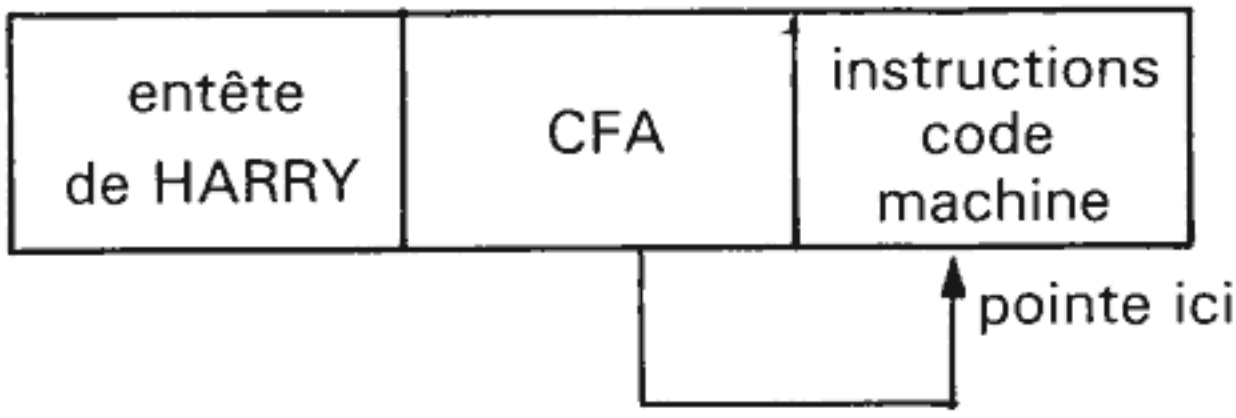
Autres types de mots.
Les autres mots utilisés ordinairement en FORTH sont CONSTANT, VARIABLE, USER ET CODE.
Exemple 1.
3 CONSTANT TILT définit le mot TILT qui a le type constante, et à pour valeur 3. L'élément du dictionnaire aura la forme :



point à « fairecons » qui définit l'opération de CONSTANT, c'est une procédure qui extrait la valeur 3 et la pousse dans la pile lorsque le mot TILT est exécuté.
Exemple 2.
5 VARIABLE DIK définit un type VARIABLE, avec la valeur initiale de la variable à 5.

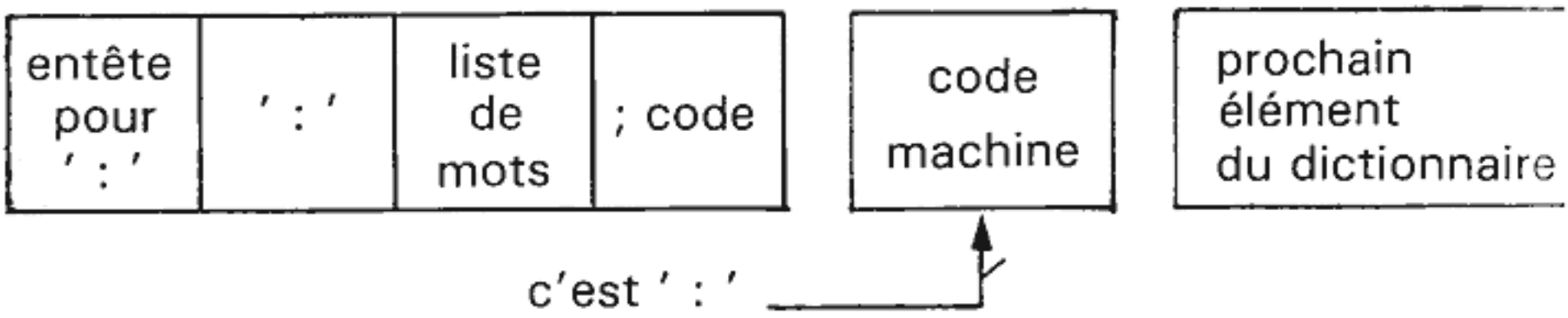


pointe vers ' fairevaria ' qui définit l'opération de VARIABLE qui consiste en ceci, lorsque DIK est exécuté, l'adresse de l'endroit où est stockée la variable est poussée au sommete de la pile.
Exemple 3.
26 USER TOM définit une variable utilisateur, de nom TOM. Les variables utilisateur sont stockées dans un bloc spécial situé en haut de la mémoire et qui sont définies ainsi : TOM sera le 26e élément dans le bloc utilisateur. Quand TOM sera exécuté, l'adresse de stockage sera poussée au sommet de la pile.
Exemple 4.
Avec un assembleur :
CODE HARRY **code**
définit une procédure en langage machine. Dans ce cas le mot contient directement le code à exécuter donc le champ code pointe à son début.



(voyez aussi le prochain chapitre).

La prochaine question est : où donc est-ce que les programmes ' : ', ' fairevaria ', etc. résident ?
La réponse est qu'ils forment une partie des « mots-définition ». Un « mot-définition » fait partie du groupe des ' : ', ' fairevaria ', etc. et est ainsi appelé parce qu'il définit un type particulier.
Les « mots-définition » peuvent être considérés comme des mots d'un type spécial, qui ont deux parties distinctes : l'une qui spécifie comment compiler un élément du type correct et la seconde partie qui définit comment il s'exécute.
Par exemple :



Lorsque ' : ' est exécuté, c'est-à-dire lorsque vous avez tapé ; FRED.... ; , la « liste de nom » sont les mots FORTH qui sont nécessaires pour créer (CREATE) une entête du dictionnaire dont le nom est FRED, et dont le champ code contient l'adresse de ' : ', le code machine qui suit le mot ; CODE.
Donc un mot-définition est constitué de :

— Une partie de « construction » (BUILD) qui fabrique l'entête correcte.

— Une partie d'exécution (DOING) qui constitue le code à exécuter pour ce type de mot.

Nous arrivons maintenant à la meilleure (ou la pire) partie.
L'un des principaux avantages de FORTH est sa capacité à créer de nouveaux « mots-définition ». C'est quelquefois très utile, et cette possibilité n'est disponible seulement dans un ou deux langages (desquels le BASIC n'est pas).

Deux sortes de mots-définition peuvent être générés, ceux dont la partie exécution (DOING) est en code machine, et ceux dont la partie DOING est écrite en FORTH. La première sorte offre une vitesse d'exécution plus grande, mais il est très important de savoir ce que vous faites. La seconde est plus facile à réaliser.
Nous nous limiterons à un seul exemple, la définition de CONSTANT :

```
: CONSTANT CREATE SMUDGE, ; CODE LDY #$2 (c'est
' fairecons ')
LDA (W),Y
PHA
INY
LDA (W),Y
JMP PUSH
```

Ainsi :
CONSTANT définit le nom de l'opération.
CREATE.SMUDGE génère l'entête du dictionnaire pour le nouveau mot. ;CODE est le mot qui met l'adresse du code machine dans le CFA du nouveau mot.
Cet échantillon de code machine (pour 6502) est celui qui prend la valeur contenue dans le champ paramètre de CONSTANT et le pousse dans la pile.
La deuxième sorte pour définir un mot-définition est d'utiliser le mot Forth DOING, ce qui est souvent utilisé pour créer d'autre type de données.
Exemple :
Voici une autre manière d'implémenter le mot CONSTANT :

```
: CONSTANT <BUILDS, DOES> @ ;
```

N'oubliez pas que CONSTANT est un mot standard.
< BUILDS signifie « tout ce qui suit jusqu'au mot DOES > est la partie construction (BUILDING) du mot-définition. » < BUILDS génère lui-même la partie entête du nom.
' , ' signifie prenez le mot au sommet de la pile et compilez-le dans la prochaine position du dictionnaire. Rappelez-vous que lorsque vous utilisez CONSTANT, la valeur est fournie au sommet de la pile, et y reste tant que le ' , ' ne l'utilise pas.
Ainsi <BUILDS génère l'entête du nouvel élément dans le dictionnaire et ' , ' met la valeur dans le champ paramètre.
DOES > signifie « tout ce qui suit est le programme FORTH qui doit être éxécuté lorsque le mot nouvellement construit sera

appelé. » DOES > met également le PFA dans la pile lorsque le nouveau mot sera exécuté. Comme dans ce cas notre constante est située dans le champ paramètre, @ mettra dans la pile la valeur stockée è cette adresse.
Ainsi si vous faites 120 CONSTANT MINE en utilisant la définition de CONSTANT, il sera construit une entête de nom MINE et la valeur sera chargée dans le champ paramètre.
Lorsque vous exécuterez ' MINE ', la partie DOES sera appelée pour prendre la valeur et la pousser dans la pile. Simple, n'est-ce pas !
En travaillant ainsi, nous pouvons créer des tableaux d'octets à une dimension :

```
: TABOCTETS <BUILDS ALLOT DOES > + ;
```

et lorsque vous utiliserez 23 TABOCTETS TRUC pour fabriquer TRUC, un tableau de 23 octets, numéroté de 0 à 22.

CHAPITRE 8.

Mots en langage machine.

Il se peut, afin d'améliorer la vitesse d'exécution, ou d'utiliser des sous-programmes en EPROM, que l'utilisation du code machine soit nécessaire.
La méthode correcte est d'utiliser l'assembleur FORTH structuré. Cependant il peut être très utile de voir comment le code machine peut être entré directement sans utiliser d'assembleur. Les instructions pour l'utilisation de l'assembleur qui est fourni dans votre cassette seront décrites plus tard dans ce manuel.
La méthode utilisée pour entrer du code machine en tant que mots ou octets est d'utiliser le mot FORTH, (virgule) et C, (C-virgule) qui permet de placer des octets dans le dictionnaire. Nous allons le voir sur un exemple. Pour créer le son ' ZAP ' : en assembleur le code à entrer est :

```
STX   XSAVE (Sauvegarde de X)
JSR   $F41B (ZAP)
LDX   XSAVE (Restitution de X)
JMP   NEXT
```

Où NEXT est un mot FORTH où TOUS les programmes machine doivent retourner lors de leur fin.
Nous allons tout d'abord calculer le code hexadecimal des instructions : nous avons :

Adresse	Code	Instruction
4000	86B5	STX XSAVE
4002	201BF4	JSR ZAP
4005	A6B5	LDX XSAVE
4007	4C4404	JMP NEXT

L'étape suivante consiste à grouper les codes en paire, dans l'ordre donné ci-dessus :
86B5 201B F4A6 B54C 4404

La troisième étape consiste à renverser les octets de chaque paire :
B586 1B20 A6F4 4CB5 0444
Nous avons maintenant un code machine compilable :
HEX
CREATE ZAP B586, 1B20, A6F4, 4CB5, 0444, SMUDGE
CREATE créée la tête du dictionnaire
SMUDGE termine la procédure
HEX est nécessaire parce que le code est un hexadécimal.
(Cette méthode de codehexa-codehexa est également valable avec ;CODE pour définir de nouveaux mots).
Afin de rendre plus sûre l'utilisation des procédures en langage machine, vous avez besoin d'en savoir plus sur ce que vous avez ou n'avez pas le droit de faire.
Au départ de la procédure en code machine.
A l'entrée du sous-programme en code machine, l'accumulateur et le registre Y du 6502 sont disponibles. Y est toujours remis à zéro.
Le registre X est le pointeur de pile FORTH et donc ne doit pas être utilisé, ou s'il doit l'être il doit être préalablement sauvegardé en le stockant à l'adresse XSAVE (qui est en hexa B5) et restauré à la fin du sous-programme.
Accès à la pile.
la pile FORTH fonctionne avec des nombres sur 16 bits. Le contenu du sommet courant de la pile est donne par 0,X pour les bits de poids faible et 1,X pour les bits de poids forts. L'élément suivant de la pile se trouve en 2,X et 3,X.'
Si vous voulez laisser de la place pour un nouvel élément dans la pile, vous devez faire DEX DEX. De même INX INX supprimera le premier élément de la pile en faisant pointer le pointeur de pile derrière le premier élément.
Page zéro.
Toutes les adresses en zéro-page comprises entre B6 et FF sont disponibles.
Branchements.
N'utilisez pas des instructions JMP sauf pour aller à la section NEXT. Si vous voulez effectuer un branchement inconditionnel, vous pouvez faire ceci :
CLC

BCC Adresse
ce qui n'utilise pas plus de mémoire et est relogeable.

Points de sorties.

Il existe toute une série de points de sortie, et tous retournent à NEXT en effectuant un certain nombre de fonctions. Ils doivent tous être utilisé avec l'instruction JMP.

NOM	ADRESSE	FONCTION
NEXT	0444	va à l'instruction FORTH suivante
POP	05EE	supprime le sommet de la pile, puis va à NEXT
POPTWO	05EC	supprime les deux premiers éléments de la pile et va à NEXT
PUSH	043D	crée un nouvel élément dans la pile, en prenant l'octet de poids fort dans l'accumulateur et celui de poids faible dans la pile des retours (où il est mis avant l'appel à PUSH)
PUSHØA	07DC	même chose que pour PUSH, sauf que l'octet de poids fort est automatiquement égal à Ø, et l'octet de poids faible est la valeur de l'accumulateur
PUT	043F	même chose que pour PUSH, sauf que le nouvel élément vient écraser l'ancien au sommet de la pile

Exemple.

Cet exemple est tiré de l'interpréteur FORTH, c'est le code qui permet de faire l'opération ' + ', qui additionne deux éléments de la pile :

```
18      CLC       ; Retenue = Ø
B500    LDA  Ø,X  ; Charge bit poids faible
7502    ADC  2,X  ; Addition bit poids faible
9502    STA  2,X  ; Stockage bit poids faible
B501    LDA  1,X  ; Charge bit poids fort
7503    ADC  3,X  ; Addition bit poids fort
9503    STA  3,X  ; Stockage bit poids fort
4CEE05  JMP  POP  ; Sortie en supprimant l'ancien élément
                  au
                  ; sommet de la pile
```

Ce qui donne :
18B5 0075 0295 02B5 0175 0395 034C EE05
et après échange des octets d'une même paire nous donne :

```
HEX
CREATE + B518, 7500, 9502, B502, 7501, 9503, 4C03,
05EE,
SMUDGE
DECIMAL.
```

ANNEXE A

Messages d'erreur

Remarque : si l'on met —1 dans la variable WARNING, la procédure ABORT ne cause aucune erreur, FORTH est relancé comme si l'on avait effectué un démarrage à chaud.
Dans la table suivante, ' XXXX ' indique le ou les mots en erreur.

1. XXXX ?

Ce message peut provenir :

— du compilateur ou de l'interpréteur s'il ne peut trouver le mot XXXX dans le dictionnaire, généralement à cause d'une faute de frappe.

— du programme de recherche de l'Editeur, lorsqu'il n'a pas pu trouver la chaîne demandée.

2. ?Empty Stack (Message N° 1)

Pile vide : on a essayé de retirer quelque chose de la pile des paramètres alors qu'elle était vide.

3. ? XXXX Isn't Unique (N° 4

Ce message sert à signaler que vous venez de compiler un mot qui a déjà été utilisé auparavant. La compilation est quand même réalisée.

4.? Disc Range (N° 6)

Vous avez utilisé un numéro d'écran qui est hors des limites autorisées.

5. ? Full stack (N° 7)

Pile pleine : s'explique de lui-même.

6. ? XXXX compilation only (N° 17)

Vous avez essayé d'exécuter XXXX directement depuis le clavier (en mode entrée), alors que ce mot est valide seulement à l'intérieur d'une définition, c'est-à-dire pendant la compilation du mot à mettre dans le dictionnaire (en mode compilation).

7.? XXXX Execution only (N° 18)
Analogue à la précédente. XXXX est seulement utilisable lors de l'exécution et ne peut être compilé.

8. ? XXXX Conditionals not Paired (N° 19)
Vous avez terminé une définition sans avoir correctement apparié l'un ou l'autre de ces mots

DO LOOP
DO +LOOP
IF ELSE ENDIF
BEGIN UNTIL
BEGIN AGAIN
BEGIN WHILE REPEAT

9. ? XXXX Definition not finished (N° 20)
Un autre message généré lors de l'emploi d'une définition incomplète.

10. ? XXXX in Protected Dictionary (N° 21)
Dans le dictionnaire protégé : vous avez essayé de supprimer (FORGET) une définition qui se trouve dans la zone protégée située devant l'adresse de la variable FENCE.

11 ?XXXX Loading Only (N° 22)
Analogue aux erreurs 6. et 7. pour les instructions qui ne peuvent être exécutées que lors d'un chargement à partir du disque.

12. ? XXXX OFF Current Screen (N° 23)
La position du curseur de l'éditeur est sortie du bloc courant de 1 Koctet. Le mot TOP réinitialise cette position en O.

13. ? Dictionary Full (N° 2)
Le mot affiché va utiliser plus de place dans le dictionnaire que ce qu'il reste : le dictionnaire est plein.

14. ? Disc Error (N° 8)
Une autre forme d'erreur de chargement.

Annexe B

Variables stockées dans la « zone utilisateur »

déplacement par rapport à la base de la zone utilisateur				chargé à partir de la table de chargement
HEX	DECIMAL	NOM	FONCTION	HEX
0	0	(NTØP)		40C
2	2	(BSCH)		40E
4	4	(UAREA)		410
6	6	(SØ)	param. initial. pointeur. pile	412
8	8	(RØ)	stockage initial pointeur pile des retours	414
A	10	TIB		416
C	12	WIDTH	WARM ↓ démarrage à chaud	418
E	14	WARNING		41A
10	16	FENCE		
12	18	DP	démarrage à	41C
14	20	VOC-LINK	COLD ↓ froid	41E
16	22	BLK		420
18	24	IN		
1A	26	OUT		
1C	28	SCR		
1E	30	OFFSET		
20	32	CONTEXT		
22	34	CURRENT		
24	36	STATE		
26	38	BASE		
28	40	DPL		
2A	42	FLO		
2C	44	CSP		
2E	46	R#		
30	48	HLD		
32	50	libre		
↓	↓			
7E	126			

WARM : lors d'un démarrage à chaud toutes les variables précédentes sont réinitialisées à partir de la table de chargement.
COLD : même chose lors d'un démarrage à froid.

Annexe C

SAUVEGARDE D'UNE APPLICATION

Lorsque vous avez développé une application, vous souhaiteriez sauvegarder la forme compilée pour former un fichier exécutable. Pour faire cela, quelques paramètres de chargement doivent être modifiés de telle façon qu'un démarrage à froid inclue votre application.
Un exemple de ce que vous avez besoin de faire est reproduit ici avec des commentaires et se trouve à la fin du code de l'éditeur.

```
FORTH DEFINITIONS DECIMAL
LATEST 12 +ORIGIN ! (pointeur chargement vers sommet du
dictionnaire)
HERE 28 +ORIGIN ! (limite au sommet du dictionnaire)
HERE 30 + ORIGIN ! (pointeur de dictionnaire au chargement)
HERE FENCE ! (limite courante)
```

Si vous avez déclaré un nouveau vocabulaire, vous devez également faire

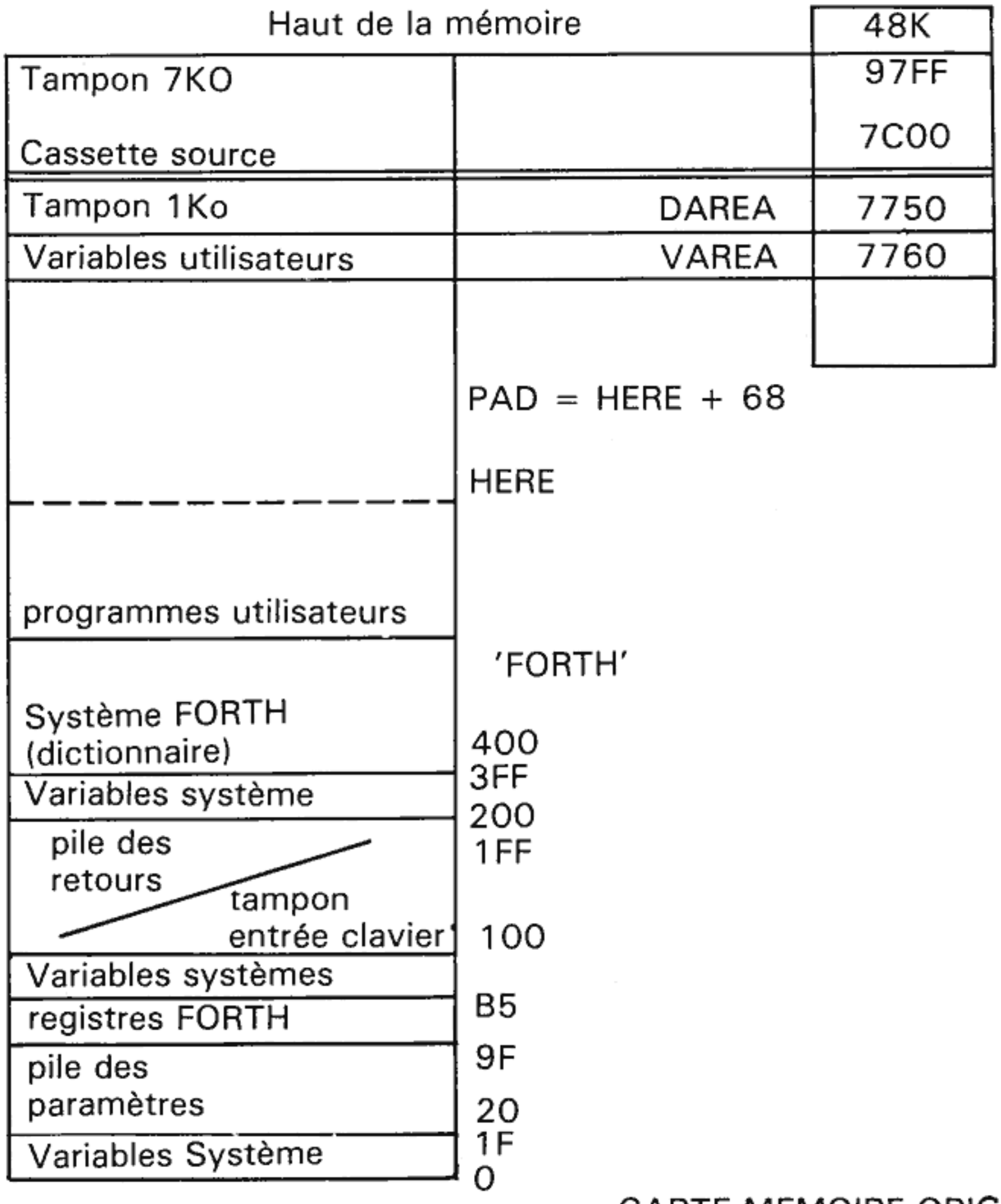
```
' XXXXX 6 + 32 +ORIGIN ! (chaînage vocabulaire initial)
```

Où XXXXX est le nom du plus récent mot du dictionnaire.
Après avoir mis tout cela à la fin de votre programme, compilez-le en FORTH, puis faites :

```
HEX HERE 1 - Ø D. (Return)
```

Et cela imprimera l'adresse haute de tout le contenu du Forth.
Charger et lancer ce nouveau fichier vous donnera le FORTH plus votre application, le tout en une seule fois.

Une chose qu'il est usuel de faire est de combiner le FORTH et l'EDITOR, pour sauvegarder l'éditeur à chaque fois.
FORTH-SAVE sauvegardera une nouvelle version de FORTH sur la cassette. Il vous demandera le nom du fichier et vous demandera d'appuyer sur la touche espace lorsque vous serez prêt. N'oubliez pas de positionner correctement la vitesse (SPEED) à Ø ou -1 pour rapide ou lent.
HEX 101 63 ! permettra un démarrage automatique.



CARTE MEMOIRE ORIC

Annexe D

CONTENU DE LA CASSETTE

Face 1 : FORTH
EDITOR écran 1-7
Face 2 : ASSEMBLER
écran 1-5
EXTENSIONS
écran 1-7
TUNESMITH
écran 1-4 (démonstration musicale)
Oric Forth sur la cassette.

Extensions au vocabulaire standard.
Entrée : sortie sur cassette.
SPEED variable utilisateur
pour la vitesse de transfert
Ø SPEED ! rapide
-1 SPEED ! lent

CLOAD n m ---- charge des blocs source de 1 Ko depuis le buffer n au buffer m

CSAVE n m ---- idem pour la sauvegarde
n doit être inférieur ou égal à m

Primitives cassettes
(STORE) n ---- exécute le code machine de sauvegarde du BASIC.

(RECALL) n ----	idem pour le chargement
SETUP n ---- n	donne les adresses des blocs depuis leur numéro
NAME n	donne un nom de fichier à une zone tampon
Autres extensions	
PAPER n ----	n = Ø à 7 pour changer la couleur du fond
INK n ----	idem pour la couleur des caractères
CAPS ----	bascule majuscule/minuscules
KLCK ----	bascule clic du clavier
FORTH-SAVE ----	sauvegarde du Forth sur cassette
TEXT ----	passe en mode text
HIRES ----	passe en mode graphique

Attention, en FORTH, le paramètre doit être placé avant l'instruction
Exemple:
4 (espace) INK < RTN >

Annexe E

ASSEMBLEUR FORTH POUR L'ORIC-1

Imaginez un macro-assembleur complet pour le 6502, avec des commandes de pseudo-structure, des symboles, des égalités, opérant en notation polonaise inversée, avec un code source de seulement 80 lignes de long.
Ne cherchez pas plus loin, le voici.

Caractéristiques :

1. macro instructions (IF, BEGIN, etc.) augmentable à volonté
2. chiffres dans n'importe quelle base (modifiable)
3. expressions utilisant un algorithme résidant
4. structures de contrôle d'imbrication
5. le symbole égal
6. des étiquettes (si c'est absolument nécessaire)
7. source code de l'assembleur dans un langage portable de haut niveau

Le processus d'assemblage.

L'assemblage d'un code machine consiste en l'interprétation du vocabulaire de l'assembleur.
L'interpréteur du FORTH essaye de reconnaître le texte introduit, dans le vocabulaire contexte, positionné à l'assembleur, et par défaut recherche dans le vocabulaire FORTH ordinaire.
Les mots de l'assembleur spécificieront des opérandes, des modes d'adressage, des codes opératoires, etc., qui seront placés à la fin de la définition du CODE. Le nouveau mot sera « décoloré » si aucune erreur n'est détectée.

Durant la procédure d'assemblage, chaque mot de l'assembleur sera exécuté lorsqu'il sera rencontré. Sa fonction à ce moment sera d'« assembler », c'est-à-dire de générer du code machine, ou une adresse. Plus tard, lors de l'exécution, le code machine ainsi assemblé sera exécuté.

Codes opératoires.

Tous les mnémoniques et les macros finissent par une virgule « , » dans l'assembleur. Sa signification est :

a) la virgule finit un groupe de mots assembleur qui correspondent à une ligne ordinaire de code d'assemblage.

b) l'opérateur FORTH virgule « , » compile un mot dans le dictionnaire aussi une virgule dans le code signifie qu'à cet endroit quelque chose doit être entré dans le dictionnaire.

c) l'utilisation de « , » permet de distinguer entre un mnémonique et les autres mots (c'est-à-dire entre le nombre hexadécimal ADC et le mnémonique ADC,).

Un exemple.

```
CODE RIEN
NOP,
NEXT JMP,
;C
```

CODE crée une en-tête de dictionnaire de nom RIEN, et met le vocabulaire context à l'assembleur. Cet en-tête sera directement un mot FORTH permettant d'exécuter le code machine lors de l'appel de ce nom.

NOP, compile le code machine du mnémonique NOP, c'est-à-dire $EA.

NEXT JMP, compile l'instruction saut à l'adresse de NEXT.

;C fait un certain nombre de tests, et « décolore » l'en-tête du dictionnaire, il ne compile rien.

Ensuite.

FORTH interprète votre définition qui doit retourner à l'adresse de l'interpréteur a appelée NEXT.

A la fin de votre définition de code, le contrôle doit donc être retourné à NEXT, ou à l'une des autres possibilités qui permettent de manipuler la pile avant de retourner à NEXT (voir le chapitre 9 pour leur liste).

Pour le 6502, on doit toujours retourner à NEXT par l'intermédiaire d'une instruction JMP (Sauf inconditionnel).

Sécurité.

Un nombre raisonnable de tests sont effectués sur votre texte assembleur. Mais ils ne peuvent pas être exhaustifs.

a) Tous les paramètres qui sont mis dans la pile pendant la définition doivent être retirés avant la fin.
b) Les modes d'adressages doivent être autorisés pour les codes utilisés.
Si une erreur d'assemblage est détectée, ; ne « décolorera » pas l'en-tête, ainsi il n'y a pas de danger que vous puissiez exécuter une définition erronée.
Attention à l'utilisation de Ø= et de Ø<qui n'ont pas la même signification dans l'assembleur et en FORTH.

Code opération du 6502.

Tous les codes standard sont disponibles, et sont divisés en quatre groupes :
— codes opératoires simples
— codes opératoires multimodes (2 groupes)
— codes opératoires de branchement.
Les codes opératoires multimodes nécessitent un péorande (dans la pile) et un mode d'adressage.
Si aucun mode d'adressage n'est donné, la valeur par défaut est l'adressage absolu. L'assembleur essaye d'utiliser la page zéro lorsque c'est possible et qu'il reste de la place disponible.

Les symboles des différents modes d'adressage sont :

.A	accumulateur	pas d'opérande
#	immédiat	un octet
,X	indexé par X	adresse absolue ou zéro-page
,Y	indexé par Y	adresse absolue ou zéro-page
X)	indirect indexé par X	adresse page zéro
)Y	indirect indexé par Y	adresse page zéro
()	indirect	adresse absolue

Exemples.

Voici quelques exemples d'instructions FORTH en assembleur et dans leur forme « normale ». Remarquez que les opérandes viennent en premier, puis le mode d'adressage (si besoin) et enfin le mnémonique. Tous les mots sont séparés par des espaces, comme cela est normal pour tous les textes FORTH.

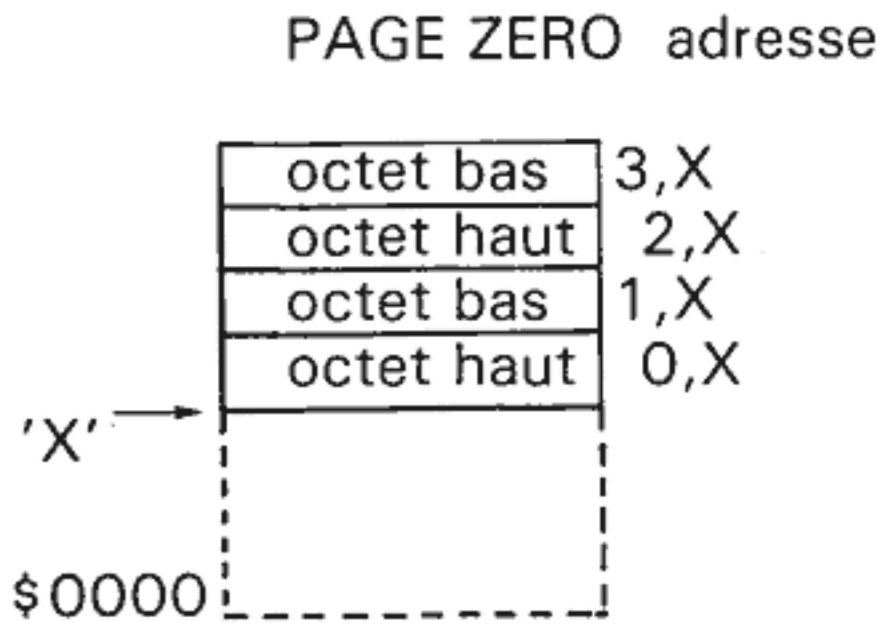
		FORTH	NORMALE	
	.A	ASL,	ASL	A
1	#	LDY,	LDY	# 1
TEMP	,X	STA,	STA	TEMPS, X
TEMP	,Y	CMP,	CMP	TEMP,Y
6	X)	ADC,	ADC	(6,X)
TABLE	)Y	LDA,	LDA	(TABLE),Y
POINT	()	JMP,	JMP	(Point)

Remarque : .A signifie l'accumulateur afin de pouvoir le distinguer du nombre hexadécimal A.

Accès à la pile.

La pile de paramètres est située dans la page zéro à partir de l'adresse $9E jusqu'à $20 ($ signifiant adresse hexadécimale).

Les éléments dans la pile sont stockés en tant qu'entités de 16 bits. Ils sont placés dans le mode normal du 6502, c'est-à-dire l'octet de poids faible à l'adresse de poids faible et l'octet de poids fort à l'adresse de poids fort. Cela permet d'utiliser le mode d'adressage (O,X) pour accéder à la mémoire lorsque le nombre contenu dans la pile est une adresse. X est le registre contenant le pointeur de pile, qui doit toujours indiquer le fond courant de la pile. Décrémenter deux fois le registre X revient à supprimer l'élément au sommet de la pile. Cela permet d'utiliser le mode n,X pour accéder à la pile et (n,X) pour utiliser la pile comme un pointeur.



Par exemple pour additionner les deux éléments au sommet de la pile (voir également le diagramme de la pile) :

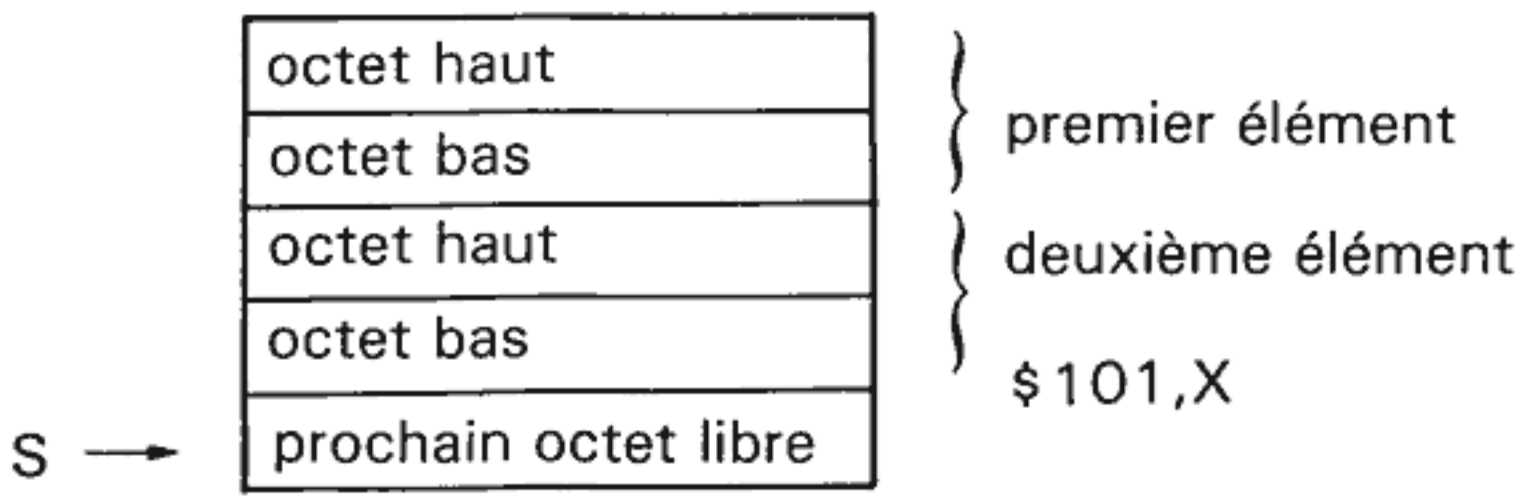
```
        CLC,
0   ,X  LDA,
2   ,X  ADC,    (addition des bits de poids faibles)
2   ,X  STA,    (et stockage)
1   ,X  LDA,
3   ,X  ADC,    (addition des bits de poids forts)
3   ,X  STA,    (et stockage)
POP     JMP,    (sortie en supprimant l'ancien sommet de la pile)
```

La pile des retours (en page 1) descend à partir de l'adresse **$1 FE** . Le registre du 6502 S pointe vers la prochaine place libre et les adresses sont placées dans la pile dans le même ordre que ci-dessus. Pour accéder à un octet arbitraire dans la pile des retours, le pointeur des retours courant doit être chargé dans le registre X, aussi X doit-il être sauvegardé avant.

```
soit : XSAVE  STX, (sauvegarde de X)
              TSX, (chargement de S)
```

A ce moment 101 ,X LDA (101 est en hexa) vous permettra d'avoir l'élément de la pile le plus récemment chargé dans la pile des retours. 102 ,X LDA vous donnera le suivant.

Pile des retours



Registres FORTH.

Il y a plusieurs « registres » FORTH. Pour le 6502, ce sont des endroits spéciaux en page zéro qui sont disponibles uniquement au niveau du code machine. Ils ont tous un nom qui peut être utilisé dans l'assembleur, qui retourne alors l'adresse appropriée.

IP = pointeur de l'interpréteur. C'est le compteur de programme de FORTH, et il pointe sur la prochaine adresse FORTH à être interprétée par NEXT.
W = adresse du pointeur vers le champ code de la définition du dictionnaire juste interprétée par NEXT. W-1 contient $6C, c'est-à-dire le code de saut indirect (JMP ()).
Un JMP W-1 effectuera un saut indirect au champ code qui pointe vers le code machine de la définition.
UP Pointeur utilisateur. Contient l'adresse de la base de la zone utilisateur.
N une zone brouillon de 9 octets commençant à N-1 jusqu'à N+7.
XSAVE tampon d'un octet permettant de sauvegarder le registre X.

Registres du microprocesseur.

Lorsque FORTH exécute NEXT et entre dans un code machine, les conditions suivantes sont réalisées :
1. le registre Y est mis à zéro et peut être utilisé librement.
2. le registre X pointe sur l'octet bas de l'élément du bas de la pile (relatif à $100).
3. le registre S pointe vers le prochain octet libre dans la pile des retours.
4. l'accumulateur peut être utilisé librement et ne contient aucune valeur particulière.
5. le processeur est en mode binaire, avec les interruptions autorisées, et s'attend à être remis dans cet état à la fin du code machine.

La zone de travail 'N'.

Lorsque des registres supplémentaires sont nécessaires, la zone 'N' peut être utilisée pour stocker des données ou des pointeurs, par exemple. Le mot assembleur N retourne une adresse en page zéro. Par convention, N-1 contient une valeur sur octet, et N, N+2, N+4, N+6 sont quatre paires d'octets permettant de stocker des valeurs de 16 bits. La procédure 'SETUP' permettra de transférer des valeurs dans la zone N. Il est important de noter que beaucoup de procédures FORTH utilisent cette zone 'N', aussi n'espérez jamais récupérer des données dans cette zone lorsque vous avez quitté une définition et que vous êtes rentrés dans une autre.

Exemple :

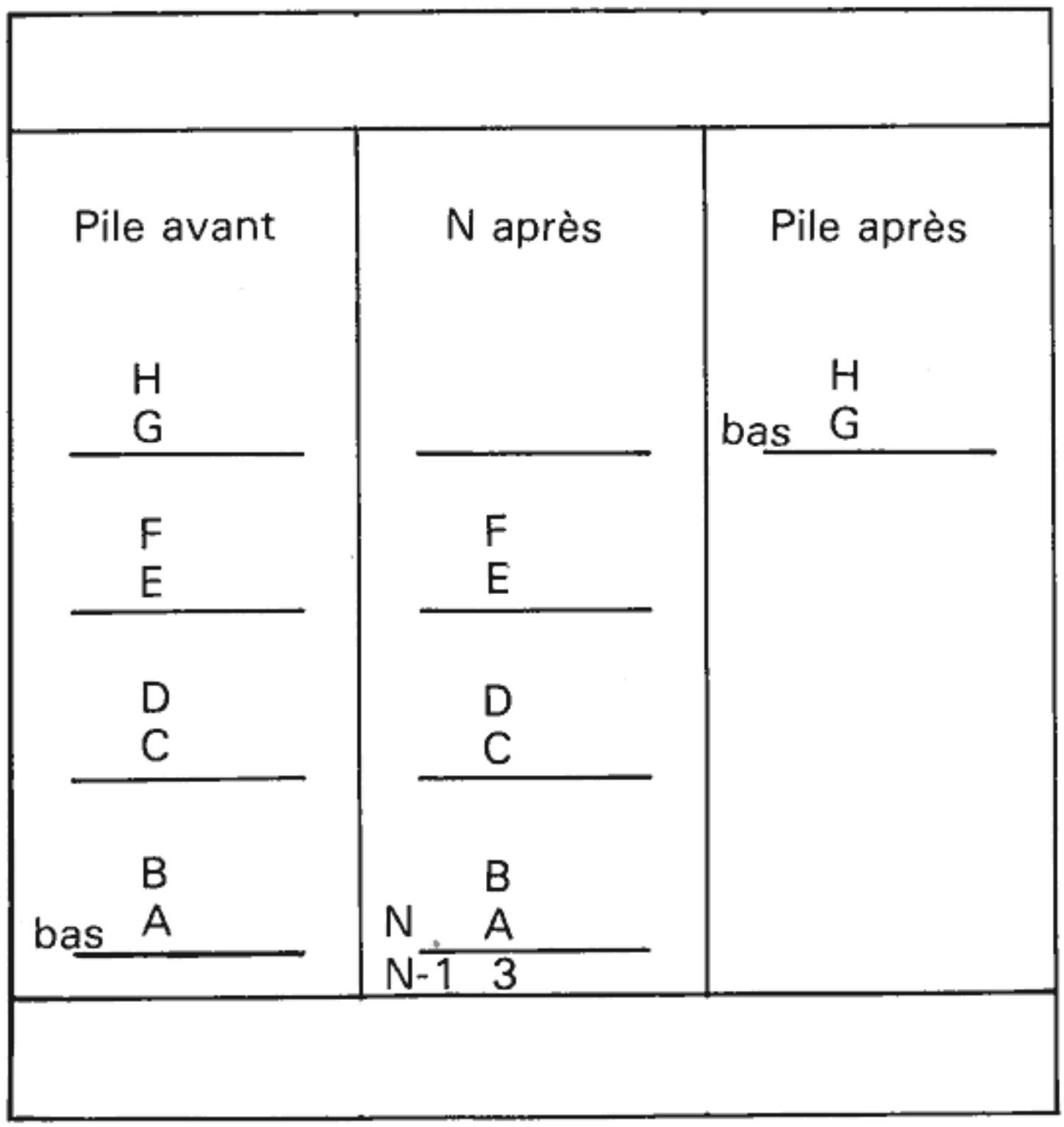
```
CODE TEST-PORT
6  # LDA, N 1 - STA      (mets un compteur dans N-1)
BEGIN, PORT BIT,         (teste une adresse)
```

```
N 1 - DEC,                    (décrémente le compteur)
0= UNTIL, NEXT JM,
;C                            (boucle jusqu'à ce que le compteur
                               soit égal à zéro)
```

'SETUP'.

Si vous voulez transférer des valeurs depuis la pile vers la zone N, la procédure 'SETUP' est celle qu'il vous faut. Lorsque vous effectuez SETUP, l'accumulateur contient le nombre d'élément de 16 bits à transférer, donc A peut seulement contenir : 1, 2, 3 ou 4.

c'est-à-dire : 3 # LDA., SETUP JSR,



Structures de contrôle et boucles.
Cet assembleur FORTH vous permet d'utiliser des instructions de branchement et des instructions de déclaration d'étiquettes. Mais vous pouvez aussi utiliser des structures de « pseudo-contrôle » (macroinstructions), approche qui est préférable, car le code obtenu sera bien mieux structuré.

Méthode 1.
Des étiquettes peuvent être déclarées par la pseudo-opération LA BEL, par exemple :

```
LA BEL TOTO Ø ≠ LDA, FRED BEQ,
```

C'est une boucle infinie. Le symbole TOTO est introduit dans une PETITE table de symbole qui contient à la fois son nom et son adresse courante. La prochaine utilisation de FRED mettra cette adresse dans la pile. Attention de ne pas saturer la table des symboles, sinon tout sera perdu !
Durant le « debugging », la table des symboles peut être vidée (entièrement ou bien partiellement) par KILL, XXX, cette instruction opère comme FORGET à l'intérieur de la table des symboles.

Méthodes 2.
Tous les mots FORTH de haut niveau permettant de contrôler l'exécution d'un programme sont disponibles (excepté DO....LOOP). Ce sont les mêmes mots qu'en FORTH, mais on leur a aiouté une virgule : IF, ELSE, ENDIF, BEGIN, UNTIL, AGAIN, WHILE, REPEAT, et ils sont utilisés de la même manière.
SAUF pour IF, WHILE, et UNTIL, les versions en langage de haut niveau testent une valeur vraie au sommet de la pile. Les versions en assembleur testent un bit dans le mot d'état du processeur. Vous devez spécifier quel est le bit que vous testez. Voici la liste des tests disponibles :

CS	bit retenue	C = 1
VS	bit overflow	V = 1
Ø<	bit négatif	N = 1
Ø=	bit zéro	Z = 1
CS NOT	bit retenue	C = Ø
VS NOT	bit overflow	V = Ø
Ø< NOT	bit négatif	N = Ø
0=	bit zéro	Z = Ø

Par exemple :
PORT LDA, Ø= IF, (aaa) ENDIF,
lis la variable PORT et exécute aaa si elle est égale à Ø (Z = 1).
PORT LDA, Ø= NOT IF, (aaa) ELSE, (bbb) ENDIF,
lis la variable PORT et exécute aaa si elle est différente de Ø et bb dans le cas contraire.
De même pour les boucles :
6 # LDY, BEGIN, PORT DEC, DEY, Ø= UNTIL,
qui décrémente PORT jusqu'à ce que Y atteigne la valeur 0.
6 # LDY, BEGIN, DEY, # = NOT WHILE, PORT DEC, REPEAT,
fait la même chose.

Remarque A.

ELSE, AGAIN, REPEAT, fabrique un branchement inconditionnel en compilant le code CLC, label BVC, (label = étiquette) aussi vous ne pouvez utiliser le bit V pour votre propre usage. Si besoin est vous pouvez réécrire la macro utilisée pour qu'elle fasse un JMP à la place.

Remarque B.

Les branchements donnant lieu à un dépassement de valeur ne sont pas testés et ne donnent lieu à aucune détection d'erreur.

GLOSSAIRE DE L'ASSEMBLEUR

Primitives.

$MB constante pour le stockage d'un octet mode pour l'instruction en cours d'assemblage.

$MM constante pour le stockage d'un masque dépendant du mode.

$GM	---- n	retourne la partie de $MB relative au mode d'adressage.
$GB	---- n	retourne la partie de $MB qui indique si l'instruction à compiler a une longueur de 1, 2 ou 3 octets.
$!M n n1	----	stocke n et n1 dans $MB et $MM
$AZ	----	convertit $MB du mode absolu en mode zéro-page.
$CA addr	---- addr t/f	retourne vrai si l'adresse est en page zéro.
$SD		met les valeurs par défaut dans $MB et $MM.

$CC nl Octets	----	compile le code final plus les 1 ou 2 octets supplémentaires s'ils existent à leur place finale.
$ME		sortie d'une erreur lors d'un mode d'adressage illégal.
$PT code	----	va chercher les arguments corrects, complète le code opératoire et appelle $CC.
$CMmode	----	teste si l'instruction courante et le mode d'adressage sont concordants, sort par $ME s'il y a une erreur.
$DC	----	convertit le mode adresse absolu en mode page zéro lorsque cela est possible.

Mots-définition.

$$ MM définit un mode d'adressage.

$IM définit un code implicite (sans argument) et compile son code opératoire.

$BR définit un code de type branchement, et effectue sa compilation avec un décalage calculé. Il n'y a pas de test sur la valeur de ce déplacement.

M1 définit un code de type 1 et effectue sa compilation.

M1 définit un code de type 2 et effectue sa compilation.

Constantes système.

EQU une forme de 'CONSTANT'

CS Ø = Ø<VS définis les codes de branchement comme des constantes.

XSAVE
N
IP
W
UP

} sont les adresses des registres FORTH

NEXT PUSH POP SETUP PUSHØA POPTWO PUT sont les points de sortie.

Structure de pseudo-contrôle.

NOT br-c ---- br-c inverse le code d'un branchement c'est-à-dire VS NOT est équivalent de VC

Ø= NOT est équivalent de Ø≠
IF, + condition c'est-à-dire Ø= IF, PHA,
ELSE *
ENFIF,
BEGIN,
UNTIL, + condition c'est-à-dire Ø< UNTIL,.....
AGAIN, *
WHILE, + condition c'est-à-dire CS WHILE, ...
REPEAT, *
ces pseudo-structures peuvent être utilisées pour former des boucles de la même manière que celles dans le langage FORTH de haut niveau.
* : ces branchements inconditionnels sont simulés par la combinaison CLV, BVC étiquette
LA BEL ajoute une étiquette à la table des symboles de l'assembleur.

soit : LA BEL TOM PHA, TOM BEQ,
KILL TOM supprime toutes les étiquettes à partir et y compris TOM dans la table des symboles.
Le message « SYM ERR » sera affiché si TOM n'existe pas dans la table.

Addition au vocabulaire FORTH.

CODE ---- définit le début d'une procédure d'assemblage de code machine, soit CODE TEST

;C ---- termine une procédure CODE ou une procédure ;CODE donc CODE ..... ;C forment une paire comme : ..... ;
L'assembleur patche la procédure ;CODE dans le FORTH pour pointer dans l'assembleur.

RESUME DU GLOSSAIRE

Les instructions dans le glossaire sont divisées en groupes fonctionnels, chacun d'entre eux sera brièvement décrit ci-dessous. Assurez-vous que vous êtes familiarisé avec cette première page qui vous montre l'agencement de chaque élément, ainsi que les divers symboles pour les types de paramètres.

1. Opérateurs arithmétiques simple précision
Le nom explique clairement le contenu, remarquez cependant que vous disposez de divers opérateurs de combinaison, ainsi que de trois opérateurs logiques (AND, OR, XOR).
2. Opérateurs double précision.
Ils travaillent sur des nombres de 32 bits.
3. Opérateurs précision mélangée.
C'est-à-dire que certain des paramètres ont 16 bits, d'autres 32.
4. Bases de numération.
Auto explicatif.
5. Opérateurs de comparaison.
Ces opérateurs retournent un booléen. Remarquez que la valeur Ø indique la valeur vraie, et 0 indique la valeur fausse. Souvent vrai = 1, mais ce n'est pas une règle générale.
6. Opérateurs sur la pile.
Ils vous permettent de manipuler les premiers éléments de la pile. Remarquez que R et I ont le même effet, mais que par con vention I est réservé pour l'utilisation dans une boucle DO ... LOOP pour indiquer qu'il retourne l'indice courant.
7. Opérateurs sur la mémoire.
Ils vous permettent de manipuler directement des emplacements mémoire.
8. Entrée/sortie sur le terminal.
C'est certainement la partie de FORTH la plus satisfaisante. Toutes les fonctions nécessaires sont présentes, mais elles sont effectuées par des mots séparés.

Remarquez que 'EXPECT' est un mot pour une entrée généralisée, et .« XXXXX » est un opérateur de sortie pour les nombres.
Les chaînes de caractères FORTH sont stockées avec un octet contenant la longueur de la chaîne au début de celle-ci (longueur max = 255). Pour imprimer une chaîne dont on connaît l'adresse, COUNT va mettre la longueur dans la pile et ajuste l'adresse pour que TYPE puisse l'utiliser. UPPER est dans le dictionnaire mais a été laissé « déconnecté ».

9. Formatage des entrées/sorties.

Ces commandes sont utilisées pour transformer les nombres en chaînes de caractères pour l'édition, mais avec plus de flexibilité (voir l'exemple au chapitre 3).

10. Entrées/sorties disque.

Titre relativement clair.

11. Impressions.

Ces commandes ont été vues dans la section ' disque '. Regardez aussi l'UTILITAIRE D'IMPRESSION.

12. Vocabulaires.

13. Structures de contrôle.

Les constructions de haut niveau disponibles.

14. Les mots-définition.

C'est la classe des mots qui vous permettent d'ajouter des mots dans le dictionnaire. ':' a déjà été décrit, VARIABLE, CONSTANT et USER permettent la définition de variables et de constantes.
CREATE est la primitive qui génère une en-tête de dictionnaire de nom donné. Il peut être utilisé directement.
Les mots-définitions peuvent être considérés comme des instructions de compilation. Il dit au compilateur FORTH comment compiler un type particulier de mot. L'une des grandes puissances de FORTH est de vous permettre d'ajouter des nouveaux mots-définition qui vous seront propres, en utilisant des combinaisons de<BUILDS DOES>,{: ;CODE} et {BUILDS ;CODE}.

15. Opérateurs sur le dictionnaire et directives de compilation.

Ce groupe de mots permet d'effectuer des opérations sur la structure du dictionnaire, et sur les divers champs d'un élément du dictionnaire.

16. Commandes systèmes.

Diverses procédures d'initialisation.

17. Primitives système.

Diverses primitives en code machine. Vous ne devez jamais les utiliser directement.

Frédéric BLANC

François NORMANT

VISA POUR ORIC

éditions SORACOM

RESUME
DES INSTRUCTIONS FORTH

Symboles arithmétiques :
n = entier signé sur 16 bits
u = entier non signé sur 16 bits
d = entier signé sur 32 bits
ud = entier non signé sur 32 bits
Autres symboles :
c = caractère ASCII, les 9 bits de poids fort à Ø
b = octet, les 8 bits de poids fort à Ø
t/f = booléen Ø faux (t pour true)
addr = addresse sur 16 bit
P signifie que l'instruction est immédiate, c'est-à-dire qu'elle sera toujours exécutée, même si FORTH est en mode compilation.

E signifie que cette instruction est disponible uniquement en mode exécution.
C signifie que cette instruction est disponible uniquement en mode compilation.
Toutes les instructions FORTH s'effectuent sur des nombres de 16 ou 32 bits stockés dans la pile des paramètres, où elles prennent leurs données et y mettent les résultats. Les octets ou les caractères sont gérés sur des mots de 16 bits, avec les bits inutiles à Ø.

Disposition du glossaire

NOM	ENTREES	PILE	SORTIE	DESCRIP-TION
c'est le nom de l'opéra-tion	ce sont les données nécessaires pour l'opération. A droite le haut de la pile		résultats qui sont retournés dans la pile. A droite le haut de la pile	
+	n1 n2	----	n3	addition

L'instruction notée + nécessite deux données qui sont détruites pendant l'opération. Elle retourne une valeur, la somme signée des deux entrées.

Opérateurs arithmétiques simple précision.

+	n1 n2 ---- n3	n3 = n1 + n2	addition
–	n1 n2 ---- n3	n3 = n1–n2	soustraction
*	n1 n2 ---- n3	n3 = n1 * n2	multiplication
/	n1 n2 ----n3	n3 = n1/n2	division, résultat tronqué
*/	n1 n2 n3 ---- n4	n4 = n1 *n2/n3	produit intermédaire 31 bits.
/MOD	n1 n2 ---- rest quo	reste et quotient de n1/n2	
MOD	n1 n2 ---- rest	reste de n1/n2 modulo	
*/MOD	n1 n2 n3 ---- n4 (r) n5 (q)	reste et quotient de */	
MINUS	n1 ----–n1	changement de signe	
MAX	n1 n2 ---- n3	n3 est le plus grand des deux	
MIN	n1 n2 ---- n3	n3 est le plus petit des deux	

ABS	n1	---- u	laisse la valeur absolue
+–	n1 n2	---- n3	applique le signe de n2 à n1 et laisse le résultat
S>D	n	---- d	extension du signe
1+	n1	---- n1 + 1	incrémentation
2+	n1	---- n1 + 2	ajout de 2
AND	n1 n2	---- n3	et logique
OR	n1 n2	---- n3	ou logique
XOR	n1 n2	---- n3	ou exclusif logique

Opérateurs double précision.

D+	d1 d2	---- d3	d3 = d1 + d2 addition
D+–	d1 n1	---- d2	comme +– sur double entier
DABS	d	---- ud	valeur absolue
DMINUS	d	----–d	changement de signe

Opérateurs précision mélangée.

U/	ud1 u1	---- u2 (r) u3 (q) division non signée, reste et quotient
U*	u1 u2	---- ud3 multiplication non signée
M/	d1 n1	---- n2 (r) n3 (q) division signée, reste et quotient
M*	n1 n2	---- d1 multiplication signée
M/MOD	ud1 u2	---- u3 (r) ud4 (q) division non signée, reste et quotient

Bases de numération.

HEX	base à 16 : hexadécimal
DECIMAL	base à 10 : décimal.

Opérateurs de comparaison.

ø<	n	---- t/f	vrai si négatif
ø=	n	---- t/f	vrai si non ø, renverse une valeur booléenne
>	n1 n2	---- t/f	vrai si n1>n2
<	n1 n2	---- t/f	vrai si n1<n2
U<	u1 u2	---- t/f	vrai si u1<u2, non signé
=	n1 n2	---- t/f	vrai si n1 = n2

Opérateurs sur la pile

ROT	n1 n2 n3	---- n2 n3 n1	met le troisième au sommet
DUP	n1	---- n1 n1	duplique le sommet
SWAP	n1 n2	---- n2 n1	échange les deux au sommet
DROP	n1	----	supprime le sommet
OVER	n1 n2	---- n1 n2 n1	recopie le second au sommet
PICK	n1	---- n2	va chercher n2 qui est la n1[e] entrée de la pile
-DUP	n1	---- n1	si n1 égal à Ø
	n1	---- n1 n1	si n1 différent de Ø alors duplication
R		---- n	copie sommet pile retour dans pile paramètres
I		---- Cn	même chose que R
>R	n	----C	met le haut de la pile dans la pile retours
R >		---- n	met le haut de la pile retours dans la pile
SP @		---- addr	laisse l'adresse du sommet de la pile avant l'ajout de l'adresse
!CSP			utilisé par le compilateur seulement pour sauvegarder la position de la pile dans CSP

Opérateurs sur la mémoire.

CMOVE	deb nb	----	transfert de nb octets depuis deb
?	addr	----	imprime sur le terminal le contenu de addr en tant que entier signé
BLANKS	addr nb	----	remplit la mémoire avec nb blancs depuis addr
ERASE	addr nb	----	idem avec nb Ø
FILL	addr nb b	----	remplit la mémoire avec nb octets remplis par b depuis l'adresse addr

C!	b addr	----	stocke l'octet à l'adresse addr
!	n addr	----	stocke le mot à l'adresse addr
C @	addr	---- b	recherche un octet à l'adresse addr
@	addr	---- n	recherche un mot à l'adresse addr
+!	n addr	----	ajoute n au contenu de l'adresse addr

Entrée/sortie sur le terminal.

	n	----	imprime n en tant qu'entier simple avec un blanc
R	n1 n2	----	imprime n1 justifié à droite dans un champ de largeur n2
D.	d	---	imprime un entier double
D.R	d n	----	imprime le double entier d justifié à droite dans un champ de largeur n
SPACES	n	----	imprime n espaces sur le terminal
WORD	c	----	lit la chaîne entrée jusqu'au caractère délimiteur (c est le caractère délimiteur).
			transfère la chaîne de caractère qui suit HERE jusqu'au premier délimiteur. Le premier octet de la chaîne compactée est la longueur.
QUERY		----	entrée de 80 caractères (ou jusqu'au (return) dans TIB. A la sortie de QUERY, IN vaut Ø
EXPECT	addr n	----	lit les caractères issus du terminal en les sotckant à l'adresse ' addr '

			jusqu'à ce que (return) ou n caractères soient reçus. Un caractère nul (Ø) est stocké à la fin de la chaîne.
.''		----P	imprime la chaîne terminée par .'', exécuté immédiatement en dehors d'une définition
COUNT	addr1	---- addr2	pour une chaîne à l'adresse addr1 dont le premier octet est la longueur, retourne la longueur n et le début. Utilisé avant TYPE pour sortir la longueur
SPACE		----	imprime un espace
CR		----	imprime un retour-chariot
?TERMINAL		---- t/f	teste pour un « break » du terminal (Ctrl C), retourne une valeur vraie si le Break est reçu, fausse sinon. ?TERMINAL n'attend pas l'appui de la touche, il regarde dans le tampon d'entrée pour voir le dernier caractère frappé
KEY		---- c	retourne le dernier caractère reçu du clavier, KEY attend qu'une touche soit pressée
EMIT	c	----	envoie c au terminal
TYPE	addr n	----	transmet une chaîne de n caractères débutant à l'adresse addr au terminal
MESSAGE	n	----	imprime le message d'erreur n au terminal si la variable ' WARNING ' = Ø n sera seulement imprimé

.LINE	lin scr	----	imprime vers le terminal la ligne lin de l'écran scr
(		----	introduction de commentaires terminés par) un espace est obligatoire après (
ID.	addr	----	imprime le nom de la définition dont le NFA se trouve dans la pile
DIGIT	c n1	---- n2 t	convertit le caractère c en binaire n2 en utilisant la base n1
	c n1	---- f	si conversion invalide

Formatage des entrées/sorties.

# S	d1	---- d2	convertit d1 en ASCII dans le buffer de sortie, en faisant des appels répétés à #, jusqu'à ce que d2 soit nul. Utilisés entre < # et # >.
#	d1	---- d2	la conversion procède à partir du chiffre le moins significatif de d1. A partir de d1, on génère le prochain caractère ASCII à mettre dans la chaîne de sortie. d2 est le quotient après division par ' BASE '.
SIGN	n d	---- d	si n est négatif, met le code ASCII pour ' — ' dans le tampon de sortie. Utilisé entre < # et # >.
<#			démarre la conversion numérique, la chaîne résultat se trouve dans PAD, exemple : < # # S SIGN # >
#>	d	---- addr n	termine la conversion numérique en laissant

NUMBER	addr	---- d	les paramètres nécessaires à TYPE convertit la chaîne ASCII à addr en un entier signé en utilisant la base courante (sur 32 bits), le premier octet de la chaîne doit être sa longueur. Si un point décimal est rencontré, sa position est retournée dans DPL. Si DPL est -1 aucun point décimal n'a été rencontré. Cela permet d'identifier des nombres de 32 bits en mettant un ' . ' en entrée dans un nombre
HOLD	c	----	insère un caractère numérique dans la chaîne de sortie. A utiliser seulement entre <· # et # >
PAD		---- adr	retourne l'adresse de début du tampon temporaire de texte
–TRAILING	adr1 n1	----adr2 n2	ajuste la longueur de la chaîne n1 pour supprimer les blancs superflus.

Entrée/sortie disque.

LOAD	n	----	commence l'interprétation de l'écran n. l'opération se termine à la fin de l'écran ou à la rencontre de ;S
-->		----	continue l'interprétation à l'écran suivant
BLOCK	n	---- addr	laisse l'adresse en mémoire du tampon contenant le bloc n du

			disque. Le bloc sera chargé à partir du disque s'il n'était pas résident. BLOCK contient l'ajustement de la variable OFFSET
BUFFER	n	---- addr	cherche le prochain tampon pouvant être utilisé et lui assigne le bloc n. S'il contenait quelque chose, il y a recopie sur le disque. addr est l'adresse de la première cellule de donnée (1er mot de 2 octets). BUFFER n'ajuste pas la variable OFFSET
DRØ DR1 DR2 DR3			commandes dépendant de l'installation de FORTH permettant de présélectionner la variable OFFSET sélectionnant les lecteurs de disques
EMPTY-BUFFERS			vide tous les tampons. Les tampons modifiés ne sont pas recopiés sur le disque. Utilisé comme procédure d'initialisation
FLUSH			force tous les tampons modifiés à être réécrits sur le disque
UPDATE			marque le tampon précédemment utilisé comme ayant été modifié (tampon pointé par PREV)

+BUF	addr	---- dr2 t/f	avance jusqu'au prochain tampon d'adresse adr2. t/f est faux si adr2 pointe sur le même tampon que PREV
.LINE	lin écr	----	primitive de la gestion des disques permettant de transférer un bloc de numéro écr à l'adresse addr vers (flg=0) ou depuis (flg=1) le disque. remarquez que 'scr' est un block FORTH de 1 Koctets.
PREV		---- addr	variable système contenant l'adresse du tampon disque utilisé le plus récemment
USE		---- addr	variable système contenant l'adresse du prochain tampon pouvant être utilisé (c'est-à-dire le moins récemment utilisé)
B/SCR		---- n	constante qui contient le nombre de tampons disque par 1024 octets d'écran
B/BUF		---- n	constante contenant le nombre d'octets par tampon (aussi la taille d'un secteur disque)
LIMIT		---- n	constante contenant l'adresse du sommet de la zone des tampons disque + 1
FIRST		---- n	constante laissant l'adresse du premier tampon disque

Impressions.

TRIAD	n	----	affiche les trois écrans contenant l'écran n commençant par un écran de numéro divisible par trois
LIST	n	----	affiche l'écran n. La variable SCR contenant
INDEX	déb fin	----	n imprime la première ligne de chaque écran depuis l'écran déb jusqu'à l'écran fin. Utilisé pour voir les lignes commentaires.

Vocabulaires.

VLIST	liste toutes les définitions contenues dans le vocabulaire CONTEXT, c'est-à-dire le vocabulaire qui est recherché d'abord, c'est-à-dire le vocabulaire d'EXECUTION
DEFINITIONS	utilisé sous la forme CCCC DEFINITIONS pour positionner le pointeur CURRENT comme étant égal au pointeur CONTEXT. En faisant CCCC on positionne CONTEXT sur le vocabulaire CCCC et en faisant DEFINITIONS on force le pointeur CURRENT à être égal au pointeur CONTEXT. CURRENT pointe sur le vocabulaire auquel les nouveaux mots seront

FORTH	---- P	ajoutés. positionne le CONTEXT sur le vocabulaire FORTH
VOCABU- LARY		utilisé sous la forme VOCABULARY CCCC pour créer la définition du vocabulaire CCCC. l'utilisation ultérieure de CCCC fera pointer le pointeur CONTEXT sur ce vocabulaire. La définition de VOCABULARY peut être déclarée IMMEDIATE
CURRENT	---- addr	variable utilisateur laissant l'adresse du premier élément du vocabulaire courant
CONTEXT	---- addr	variable utilisateur laissant l'adresse du début du vocabulaire de contexte (recherché en premier).

Structures de contrôle.

IF	t/f	----PC	
ELSE		----PC	

ENDIF		----PC	les trois mots forment la structure d'une exécution conditionnelle :
			IF (partie vraie)... ENDIF
			IF (partie vraie) ELSE (partie fausse) ENDIF la sélection entre la partie fausse ou vraie est effectuée à partir du booléen se trouvant au sommet de la pile

DO	n1 n2	----PC	

LOOP		----PC	en programme remarquez que les paramètres de la boucle DO... LOOP sont stockés au sommet de la pile des retours pendant l'exécution de la boucle.
+LOOP	n1	----PC	Les trois éléments ci-dessus forment une boucle où le test pour la sortie ou la continuation est effectué à la fin de la boucle DO... LOOP ' DO ' demande au sommet de la pile n2 la valeur initiale de l'index et n1 la limite de la boucle. ' LOOP ' incrémente l'indice par 1 et si le résultat est strictement inférieur à la limite n1 la boucle se poursuit en revenant à son départ. DO... n1 +LOOP travaille comme ci-dessus, mais la valeur signée n1 située au sommet de la pile est ajoutée à l'indice de boucle. n1 doit être présent dans la pile à chaque fois avant +LOOP dans la boucle. Le branchement en arrière dans la boucle a lieu jusqu'à ce que le nouvel index soit supérieur ou égal à la limite (si n1 est positif) ou jusqu'à ce que le nouvel indice soit inférieur ou égal à la limite si n1 est négatif.

LEAVE		----C	force la fin d'une boucle DO... LOOP en mettant la limite égale à l'indice courant.
I		----C n	utilisé à l'intérieur d'une boucle DO... LOOP pour copier l'indice de boucle au sommet de la pile des paramètres (voir aussi R)

Les 6 instructions suivantes fournissent une forme plus générale de boucle.

BEGIN		----P	marque le début de la boucle
WHILE	t/f	----PC	point de sortie conditionnel. Une valeur fausse permet de sortir de la boucle
REPEAT		----PC	terminaison de boucle après WHILE
UNTIL	t/f	----PC	terminaison de boucle qui permet de sortir si la valeur est vraie, sinon continue la boucle.
AGAIN		----PC	terminaison de boucle qui retourne inconditionnellement au BEGIN
BACK	addr	----	utilisé uniquement par le compilateur. Compile un branchement en arrière.

Les boucles suivantes peuvent être construites :

BEGIN	... UNTIL	la boucle est exécutée tant que la valeur booléenne au sommet de la pile lors de l'exécution de UNTIL est vraie, c'est-à-dire une boucle jusqu'à vrai.

BEGIN...	WHILE... REPEAT	la valeur booléenne est testée par WHILE et la sortie de la boucle s'effectue quand la valeur est fausse, c'est dire une boucle tant que la valeur est vraie.
BEGIN... AGAIN		boucle inconditionnelle.

Mots-définition.

:	----PE	débute une définition deux-points d'une nouvelle procédure créant une nouvelle en-tête de dictionnaire et se mettant en mode compilation.
;	----PC	termine une définition deux-points d'un mot de « haut niveau » et termine le mode compilation.
;CODE	----PC	compile la procédure ;S dans le dictionnaire. termine une définition deux-points d'un mot défini en code machine qui se trouve derrière la séquence :
	FRED...	; compile un élément du dictionnaire de nom FRED dont la CFA pointe sur une procédure qui effectue la fonction de sous-programme et laisse l'adresse du premier du champ paramètre dans le pointeur de l'interpréteur (IP).

la séquence :		JIM...	;CODE compile un nouveau mot dont le nom est JIM. Quand JIM sera exécuté sous la forme : JIM JACK, un élément du dictionnaire nommé JACK sera créé dont le CFA pointera sur la procédure en code machine définie en JIM

REMARQUE ;CODE nécessite un ASSEMBLER pour compiler la partie en code machine.

CONSTANT	n	----	utilisé dans la forme n CONSTANT MARIE pour compiler dans le dictionnaire un élément dont le nom est MARIE et dont le PFA contient la valeur constante n. Le CFA pointe vers une procédure qui pousse la valeur de la constante au sommet de la pile lorsque MARIE est exécuté.
VARIABLE	n	----E	utilisé sous la forme n VARIABLE PAUL pour compiler dans le dictionnaire un élément dont le nom est PAUL et dont le PFA contient la variable n. Le CFA pointe vers une procédure qui met le PFA au sommet de la pile lorsque PAUL est exécuté.
USER	n	----	utilisé pour créer un élément dans la zone utilisateur. n USER FLOB

		génère un élément dans le dictionnaire de nom FLOB. Son PFA contient la valeur n qui est le déplacement dans la zone utilisateur de la variable. Lorsque FLOB est exécuté, il met la valeur de l'adresse en mémoire de cet élément.
VOCABULARY	----E	utilisé en VOCABULARY GROT IMMEDIATE pour créer un vocabulaire de nom GROT.
<BUILDS DOES >	----C	utilisé à l'intérieur d'une définition deux-points : S P I T <BUILDS...DOES>... ; pour générer un mot de nom SPIT qui construit un élément du dictionnaire selon la partie qui suit <BUILDS et dont la procédure d'exécution sont les instructions de haut niveau qui suivent DOES jusqu'au point-virgule. Cela est mieux expliqué avec un exemple et une série d'essais et d'erreurs. Remarquez que lorsque le nouveau mot est exécuté, DOES> laisse le PFA au sommet de la pile.
CREATE	----	c'est la primitive qui crée un en tête dans le dictionnaire au sommet de celui-ci. L'en-tête est

« colorée » et le CFA pointe sur le PFA (c'est-à-dire que l'adresse suivant le CFA) il peut être utilisé sous la forme suivante

CREATE
CC

pour générer l'en tête CC. A utiliser avec précaution.

Opérateurs sur le dictionnaire et directives de compilation.

COMPILE

lorsque l'élément contenant le mot COMPILE est exécuté, le CFA du mot suivant est compilé dans le prochain endroit disponible.

COM-
PILE

utilisé à l'intérieur d'une définition dans la forme :

: XXXX [COMPILE] YYYY ; force YYYY à être compilé au lieu d'être exécuté immédiatement
immédiat : suspend la compilation à l'intérieur d'une définition pour permettre à un calcul de s'effectuer.
reprend la compilation, c'est-à-dire que la forme : : XXXX [des mots] d'autres mots ; les mots compris entre [et]sont exécutés et non compilés. Cela permet d'utiliser des paramètres dans la compilation.

IMMEDIATE			marque la plus récente définition contenue dans le dictionnaire comme IMMEDIATE, c'est-à-dire que le bit de précédence est mis à 1 dans l'entête.
TOGGLE	addr b	----	complémente le contenu de addr par le masque b. TOGGLE est utilisé par SMUDGE
SMUDGE			utilisé à l'intérieur d'une définition pour « colorier » le bit de « coloriage » dans le champ nom de la définition utilisé automatiquement par ' : ' et ' ; '. Un en-tête colorié ne peut être trouvé lors d'une recherche dans le dictionnaire.
LATEST		---- addr	donne le PFA du dernier mot du dictionnaire CURRENT.
PFA	nfa	---- pfa	convertit le NFA en PFA du mot
LFA	pfa	----lfa	convertit le PFA en LFA
CFA	pfa	----cfa	convertit le PFA en CFA
NFA	pfa	----nfa	convertit le PFA en NFA
		P addr	utilisé sous la forme ' FRED pour laisser dans la pile le PFA de FRED. Si le mode est compilation le pfa est compilé en tant que litéral.
TRAVERSE	addr1 n	----addr2	passe à travers d'un en-tête de nom dans la direction n (n = 1 en direction des adresses hautes, n = 1 en direction des adresses basses)

ID.	nfa	----	imprime le nom de la définition à partir du NFA
-FIND		----pfa b t	si trouvé : vrai et b est le nombre d'octets du nom.
		---- f	si pas trouvé : faux.
			-FIND demande une chaîne de caractères délimitée par des blancs et essaye de trouver une correspondance dans le dictionnaire.
ALLOT	n	----	réserve dans le dictionnaire de l'espace pour n octets en additionnant n au pointeur du dictionnaire.
,	n	----	stocke n dans la prochaine cellule disponible du dictionnaire et incrémente le pointeur du dictionnaire. (virgule)
C,	b	----	même chose que pour, (virgule) sauf qu'un seul octet est stocké.
LITERAL	n	----PC	compile n en tant qu'une valeur de 16 bits.
DLITERAL	d	----P	compile une valeur de 32 bits.
			Lors de l'exécution LITERAL et DLITERAL pousseront la valeur dans la pile.
HERE		----addr	retourne l'adresse de la prochaine cellule disponible dans le dictionnaire pointée par la variable DP
;S			arrête l'interprétation d'un écran disque.

→			;S est aussi la procédure compilée par ; continue l'interprétation au prochain écran.
Commandes système.			
MON			sortie vers le BASIC. A utiliser avec précaution.
FORGET			utilisé sous la forme FORGET FRED, supprime la définition FRED et toutes les définitions la suivant dans le dictionnaire.
+ORIGIN	n	---- addr	pour le déplacement n par rapport à l'origine actuelle de la mémoire.
RP!			
SP!			initialise le pointeur de la pile des retours.
			initialise le pointeur de la pile des paramètres.
QUIT			annule la pile des retours, arrête la compilation et retourne à l'entrée de données au clavier.
ABORT			annule toutes les piles, retourne au contrôle du clavier avec le message de bienvenue.
COLD			procédure de démarrage à froid, initialise les piles, le pointeur des variables utilisateur et redémarre par l'intermédiaire de ABORT
EXECUTE	addr	----	exécute la définition dont le CFA est dans la pile, l'interpréteur de
INTERPRET			texte qui compile ou exécute les chaînes

entrées soit au clavier, soit à partir du disque selon la valeur de STATE.

Primitives système.

(LINE)	n1 n2	---- addr nbre	convertit la ligne n1 de l'écran n2 en l'adresse du buffer et le nombre de caractères (max 64).
(ABORT)			procédure de redémarrage. Est également exécutée après une erreur si WARNING est égal à −1
(NUMBER)	d1 addr1	----d2 addr2	convertit une chaîne de caractères ASCII à l'adresse addr1 + 1 et le nombre d1, laissant le résultat d2. addr2 est le premier caractère inconvertible.
(.'')			la procédure compilée par .''
(FIND)	addr1 addr2	----pfa b	vrai si trouvé
		---- f	faux si non trouvé. (FIND) recherche dans le dictionnaire à partir du NFA à l'adresse addr2 correspondant avec le texte à l'adresse addr1.
(DO)			procédure compilée par DO
(LOOP)			de même pour LOOP. Le prochain mot compilé est un déplacement pour un branchement.
(+LOOP)			de même pour +LOOP
ØBRANCH			procédure pour le branchement conditionnel,

suivi par un déplacement pour un branchement.

BRANCH — idem pour les branchements inconditionnels.

CLIT — procédure permettant de mettre un caractère au sommet de la pile.

LIT — de même pour des mots sur 2 octets.

ENCLOSE addr1 c ---- addr1 n1 n2 n3 — primitive de recherche de texte utilisée par WORD. Elle parcourt la chaîne en recherchant le délimitateur c en partant de l'adresse addr1. Les occurrences initiales de 'c' sont ignorées. Lorsqu'un caractère non délimitateur est trouvé, la prochaine occurrence de 'c' est recherchée et la pile contient alors le déplacement n1 par rapport à addr1 du premier caractère non délimitateur, le déplacement n2 par rapport à addr1 du premier délimitateur après la chaîne, et n3 le déplacement par rapport à addr1 du premier caractère non inclus dans la chaîne. Remarquez que si le caractère NULL (0) est rencontré, il est considéré comme un délimitateur inconditionnel.

(;CODE) — procédure compilée par ;CODE

Procédures d'erreur.

ERROR	n	----in blk	exécute l'erreur n et redémarre le système en laissant in et blk si WARNING = 0 le message numéro n est affiché. si WARNING = 1 le message est pris dans les écrans 4 et 5 du disque 0. si WARNING = -1, le système ABORT (redémarre).
?ERROR	t/f n	----	imprime l'erreur n si la valeur est vraie.
?STACK			imprime un message d'erreur si la pile est hors limites.
?			idem si non exécution
?COMP			idem si non compilation
?PAIRS	n1 n2	----	émet l'erreur si n1 est non égal à n2, est utilisé pour les conditions appariées.
?CSP			émet une erreur si la position de la pile n'est pas la même que celle stockée dans CSP.
?LOADING			émet une erreur si le chargement s'est mal effectué.

Constantes.

B/SCR	retourne le nombre de secteurs disque (blocs) par écran d'édition.
B/BUF	retourne le nombre d'octets dans un secteur disque.
LIMIT	retourne l'adresse mémoire + 1 du tampon disque.
FIRST	retourne l'adresse mémoire du premier tampon disque.

C/L	nombre de caractères par lignes (64 en décimal).
BL	code ASCII pour l'espace
3	trois
2	deux
1	un
0	zéro, cette définition en constante permet une plus grande rapidité.

Variables.

USE	donne l'adresse du prochain tampon disque à utiliser.
PREV	donne l'adresse du plus récemment utilisé des tampons des disques.

Variables utilisateur.

TIB	donne l'adresse du tampon du terminal utilisé.
WIDTH	contient la longueur maximum admise pour un nom. (maximum 31 en décimal)
WARNING	contrôle des messages disques.
FENCE	adresse limite au FORGET, c'est-à-dire qu'au delà de cette limite, il est impossible de supprimer des définitions.
DP	contient l'adresse de la prochaine cellule (groupe de 2 octets) disponible dans le dictionnaire.
VOC-LINK	contient l'adresse du champ dans le plus récent dictionnaire.
BLK	contient le bloc courant. 0 signifie que l'entrée se fait à partir dé TIB.
IN	pointeur de déplacement par rapport au texte d'entrée. Utilisé par WORD.
OUT	nombre de caractères en sortie du terminal. Utilisé pour le formatage. Il est remis à zéro lorsque CR ou FF est émis vers le terminal.
SCR	contient le dernier numéro d'écran utilisé par LIST
OFFSET	pointe vers le disque suivant, consiste en un déplacement

CONTEXT	pointe au sommet du vocabulaire de context.
CURRENT	pointe au sommet du vocabulaire courant.
STATE	état de compilation, non zéro lors de la compilation.
BASE	base de numération courante.
DPL	après l'introduction d'un nombre converti, contient le nombre de chiffre à droite du point décimal ; si aucun point décimal n'est trouvé DPL =-1.
FLD	pour le contrôle de la largeur du champ de sortie des nombres.
	non implémenté dans le FIG-FORTH
CSP	utilisé par le compilateur pour stocker la position de la pile.
R#	utilisé par l'éditeur comme déplacement courant du curseur.
HLD	contient l'adresse du dernier caractère lors d'une conversion d'un nombre en sortie.

Commandes de l'éditeur.

EDITOR	appel de l'éditeur.
n LIST	liste d'écran n et le sélectionne pour l'édition
n CLEAR	détruit le contenu de l'écran n et le sélectionne pour l'édition.

Introduction de texte.

P	n	----	met le texte qui suit à la ligne n : 3 P CE TEXTE VA EN LIGNE 3 (Return)
NEW	n	----	sélectionne et affiche la ligne n pour l'édition (Return) termine et sélectionne la ligne suivante. Les lignes existantes sont écrasées par les lignes introduites.

UNDER	n	----	sélectionne la ligne n + 1 pour l'insertion de texte. Les lignes originales n + 1 jusqu'à la fin de l'écran sont descendues d'un cran. La ligne 15 est perdue.

Control du curseur.

TOP			place le curseur de l'éditeur au début de l'écran.
M	n	----	déplace le curseur du déplacement signé n. La nouvelle ligne où se trouve le curseur est éditée.

Manipulation de lignes.

H	n	----	met la ligne n dans le tampon PAD
D	n	----	supprime la ligne n en la sauvant dans le PAD. Les lignes n+1 sont montées d'un cran, la ligne 15 est remplie par des blancs.
T	n	----	tape la ligne n et la sauve dans le PAD
R	n	----	remplace la ligne n avec le contenu de PAD
I	n	----	insère le texte contenu dans le PAD à la ligne n, les autres lignes sont descendues et la ligne 15 perdue.
E	n	----	remplit la ligne n par des blancs.

S	n	----	Les lignes n à 14 sont descendues d'un cran, laissant la ligne remplie de blancs.

Commandes de chaînes.

F	texte	(Return)	recherche à partir de la position actuelle du curseur de la chaîne « texte ». Le curseur est laissé à la fin de la chaîne et la ligne contenant le curseur est affichée. Une erreur ' NOT FOUND ' est donnée si la chaîne n'est pas trouvée.
B			déplace le curseur en arrière d'une longueur égale à celle de la chaîne utilisée lors de la recherche. Utilisé pour positionner le curseur au début de la chaîne recherchée.
X	texte	(Return)	recherche et supprime la chaîne « texte ».
N			utilisé après F pour avoir la prochaine occurrence de la même chaîne.
C	texte	(Return)	copie le « texte » dans la ligne courante à la position courante du curseur.
TILL	texte	(Return)	supprime la chaîne commençant à la position du curseur jusqu'à et y compris la chaîne « texte ».

Remarque : taper C sans texte copiera un caractère NULL (0) dans la ligne, ce qui causera plus tard une erreur lors de la compilation.

Manipulation d'écran.

LIST			voir le glossaire FORTH
CLEAR	n	----	remplit l'écran de n blancs.
COPY	n1 n2	----	copie l'écran n1 dans l'écran n2
L			reliste l'écran courant ainsi que la ligne courante.
FLUSH			utilisé à la fin d'une modification pour s'assurer que tous les tampons modifiés sont bien stockés sur disque.

Primitives de l'éditeur.

TEXT	c	----	met le texte suivant dans PAD, 'c' étant le caractère délimiteur.
LINE	n	---- addr	convertit la ligne numéro n de l'écran courant en l'adresse du tampon contenant cette ligne.
WHERE	n1 n2	----	n2 = numéro du bloc, n1 déplacement dans le bloc. Si une erreur arrive durant le chargement du disque, ERROR laisse n1 et n2 dans la pile pour permettre de localilser la faute. WHERE imprime une « image » de la faute.
#LOCATE		---- n1 n2	convertit la position courante du curseur en numéro de ligne n2 et son décalage par rapport au début de la ligne n1
#LEAD		---- addr n	laisse l'adresse de la ligne du curseur et le

			déplacement n par rapport au début.
# LAG		---- addr n	laisse l'adresse du curseur et le nombre de caractères jusqu'à la fin de la ligne.
-MOVE	addr n	----	transfère une ligne de texte depuis l'adresse addr jusqu'à la ligne n de l'écran courant.
- TEXT	addr1 addr2 n2 ---- t/f		compare deux chaînes, la longueur de la plus courte étant n, les chaînes commencant aux adresses addr1 et addr2, le booléen est vrai si la chaîne est trouvée, sinon faux.
MATCH	addr1 n1 addr2 n2 ---- t/f		n3 addr1 = adresse du curseur, n1 nombre de caractères jusqu'à la fin de la ligne, addr2 est l'adresse de la chaîne, n2 le nombre d'octets de la chaîne. MATCH recherche à partir de addr1 une chaîne correspondant à la chaîne d'adresse addr2 et de longueur n2. Il retourne la valeur vraie si la chaîne est trouvée, et n3 est la nouvelle position du curseur. Si la chaîne n'est pas trouvée, la valeur est fausse, et n3 est le

			nombre d'octets jusqu'à la fin de la ligne.
1LINE		---- t/f	parcourt la ligne courante pour voir si la chaîne dans le PAD correspond. Un booléen est retourné, le curseur mis à jour.
FIND		----	recherche dans la chaîne depuis le curseur d'une chaîne correspondant à celle dans le PAD. S'il n'y a pas de correspondance, une erreur est émise et le curseur revient en haut du texte à son début.
BS		----	émet un caractère « backspace », retour en arrière.
NULL?			
ENTER			
ENTER ?			
2DROP			suppression d'un entier double.
2DUP			duplique un entier double.
2SWAP			échange deux entiers doubles.

Mots optionnels sur la cassette

Options de l'écran 1 :

PTC	yx	----	met le curseur de l'écran sur la ligne Y, à la position X dans la ligne.
GTC		---- y x	retourne la position courante du curseur. Met le curseur au début de ligne.
CLINE	y	----	remplit d'espaces la ligne y.

IN #		---- n	introduction d'un nombre à partir du clavier.
PON		----	toutes les sorties vont sur l'imprimante.
POFF		----	déconnecte les sorties vers l'imprimante.
PICK	n	----	prend la n^{e} valeur dans la pile et le met au sommet.

Options de l'écran 2 :

VRND			variable contenant le dernier nombre aléatoire.
PRND			primitive pour le calcul qui est nécessaire.
RND		---- n	un entier aléatoire est retourné, n est toujours positif.
SRND	n2	---- n	retourne un entier aléatoire n qui est compris entre 0 et n2.
RANDOMISE	n	----	initialise le générateur avec n.

Options de l'écran 3 :

1ARRAY			fabrique un tableau à une dimension, c'est-à-dire que 3 1ARRAY FRED construit un tableau de trois entiers de deux octets (l'indice allant de 0 à 2). La dimension est également stockée. L'exécution de 2 FRED retourne l'adresse de l'élément 2.
1CARRAY			comme ci-dessus mais avec des octets et non des mots de deux octets.

Il y a également une autre version de 1ARRAY utilisable lors du debugging qui vérifie l'indice avant chaque exécution, et sort un message d'erreur si l'indice est hors limite.

Option de l'écran 4 :
Cet écran contient 5 mots qui permettent d'avoir un ordre similaire à celui du PASCAL : l'ordre CASE (Selon). L'instruction CASE est similaire à un branchement calculé, permettant de 1 à N choix possibles dépendant de la valeur d'entrée, qui est dans cette version une valeur entière.
Il y a également une option qui teste si la valeur correspond au contenu du corps du programme et qui exécute une procédure de défaut.
Les cinq mots de l'écran sont :
(OF) primitive en langage machine compilée par OF
CASE le mot débutant l'instruction CASE
OF teste la valeur entrée avec la valeur clé pour trouver l'égalité
ENDOF fin d'une instruction OF
ENDCASE fin de l'instruction CASE
Exemple d'utilisation :
: FRED CASE (débute la définition de FRED et l'instruction CASE)
23 OF..... ENDOF
12 OF..... ENDOF
134 OF..... ENDOF
DEFAULT CODE
ENDCASE
L'instruction CASE attend une valeur dans la pile qui est comparée avec chacun des nombres donnés avant le mot OF. Si la correspondance est obtenue, l'argument est supprimé et le code compris entre OF et ENDOF est exécuté, suivi par un saut à la fin de ENDCASE qui permet de sortir de l'instruction CASE. Si aucune correspondance n'est obtenue, le 'DEFAULT CODE' est exécuté, ceci étant optionnel.
Remarquez que ENDCASE exécute toujours un DROP en premier. Cela vous permet de mettre ce que vous voulez dans le DEFAULT CODE.

Cet ordre CASE fait ses tests en temps réel, et peut être imbriqué de OF .... ENDOF. Voici un exemple :
A l'entrée de CASE N ---- n (n est la clé)
après une recherche
avec clé trouvée ----- après exécution de OF
Si pas trouvée ---- n est supprimé à la fin du ENDCASE
Donc la clé d'entrée est toujours perdue, aussi vous devez la dupliquer (DUP) si vous voulez l'utiliser plus tard.

Options de l'écran 5 :

PING		----	idem en BASIC
SHOOT		----	idem en BASIC
EXPLODE		----	idem en BASIC
ZAP		----	idem en BASIC
SOUND	c p v	----	idem en BASIC avec c = voie,p = période,v = volume
MUSIC	c o n v	----	idem en BASIC o = octave N = note
PLAY	t b m p	----	idem en BASIC t = voie pour le son, b = voie pour le bruit, m = mode enveloppe.

Options de l'écran 6 :

(CURSET)	primitives en code machine, à ne pas utiliser directement.
(CURMOV)	
(DRAW)	
(CIRCLE)	
(PATTN)	
(CHAR)	
(POINT)	
(FILL)	

Options de l'écran 7 :

CURSET	x y fb	----	idem en BASIC avec x position en X, y position en Y, fb fond/devant
CURMOV	xr yr fb	----	idem en BASIC yr position relative en Y, xr position relative en X

DRAW	xr yr fb	----	idem en BASIC
CIRCLE	r fb	----	idem en BASIC r rayon
PATTERN	n	----	idem en BASIC
CHAR	x s fb	----	idem en BASIC x valeur ASCII, s jeu de caractères
POINT	xr yr	----l	idem en BASIC l valeur logique on/off
FILL	b a n	----	idem en BASIC b ligne,a cellule,n valeur.

Remarque : les commandes dont les valeurs dépassent les limites ne sont pas exécutées. Pour voir si une telle erreur s'est produite lisez en $2EØ (en hexa) Ø : correct 1 : erreur.

Maquette SORACOM
JOUVE Mayenne
N° 12130 - N° éditeur 018
Dépôt légal : 4e trimestre 1983

ISBN 2 904032 10 X