FORTH

pour

THOMSON MO5

Langage de Programmation

Schiex T. & Bandini JC.

PRÉFACE

Tout d'abord, nous vous remercions pour l'acquisition de ce système Forth pour MO5. Ce programme va doter votre ordinateur d'un langage différent qui va vous faire découvrir de nouveaux domaines.

Basic ou Forth ?

Le Basic et le Forth sont deux langages radicalement différents. Il convient donc lorsque l'on étudie le Forth d'oublier les habitudes du Basic. Le Forth est un langage plus proche du fonctionnement réel de la machine, il utilise des concepts informatiques avancés que l'on ne trouve pas avec le Basic: piles, vocabulaire extensif, compilation, manipulation plus directe de la mémoire etc.

Le Forth n'offre pas un environnement de programmation aussi protégé que celui du Basic. En contrepartie il utilise mieux les ressources de la machine et donc autorise des vitesses d'exécution supérieures à celles du Basic (les rapports de vitesse peuvent atteindre 25). Ceci permet l'écriture de programmes qui auraient nécessité l'utilisation de l'assembleur. De plus le Forth est un langage extensif, c'est à dire que l'utilisateur peut intégrer de nouveaux mots au langage. Il faut aussi remarquer que le Forth de part sa nature impose une programmation structurée .

Enfin, cette version pour MO5 offre des possiblitées graphiques importantes. Il est possible de créer des sprites. Cette caractéristique à laquelle s'ajoute la rapidité du Forth permet la réalisation de jeux vidéo. Un exemple est fourni avec la cassette.

Utilisation du manuel:

Nous vous conseillons de lire ce manuel en essayant au fur et à mesure les fonctions que vous découvrirez. Il est difficile d'assimiler le Forth en une lecture, n'hésitez donc pas à passer rapidement certains paragraphes qui s'éclairciront lorsque vous aurez une vue d'ensemble du langage. Si vous connaissez déjà le Forth la table des matières et l'index vous permettront de trouver rapidement les renseignements dont vous avez besoin.

Pour charger le système, il faut sè placer en début de cassette et taper RUN"; le chargement et le démarrage sont automatiques.

Nous vous souhaitons maintenant une bonne lecture et surtout une bonne programmation.

Comment charger votre Loriciels

1. Rembobinez la bande
2. Tapez RUN " "
3. Pressez la touche [ENTREE]
4. Appuyez sur la touche [PLAY] ou [LECTURE] du Lecteur.

TABLE DES MATIÈRES

1.0 PRINCIPES GÉNÉRAUX DU FORTH

1.1 La mémoire

Une bonne compréhension de la structure de la mémoire du MO5 est indispensable pour dominer complètement le Forth. Ceux d'entre vous qui sont familiarisés avec ces concepts peuvent se permettre de passer à la partie qui suit.

Avant tout, rappellons que la donnée élémentaire que peut manipuler un ordinateur (et le MO5 n'échappe pas à cette règle) est le bit, qui peut prendre deux valeurs : 0 ou 1. Heureusement, le bit n'est pas la seule chose accessible à l'ordinateur; la structure de données suivante dans l'ordre de complexité croissante est l'octet.

L'octet est l'union de huit bits et permet de stocker des nombres variants de 0 à 255 (ou de -128 à 127 si l'on change d'interprétation, on parle dans ce cas de binaire signé en complément à deux). Si l'octet permet déjà un certain intervalle de variation, il est cependant insuffisant pour la majorité des applications, on utilise alors l'union de deux ou même quatre octets pour manipuler des nombres plus importants.

C'est le cas du Forth qui utilise, au choix du programmeur, deux ou quatre octets pour ses opérations arithmétiques. Il permet ainsi de traiter des nombres de 0 à 65535 (respectivement -32768 à +32767 en complément à deux) pour deux octets et de 0 à plus de quatre milliards pour quatre octets (soit approximativement de -2 milliards à +2 milliards en complément à deux). Le mode quatre octets est appelé mode double précision et permet le traitement de grands nombres moyennant une légère baisse de rapidité. Notons de plus que le Forth imprime toujours les nombres en complément à deux et autorise donc l'emploi de nombres négatifs.

1.2 La base hexadécimale

La base hexadécimale ou base seize est trés utilisée en informatique car elle est adaptée à la représentation des octets. Les chiffres employés sont 0,1,2,3,4,5,6,7,8,9 puis A,B,C,D,E,F. Ainsi B en hexadécimal est égal à 11 en décimal.

Un octet peut ainsi prendre en hexadécimal des valeurs de 0 à FF=255, deux octets des valeurs de 0 à FFFF et quatre octets de 0 à FFFFFFFF. On voit donc l'intérêt de l'hexadécimal qui permet de saisir facilement la valeur d'un ou de plusieurs octets.

```
Quelques exemples d'opérations en hexadécimal :

0F+01=10 soit en décimal 15+01=16,
80+21=A1 soit en décimal 128+33=161,
FF-1F=E0 soit en décimal 255-31=224.
```

1.3 Les piles en Forth

La pile en informatique correspond exactement aux piles que nous connaissons tous (piles d'assiettes...). Ainsi quand vous prenez un objet sur une pile, c'est toujours le dernier posé.

C'est la même chose en Forth. Quand vous voulez sauvegarder une donnée, il suffit de déposer cette donnée au sommet de la pile. Ulterieurement, il sera possible de prendre le dernier élément déposé pour l'afficher par exemple.

Il existe en Forth deux piles indépendantes. Tout d'abord la pile dite des données, utilisée le plus souvent par l'utilisateur et la pile dite de retour, utilisée quasi-exclusivement par le système Forth (bien que l'utilisateur y ait accès).

Pour déposer un nombre sur la pile des données, il suffit de taper ce nombre au clavier suivi de la frappe de la touche (ENTREE). (Nous noterons dorénavant entre parenthèses les touches comme ENTREE et CTL sans préciser qu'il s'agit des touches).

```
Exemple

1 (ENTREE)    : dépose 1 sur la pile des données,
2 (ENTREE)    : dépose 2 sur la pile,
. (ENTREE)    : affiche le nombre du sommet de la pile soit 2,
. (ENTREE)    : affiche le nombre suivant, ici 1.
```

Attention, le Forth ne comporte pas comme le Basic un éditeur pleine page permanent. Celui ci n'est disponible sous Forth que si vous charger l'éditeur (Cf le chapitre 9.0). En conséquence, les touches flèches que vous utilisez habituellement pour corriger les erreurs doivent être inutilisées si ce n'est la touche (←) qui efface et correspond à la touche EFF du Basic.

1.4 La notation polonaise inverse

Vous avez probablement l'habitude (à moins que vous ne possédiez une calculatrice Hewlett Packard) de calculer des expressions en utilisant la notation dite algébrique ou directe. Il existe d'autres notations dont la notation polonaise inverse qu'utilise le Forth.

Cette notation consiste à introduire les données du calcul d'abord, puis les opérations à effectuer sur ces données. Examinons l'exemple suivant dans lequel nous allons calculer 2*3=6.

```
Calcul de 2*3

2 (ENTREE)    : dépose 2 sur la pile,
3 (ENTREE)    : dépose 3 sur la pile,
* (ENTREE)    : multiplie les deux nombres au sommet de la pile
et laisse le résultat au sommet, à la place des deux nombres,
. (ENTREE)    : affiche le produit, soit 6.
```

Continuons avec un exemple un peu plus compliqué, le calcul de (2+5)*3+4, ceci tout en visualisant les mouvements des données sur la pile.

```
Calcul de (2+5)*3+4=25, Tapez :

  2 5 + 3 * 4 + (ENTREE)   : calcul de l'expression,
  . (ENTREE)               : affichage du résultat.

Observons l'évolution des données sur la pile:
```

2	5	+	3	*	4	+	.
2	5	7	3	21	4	25	-
-	2	-	7	-	21	-	-

Naturellememt la pile des données du Forth n'est pas limitée à deux ou quatres éléments comme les calculatrices HP mais possède autant d'éléments qu'il reste de mémoire à l'utilisateur. Il est donc quasiment impossible de la remplir, toutefois, si cela arrivait, le Forth vous en avertirait en signalant 'Full stack'.

2.0 GESTION DE LA PILE ET ARITHMÉTIQUE

La pile est une des caractéristiques les plus importantes d'un système Forth. En effet elle permet de passer les arguments entre les fonctions: à l'appel, la fonction prend les paramètres initiaux sur la pile, évalue la fonction et dépose les résultats sur la pile. Ainsi l'utilisation de variables est moins courantes qu'en Basic.

```
La pile sera représentée de la facon suivante :
     n1  n2  n3              n3 se trouve au sommet de la pile
                             n1 est l'élément le plus profond
( ceci correspond à l'ordre d'empilement)
Les notations utilisées dans la suite sont les suivantes:
          n          entier simple précision signé
          u          entier simple précision non signé
          d          entier double précision signé
          ud         entier double précision non signé
```

2.1 la pile

Etant donnée l'importance de la pile, il convient de disposer d'instructions capables de la modifier.

DUP : cette instruction permet de dupliquer le sommet de la pile

```
 n1  DUP   ---   n1 n1
Exemple: 1 5 DUP . . .  (ENTREE)
Affichage: 5 5 1
( . affiche l'élément se trouvant au sommet de la pile)
```

SWAP : cette instruction permet d'échanger les deux éléments se trouvant au sommet de la pile.

```
 n1 n2  SWAP   ---   n2 n1
Exemple: 1 2 SWAP . .   (ENTREE)
Affichage: 1 2
```

DROP : cette instruction permet d'éliminer l'élément se trouvant au sommet de la pile.

```
 n1 n2  DROP   ---   n1
Exemple: 5 7 DROP .    (ENTREE)
Affichage: 5
```

OVER : cette instruction permet de dupliquer l'élément se trouvant sous le sommet de la pile.

```
 n1 n2  OVER   ---   n1 n2 n1
Exemple: 0 5 OVER . . .    (ENTREE)
Affichage: 0 5 0
```

ROT : cette instruction place le troisiéme élément au sommet de la pile.

```
 n1 n2  n3  ROT   ---   n2 n3 n1
Exemple: -1 4 5  ROT . . .   (ENTREE)
Affichage: -1 5 4
```

PICK: cette instruction duplique le n-iéme élément au sommet de la pile. (n est la valeur se trouvant sur la pile) Le sommet de la pile est l'élément 0.

```
 n   PICK  ---   n-iéme élément
Exemple: 8 -1 0 5 11 3  PICK . . . . . .   (ENTREE)
Affichage: -1 11 5 0 -1 8
Le troisiéme élément (-1) remplace le 3 sur la pile
Exemple: 8 -1 0 5 11 0  PICK . . . . . .   (ENTREE)
Affichage: 11 11 5 0 -1 8
Le 0-iéme élément (11) remplace le 0 sur la pile
```

-DUP : cette instruction duplique le sommet de la pile seulement si il est non nul.

```
 n1  -DUP   ---   n1 n1          si n1 non-nul
Exemple: -5 1  -DUP . . .   (ENTREE)
Affichage: 1 1 -5

 0  -DUP   ---   0
Exemple: -5 0  -DUP . .     (ENTREE)
Affichage: 0 -5
```

Cette fonction est souvent utile avant un IF car elle évite parfois le ELSE DROP ENDIF.

R : cette instruction recopie le sommet de la pile de retour (return stack) sur la pile des données (data stack).

```
      R   ---   n1
```

R> : cette instruction transfert le sommet de la pile de retour (return stack) sur la pile des données (data stack).

```
      R>  ---   n1
Cette instruction  doit être utilisée avec précaution car la pile
de retour est utilisée par le système.
```

>R : cette instruction transfert le sommet de la pile des données (data stack) sur la pile de retour (return stack).

```
 n1  >R   ---
Cette instruction  doit être utilisée avec précaution car la pile
de retour est utilisée par le système.
```

Les 3 instructions précédentes permettent d'utiliser la pile de retour comme mémoire temporaire mais il faut veiller à préserver l'état de la pile de retour.

2.2 *L'arithmétique*

Le Forth travaille sur des nombres entiers signés. Il existe 2 types de précisions: simple précision et double précision. De plus il est possible de choisir la base utilisée pour l'entrée et l'affichage des nombres. Le Forth ne signale pas les dépassements de capacité.

2.2.1 Arithmétique simple précision

L'arithmétique simple précision porte sur des nombres stockés en mémoire sur deux octets. On dispose donc d'une capacité allant de -32768 a 32767.

+ : cet opérateur prend les deux élément se trouvant au sommet de la pile, les additionne et dépose le résultat au sommet de la pile


```
 n1 n2   +    ---   n1+n2
Exemple: 9 -7 5  +  . .    (ENTREE)
Affichage: -2 9
```

1+ : cet opérateur incrémente le sommet de la pile.

```
 n1   1+    ---   n1+1
Exemple: 12   1+  .     (ENTREE)
Affichage: 13
```

1- : cet opérateur décrémente le sommet de la pile.

```
 n1   1-    ---   n1-1
Exemple: 12   1-  .     (ENTREE)
Affichage: 11
```

2+ : cet opérateur additionne 2 au sommet de la pile.

```
 n1   2+    ---   n1+2
Exemple: 12   2+  .     (ENTREE)
Affichage: 14
```

2- : cet opérateur soustrait 2 au sommet de la pile.

```
 n1   2-    ---   n1-2
Exemple: 12   2-  .     (ENTREE)
Affichage: 10
```

L'avantage de ces opérateurs est un gain de temps et de mémoire.

- : cet opérateur prend les deux éléments se trouvant au sommet de la pile, soustrait le premier au second et dépose le résultat sur la pile.

```
 n1 n2   -    ---   n1-n2
Exemple: 8 3   -  .     (ENTREE)
Affichage: 5
```

MINUS : cet opérateur change le signe de l'élément se trouvant au sommet de la pile.

```
 n1 MINUS   ---  -n1
Exemple: 4  MINUS  .     (ENTREE)
Affichage: -4
```

+- : cet opérateur applique le signe de l'élément se trouvant au sommet de la pile à l'élément inférieur.

```
 n1 n2  +-  ---  n3
Exemple: 4 -2 +-  .     (ENTREE)
Affichage: -4
Exemple: 4 2 +-  .     (ENTREE)
Affichage: 4
```

ABS : cet opérateur donne la valeur absolue de l'élément se trouvant au sommet de la pile.

```
 n1   ABS  ---  Abs(n1)
Exemple: -5  ABS  .     (ENTREE)
Affichage: 5
```

* : cet opérateur prend les deux éléments se trouvant au sommet de la pile, effectue le produit et dépose le résultat sur la pile.

```
 n1 n2   *     ---   n1*n2
Exemple: 4 -2   *  .     (ENTREE)
Affichage: -8
```

/ : cet opérateur prend les deux éléments se trouvant au sommet de la pile, effectue la division entière et dépose le résultat sur la pile.

```
 n1 n2   /     ---   n1/n2
Exemple: 11 5   /  .     (ENTREE)
Affichage: 2
```

*/ : cet opérateur effectue une règle de trois sur les trois éléments du sommet de la pile (Le produit intermédiaire est calculé en double précision).

```
 n1 n2 n3  */     ---   n1*n2/n3
Exemple: 2 5 3  /  .     (ENTREE)
Affichage: 3
```

/MOD : cet opérateur effectue une division entière avec reste.

```
 n1 n2   /MOD     ---   reste n1/n2
Exemple: 9 2   /MOD  . .    (ENTREE)
Affichage: 4 1
```

*/MOD : cet opérateur effectue une règle de trois avec reste. Le produit intermédiaire est calculé en double précision.

```
 n1 n2 n3  */MOD     ---   reste  n1*n2/n3
Exemple: 11 2 3  */MOD  . .  (ENTREE)
Affichage: 7 1
```

MOD : cet opérateur calcule le reste de la division entière.

```
 n1 n2   MOD     ---   reste
Exemple: 11 3   MOD  .  (ENTREE)
Affichage: 2
```

MIN : cet opérateur laisse sur la pile le plus petit des deux nombres au sommet.

```
 n1 n2   MIN     ---   min(n1,n2)
Exemple: 15 3   MIN  .  (ENTREE)
Affichage: 3
```

MAX : cet opérateur laisse sur la pile le plus grand des deux nombres au sommet.

```
 n1 n2   MAX     ---   max(n1,n2)
Exemple: 15 3   MAX  .  (ENTREE)
Affichage: 15
```

AND : cet opérateur effectue un "et" logique entre les deux nombres au sommet de la pile.

```
 n1 n2   AND     ---   et(n1,n2)
Exemple: 5 4   AND  .  (ENTREE)
Affichage: 4
```

OR : cet opérateur effectue un "ou" logique entre les deux nombres au sommet de la pile.

```
 n1 n2   OR      ---   ou(n1,n2)
Exemple: 5 3   OR   .  (ENTREE)
Affichage: 7
```


XOR : cet opérateur effectue un "ou" exclusif entre les deux nombres au sommet de la pile.

```
 n1 n2   XOR     ---   oue(n1,n2)
Exemple: 5 3  XOR   .  (ENTREE)
Affichage: 6
```

Les opérations entre conditions logiques peuvent être faites à l'aide des opérateurs précédents. Pour plus de détails sur les tests, se reporter en 2.2.5 .

2.2.2 Double précision

Les nombres double précision sont représenté en mémoire sur 4 octets: ils sont donc plus long a manipuler, en revanche leur capacité est beaucoup plus importante (précision supérieure à celle du Basic). Pour rentrer un nombre en double précision dans la pile, il suffit de taper un point à la fin de celui-ci. Pour afficher un nombre double précision, il suffit de faire D. au lieu de '.'.

```
123456789.   D.      (ENTREE)
Affichage: 123456789
```

D+ : cet opérateur additionne le deux nombres double précision du sommet de la pile.

```
 d1 d2   D+      ---    d1+d2
Exemple: 222222.  333333.  D+  D.  (ENTREE)
Affichage:  555555
```

DMINUS : cet opérateur change le signe du nombre double précision du sommet de la pile.

```
 d1   DMINUS     ---    -d1
Exemple:   70000. DMINUS  D.  (ENTREE)
Affichage:  -70000
```

D+- : cet opérateur applique le signe du nombre au sommet de la pile au nombre double précision en dessous.

```
 d1  n   D+-     ---     d2
Exemple:   250000.  -1  D+-  D.  (ENTREE)
Affichage:   -250000
```

DABS : cet opérateur donne la valeur absolue du nombre double précision au sommet de la pile.

```
 d1    DABS     ---    abs(d1)
Exemple:   -38000  DABS  D.  (ENTREE)
Affichage:   38000
```

2.2.3 Opérations en précision mixte

U/ : cet opérateur effectue la division non signée avec reste d'un nombre double précision par un simple précision.

```
 ud1  u1  U/     ---     u2(reste) u3(quotient)
Exemple:   70002.  7  U/  .  .  (ENTREE)
Affichage:   10000  2
```

U* : cet opérateur effectue le produit de deux nombres simples précision non signés et fournit un résultat en double précision .

```
 u1  u2  U*     ---    ud3
Exemple:  20000  4  U*  D.  (ENTREE)
Affichage:  80000
```

M/ : cet opérateur effectue la division avec reste d'un nombre double précision par un nombre simple précision. Les deux résultats sont en simple précision.

```
 d1  n1  M/     ---     n2(reste)  n3 (quotient)
Exemple:  80003.  8  M/  .  .  (ENTREE)
Affichage:   10000  3
```

M* : cet opérateur effectue la multiplication des deux nombres simple précision du sommet de la pile et donne un résultat en double précision.

```
 n1  n2  M*     ---     d1          d1 est égal a n1*n2
Exemple:   -30000  4  M*  D.  (ENTREE)
Affichage:   -120000
```

M/MOD : cet opérateur effectue la division non signée d'un nombre double précision non signé par un nombre simple précision non signé. Le quotient est donné en double précision et le reste en simple précision

```
 ud1  u1  M/MOD     ---  u3(reste)    ud4(quotient)
Exemple:  360002.  3  M/MOD  D.  .  (ENTREE)
Affichage:  120000  2
```

S->D : cet opérateur convertit un nombre simple précision en un nombre double précision.

```
 n     S->D     ---    d
Exemple:   -56   S->D   D.  (ENTREE)
Affichage:   -56
```

2.2.4 Choix de la base

A l'initialisation le Forth est en base 10 . Il est possible de choisir la base dans laquelle va s'effectuer l'affichage et la saisie des nombres. La numérotation est classique: 1 2 ...9 A B C ... Voici les mots permettant d'agir sur la base utilisée.

DECIMAL : ce mot sélectionne la base décimale.

HEX : ce mot sélectionne la base hexadécimale (base 16)

Il existe une variable système contenant la base courante. Pour la modifier il suffit de faire : n BASE ! où n est la base désirée (exprimée dans la base courante). Pour plus de précisions sur les variables système se reporter au chapitre 12.1. La base 2 est particulièrement utile pour la définition des sprites.

2.2.5 Les flags

Les fonctions de test pour les boucles, les branchements conditionnels, etc sont des fonctions qui déposent un flag ou drapeau sur la pile, celui-ci est ensuite interprété par les instructions de contrôle (cf chapitre 5.0). En Forth une valeur non nulle représente la valeur logique vraie, et le 0 représente la valeur logique faux. Nous noterons tf la valeur logique vraie et ff la valeur logique faux (true flag et false flag). Nous allons maintenant décrire l'utilisation des fonctions retournant des valeurs logiques.

0< : cet opérateur teste le signe du nombre se trouvant au sommet de la pile et dépose un flag vrai si le signe est négatif.

```
    n     0<    -------   tf       si n negatif
    n     0<    -------   ff       si n positif (ff=0)
 Exemple: -4  0<   .     (ENTREE)
 Affichage:  1       vrai
 Exemple: 7   0<   .     (ENTREE)
 Affichage:  0               faux
```

0= : cet opérateur teste si le nombre se trouvant au sommet de la pile est nul. Utilisé sur une valeur logique, cet opérateur est équivalent à l'opération logique NOT.


```
    0     0=    -------   tf       si n nul
    n     0=    -------   ff       si n non nul
 Exemple: 0   0=   .     (ENTREE)
 Affichage:  1           vrai
 Exemple: 12  0=   .     (ENTREE)
 Affichage:  0                faux
```

= : cet opérateur teste l'égalité des deux nombres se trouvant au sommet de la pile.

```
 n1  n2   =     -------   tf       si n1=n2
 n1  n2   =     -------   ff       si n1≠n2
Exemple: 9 9  =   .     (ENTREE)
Affichage:  1           vrai
Exemple: 5 0  =   .     (ENTREE)
Affichage:  0           faux
```

> : cet opérateur teste les deux nombres se trouvant au sommet de la pile comme suit:

```
 n1  n2   >     -------   tf       si n1>n2
 n1  n2   >     -------   ff       si n1≤n2
Exemple: 9 4  >   .     (ENTREE)
Affichage:  1           vrai
Exemple: 1 4  >   .     (ENTREE)
Affichage:  0           faux
```

< : cet opérateur teste les deux nombres se trouvant au sommet de la pile comme suit:

```
 n1  n2   <     -------   tf       si n1<n2
 n1  n2   <     -------   ff       si n1n2
Exemple: -3 8  <   .     (ENTREE)
Affichage:  1           vrai
Exemple: 1 -4  <   .     (ENTREE)
Affichage:  0           faux
```

U< : cet opérateur teste les deux nombres non signés simple précision se trouvant au sommet de la pile.

```
 u1  u2   U<    -------   tf       si u1<u2
 u1  u2   U<    -------   ff       si u1u2
Exemple: 1 -1  U<   .     (ENTREE)
Affichage:  1           vrai car en hexa -1=FFFF
```

Pour conbiner ces tests, on peut utiliser les instructions (OR AND XOR).

3.0 : MANIPULATION DE LA MÉMOIRE

Le Forth permet une manipulation plus précise et plus efficace de la mémoire que celle autorisée par le Basic.Nous allons donc examiner le vocabulaire qui se rapporte aux manipulations mémoire.

@ : cette instruction permet de lire l'entier dont l'adresse est placée sur la pile. L'utilisation de cette instruction s'éclaircira la lecture des chapitres 1.1, 4.3 et 12.1.

```
addr  @ -----  n        n entier contenu en (addr,addr+1)
```

C@ : cette instruction permet de lire l'octet dont l'adresse est placé sur la pile.(C = memory Cell = octet)

```
addr  C@ -----  b        b octet contenu en (addr)
```

! : cette instruction permet de stocker un entier en mémoire.

```
n addr  !  -----      l'entier n est stocké en (addr,addr+1)
```

C! : cette instruction permet de stocker un octet en mémoire.

```
b addr  C!  -----          l'octet b est stocké en (addr)
```

L'utilité de ces opérateurs apparaitra au chapitre suivant.

+! : cette instruction permet d'additionner un entier à un entier dont on connait l'adresse.

```
b addr  +!  -----          l'entier b est additionné au
                               contenu de l'adresse addr
```

? : cette instruction affiche le contenu d'une adresse.

```
addr  ?  -----  n      n est l'entier contenu en (addr,addr+1)
```

CMOVE : cette instruction permet de recopier une partie de la mémoire en une autre partie de la mémoire.

```
dep  arr  nb CMOVE    ----
```

Cette instruction déplace nb octets de l'adresse dep à l'adresse arr Le contenu de dep est déplacé, puis celui de dep+1 etc.

CMOVE- : cette instruction permet de recopier une partie de la mémoire. La différence avec l'instruction CMOVE est que le déplacement se fait en descendant. Il est en effet utile de disposer des deux possibilités lorsque l'on manipule des zones se recouvrant.

```
dep  arr  nb  CMOVE-   ----
```

Cette instruction déplace nb octets de l'adresse dep à l'adresse arr Le contenu de dep+nb-1 est déplacé, puis celui de dep+nb-2 etc.jusqu'à dep.

FILL : cette instruction remplit une zone mémoire avec des octets de valeur donnée.

```
addr nb  b    FILL   ----
```


Cette instruction initialise nb octets à partir de l'adresse addr avec la valeur b.

ERASE : cette instruction remplit une zone mémoire avec des 0.

```
addr nb      ERASE   ----
```

Cette instruction initialise nb octets à partir de l'adresse addr avec la valeur 0.

BLANKS : cette instruction remplit une zone mémoire avec des 32 c'est à dire le code de l'espace.

```
addr nb      BLANKS  ----
```

Cette instruction initialise nb octets à partir de l'adresse addr avec des espaces (valeur 32)

Il convient d'être prudent dans les manipulations mémoire afin de ne pas détruire le Forth. Pour plus de précision se reporter à la carte mémoire donnée en annexe B.

4.0 LES DEFINING WORDS

Il existe plusieurs type de mots dans le vocabulaire du langage Forth. Les mots qui servent à définir le type d'un mot sont appellés Defining word ou mot de définition. Nous allons maintenant examiner ces mots et leurs utilisations respectives.

4.1 : ou colon

Le Forth permet à l'utilisateur de créer ses propres mots qui font alors partie du langage et qui pourront donc servir à créer d'autres mots etc. Cette caractéristique permet la mise en oeuvre d'une programmation structurée.

La syntaxe générale pour créer un nouveau mot est la suivante:

```
: nom-du-nouveau-mot  définition ;
où définition représente les instructions à exécuter.
```

Pour comprendre l'utilisation de ces instructions nous allons effectuer quelques exemples.

4.1.1 Exemple : CARRE

Nous allons créer un mot qui élévera au carré le mot se trouvant au sommet de la pile. Elevé un nombre au carré c'est le multiplier par lui-même. Donc si le nombre est au sommet de la pile, il suffit de le dupliquer et d'appliquer l'opérateur multiplication qui déposera le résultat sur la pile. Si nous décidons d'appeler ce mot CARRE , on le définit de la maniére suivante:

```
: CARRE DUP * ;     (ENTREE)
```

On vient donc de créer un nouveau mot qui fait partie du vocabulaire Forth. Si on fait VLIST on voit apparaitre en début de liste le mot CARRE (VLIST donne la liste des mots du vocabulaire Forth, pour plus de précision sur cette instruction se reporter au chapitre 6.3). L'opération qui consiste à rajouter un nouveau mot au vocabulaire s'appelle une compilation. Par conséquent : est un mot qui fait passer en état compilation et ; sert à clôturer la définition d'un mot. Nous pouvons maintenant utiliser le mot CARRE de la même façon que les autres mots du Forth, il peut même être utilisé pour construire de nouveaux mots.

```
Exemple :  5 CARRE .    (ENTREE)
Affichage:  25
Donc:       n     CARRE    -----     n*n
```

4.1.2 Exemple POLY

Nous allons créer un mot qui calculera l'image par le polynôme $2x^2+3x+5$ du nombre déposé au sommet de la pile. Pour cela il faut d'abord trouver la suite des opération à faire pour calculer la valeur du polynôme.


```
      Opérations                Pile

(état initial)      -----   x
          DUP       -----   x              x
          DUP       -----   x              x              x
           *        -----   x              (x*x)
           2        -----   x              (x*x)          2
           *        -----   x              (2*x*x)
          SWAP      -----   (2*x*x)        x
           3        -----   (2*x*x)        x              3
           *        -----   (2*x*x)        (3*x)
           +        -----   (2*x*x+3*x)
           5        -----   (2*x*x+3*x)    5
           +        -----   (2*x*x+3*x+5)
```

Une méthode plus efficace en temps et en place mémoire est possible. Pour cela on écrit le polynome sous la forme suivante : $(2x+3).x+5$. Ce qui donne en Forth:

```
      Opérations                Pile

(état initial)      -----   x
          DUP       -----   x              x
           2        -----   x              x              2
           *        -----   x              (2*x)
           3        -----   x              (2*x)          3
           +        -----   x              (2*x+3)
           *        -----   (2*x+3)*x
           5        -----   (2*x+3)*x      5
           +        -----   (2*x*x+3*x+5)
```

```
D'où la définition du mot POLY:
:  POLY  DUP  2  *  3  +  *  5  +  ;    (ENTREE)

Exemple d'utilisation:  2  POLY  .    (ENTREE)
Affichage: 19
Remarque: lorsqu'une définition est longue, on peut la taper sur
plusieurs ligne.Exemple avec POLY:
:  POLY  DUP  2  *  3          (ENTREE)
 +  *  5  +  ;                 (ENTREE)
```

Remarque : le mot Forth ;S correspond à la run-time routine (partie exécution) de ;

4.1.3 Conclusion:

Il apparait donc que l'écriture de programme Forth consiste à enrichir le dictionnaire de mots nouveaux. Les mots créés pourront bien évidemment utiliser des fonctions autres que les fonctions arithmétiques Les chapitres suivant seront consacrés à l'étude de mots nouveaux qui pourront être utilisés dans les définitions (structures de controles, manipulations graphiques etc).

4.2 Constantes:

Lorsqu'une valeur numérique est souvent utilisée dans un programme il est possible de lui donner un nom. C'est le rôle de l'instruction CONSTANT. La syntaxe en est la suivante:

```
valeur   CONSTANTE      nom-de-la-constante
```

Une fois compilée dans le dictionnaire, le mot 'nom-de-la-constante' a pour effet de pousser sur la pile la valeur de la constante.

```
Exemple: on veut utiliser la constante 320 que l'on baptise
 LONG ,on rentre donc :
   320   CONSTANT   LONG            (ENTREE)
Par VLIST on peut vérifier que le mot LONG figure en tête du
dictionnaire. Quand on a besoin de la constante il suffit de
taper LONG. De plus LONG peut aussi être utilisé dans une
définition.
    LONG   .      (ENTREE)
Affichage:   320
```

4.3 *Variables :*

Dans un programme il est souvent nécessaire de disposer d'emplacements mémoire permettant de stocker des variables. L'utilisation d'adresse mémoire est fastidieuse et source d'erreurs, aussi est-il souhaitable de pouvoir donner un nom symbolique à un emplacement mémoire.C'est le rôle de l'instruction VARIABLE. Cette instruction réserve deux octets où l'on pourra stocker un entier. La syntaxe est la suivante:

```
valeur-initiale   VARIABLE        nom-de-la-variable
```

Une fois compilé dans le dictionnaire, le mot 'nom-de-la-variable' a pour effet de pousser sur la pile l'adresse mémoire du premier des deux octets réservés. Une fois l'adresse sur la pile on peut utiliser les instructions de manipulation mémoire que nous avons vus au chapitre précédent (! et @).

```
Exemple: on veut créer la variable ALTITUDE qui a pour valeur
initiale 0 :
   0     VARIABLE   ALTITUDE         (ENTREE)
```

Par VLIST on vérifie que le mot ALTITUDE figure en tête du dictionnaire. ALTITUDE a pour effet de pousser sur la pile l'adresse où est stocké la variable, pour la lire il suffit donc de faire @

```
    ALTITUDE   .     (ENTREE)
Affichage:   adresse de la variable
Puis:
    ALTITUDE   @     .     (ENTREE)
Affichage:   0          (on vient de lire le contenu de ALTITUDE)

Pour modifier le contenu de ALTITUDE :
  100 ALTITUDE   !     (ENTREE)

On rappelle que ! stocke n à l'adresse addr:
   n    addr  !

Vérifions:

    ALTITUDE   @     .     (ENTREE)
Affichage:  100         (on vient de lire le contenu de ALTITUDE)

Pour ajouter un nombre au contenu de ALTITUDE :
  20    ALTITUDE   +!     (ENTREE)

Vérifions:

    ALTITUDE   @          (ENTREE)
Affichage:  120         (on vient de lire le contenu de ALTITUDE)
```


4.4 *Variables utilisateur:*

Il est possible d'utiliser des variables dont le stockage se fera dans une zone réservée de la mémoire (alors que les variables créées avec VARIABLE réservent deux octet dans le dictionnaire lui-même). Cette possibilité est parfois utile dans certains cas particuliers. On peut ainsi attribuer un nom symbolique à un emplacement de la zone mémoire réservée à ces variables. La syntaxe est :

```
 numéro     USER       nom-de-la-variable
(numéro=numéro de l'emplacement mémoire dans la zone réservée)
```

Il faut remarquer que si vous voulez réserver une emplacement mémoire pour deux octets, il sera nécessaire de sauter un numéro dans la suite des numéros des variables users. De plus, les 36 premiers emplacements sont réservés aux variables systèmes et seuls les emplacements 36 à 66 sont libres et réservés à l'utilisateur.

Une fois l'adresse sur la pile on peut utiliser les instructions de manipulation mémoire que nous avons vus au chapitre précédent (! et @ ou C! et C@ si un seul octet est utilisé).

```
Exemple: on veut créer la variable X23 dans les 40-ième et
41-ième emplacements de la zone réservée:
   40    USER    X23          (ENTREE)
```

X23 a pour effet de pousser sur la pile l'adresse où est stocké la variable, pour la lire il suffit donc de faire @ .

```
    X23   .     (ENTREE)
Affichage:   adresse de la variable

Pour initialiser le contenu de X23 :
   12    X23    !     (ENTREE)

Vérifions:
    X23   @     .     (ENTREE)

Affichage:    12          (on vient de lire le contenu de X23)
```

Pour plus de détail, voici la liste des variables systèmes avec le numéro de l'emplacement qu'elles utilisent:

```
USER 0  --->  FENCE
USER 2  --->  DP
USER 4  --->  VOC-LINK
USER 6  --->  S0
USER 8  --->  TIB
USER 10 --->  R0
USER 12 --->  WIDTH
USER 14 --->  WARNING
USER 16 --->  IN
USER 18 --->  BLK
USER 20 --->  SCR
USER 22 --->  CURRENT
USER 24 --->  CONTEXT
USER 26 --->  STATE
USER 28 --->  BASE
USER 30 --->  DPL
USER 32 --->  HLD
USER 34 --->  CSP
```

Pour plus de détails sur les variables systèmes reportez vous au chapitre 12.1.

4.5 CREATE:

Nous vous conseillons en première lecture de sauter cette partie concernant le mot CREATE.

Il est parfois utile de pouvoir créer un mot en langage machine afin d'augmenter les temps d'exécution ou d'utiliser certaines ressources particulieres du système (ROM, entrées-sorties etc). Pour définir un mot en langage machine, il faut en premier lieu créer une entrée dans le dictionnaire, puis compiler dans le PFA du mot les codes machines. L'écriture des codes machines dans le PFA s'effectue à l'aide des instructions , et C, .

Lors de l'écriture d'une routine en langage machine, il convient de respecter un certain nombre de règles. Tout d'abord il ne faut pas altérer le contenu des registres S, Y et DP, si on doit les utiliser il faut les sauvegarder et les restituer à la fin de la routine. Le registre DP contient $45 (hexa). Le registre Y est le pointeur d'instruction.A l'entré de la routine X pointe sur le CFA du mot. De plus U est le pointeur de pile des données. Il existe un zone mémoire de 10 octets réservés à l'adresse $454C, que l'on peut l'utiliser dans une routine. Les routines en langage machines ne doivent pas se terminer par l'instruction RTS: il faut utiliser un JMP à l'une des adresses suivantes:

```
NEXT   $45B6     passage à l'instruction suivante
PUSH   $45B4     pousse D sur la pile des données
                 et passe à la prochaine instruction
PULL   $45AC     dépile le sommet de la pile et passe
                 à la prochaine instruction
PUT    $45B0     le contenu de D remplace le nombre
                 se trouvant au sommet de la pile et
                 passage à l'instruction suivante
```

Le jump peut être un JMP en adressage direct (0E) sachant que DP vaut $45.

A la fin de la définition, il convient de taper SMUDGE afin que le mot puisse être trouvé.Par exemple: nous allons créer un mot qui effectue une multiplication par deux de façon nettement plus rapide que ne le feraient les instructions Forth 2 * . Les instructions langage machine sont:

```
PULU    D       37 06   lit le sommet de la pile
ASLB            58      décalage de B
ROLA            49      décalage de A
JMP     PUSH    0E B4   pousse D sur la pile
                        passage à l'instruction suivante
```

D'où

```
CREATE MULT2  HEX 3706 , 5849 , 0EB4 , SMUDGE
```

4.6 Autres defining word:

- Il existe d'autres defining words d'usage plus particulier. Il seront examinés dans des chapitres ultérieurs.

SPRITE : Il s'agit d'un mot permettent de créer des sprites pour l'animation graphique. Il sera examiné en détail dans le chapitre 11.4.

D'autres defining words seront créés par le programme EXTENS qui est fournit avec le Forth (Cf chapitre 13.2).

Le Forth offre la posibilité de définir des défining word: Ceci permet la création de structures de données originales. Le chapitre 7.0 sera consacré a cette caractéristique qui ne trouve son équivalente que dans peu de langages .

5.0 LES STRUCTURES DE CONTRÔLE

Nous allons maintenant étudier l'utilisation des structures de contrôle en Forth.

5.1 Les structures IF,ELSE,ENDIF

Commençons par le couple IF,ENDIF. Cet ensemble de mots permet des tests et correspond plus ou moins au couple IF,THEN du Basic MO5. Son emploi est strictement réservé à la création de mots (i.e. il ne peut être employé en mode direct) et a la syntaxe suivante :

```
<condition> IF <bloc conditionnel> ENDIF
```

Nous allons essayer de recréer le mot ABS du Forth qui donne la valeur absolue du nombre situé au sommet de la pile.

```
: ABS           ;début de la création du mot,
0<              ;si le nombre est négatif,
IF MINUS        ;changer son signe,
ENDIF ;         ;fin du bloc conditionnel et de la définition.
```

La seconde structure utilisable est la structure IF,ELSE,ENDIF qui correspond au IF THEN ELSE du Basic MO5. De même que pour la structure IF,ENDIF, ces mots ne sont pas utilisables en mode direct. La syntaxe est la suivante:

```
<Condition> IF <Bloc cond 1> ELSE <Bloc cond 2> ENDIF
```

Nous allons créer par exemple le mot 5= qui dépose 1 sur la pile si le nombre initial est égal à 5, et dépose -1 sinon.

```
: 5=            ;début de création du mot,
5 = IF 1        ;si le mot est égal à 5,on dépose 1
ELSE -1         ;sinon on dépose -1
ENDIF ;         ;fin de la structure et du mot.
```

5.2 Les structures BEGIN,AGAIN,UNTIL,WHILE,REPEAT

Ces structures de contrôle permettent toutes sortes de boucles controlées et n'ont pas leurs équivalents en Basic MO5. Comme toutes les structures de contrôle, ces mots ne sont pas utilisables en mode direct. Il existe trois modes d'utilisation de ces mots :

5.2.1 La structure BEGIN...AGAIN

Ces instructions correspondent à une boucle infinie et sont donc à employer avec trés grande précaution (sous peine de plantage du système). La syntaxe est trés simple:

```
BEGIN < bloc d'instructions > AGAIN
```

Dans ce cas la bloc est exécuté indéfinimemt.

5.2.2 La structure BEGIN...UNTIL

Il s'agit là encore d'une boucle, mais le dernier mot (UNTIL) permet la sortie de la boucle dès que la condition précédant le mot UNTIL est vraie (donc si un nombre différent de 0 est sur la pile à ce moment). Sa syntaxe est donc très semblable à celle de BEGIN...AGAIN :

```
BEGIN < bloc d'instruction > < Test de fin > UNTIL
```

```
Par exemple, nous pouvons créer un mot qui imprime tous les
nombres de 1 à 10.
 : 1A10          ;début de création
 0 BEGIN         ;dépose 0 sur la pile et commence la boucle
 1+ DUP .        ;incrémente le nombre sur la pile, le duplique,
                 l'imprime (il reste donc présent sur la pile)
 DUP             ;duplique à nouveau avant le test
 10 = UNTIL      ; test et fin de boucle
 DROP ;          ;laisse tomber le nombre restant et fin.
```

5.2.3 La structure BEGIN...WHILE...REPEAT

Ces trois mots permettent la réalisation de boucles qui comportent un test d'arrêt du type 'Tant que...' et non 'Jusqu'à ce que' comme dans la boucle BEGIN...UNTIL. La syntaxe de la structure est :

```
BEGIN < Bloc > < Test > WHILE < Bloc > REPEAT
```

Les deux blocs sont exécutés tant que le test précédant WHILE est vrai, et sort de la boucle après exécution du premier bloc si le résultat du test est faux.

5.3 *Les structures DO,LOOP et +LOOP*

Ces structures de contrôle se rapprochent beaucoup des boucles FOR NEXT du Basic MO5. Elles sont naturellement beaucoup plus rapides (de l'ordre de 25 fois pour une boucle à vide) et ne sont pas utilisables en mode direct. Deux structures sont possibles :

5.3.1 La structure DO...LOOP

Elle permet la réalisation de boucles du type FOR NEXT (i.e. avec un compteur fixant l'arrêt de la boucle) avec un incrément de compteur toujours égal à un. Les limites supérieures et inférieures de la boucle sont déposées sur la pile avant l'instruction DO et la boucle s'interrompt quand le compteur atteint la valeur supérieure moins un. A tout moment dans la boucle, la valeur courante du compteur peut être rappelée et déposée sur la pile grâce au mot I, de même, lors de l'utilisation de boucles imbriquées les mots J et K permettent l' accés aux compteurs des boucles externes (J : un niveau en dessus, K : deux niveaux en dessus).

Enfin, et toujours en rapport avec les structures DO...LOOP, le mot LEAVE, inséré dans une boucle aprés un test permet de quitter cette boucle en fixant la valeur du compteur à la valeur de la limite supérieure. Ainsi, a la prochaine rencontre de LOOP ou +LOOP, la boucle sera quittée.

La syntaxe de DO...LOOP est :

```
<limite sup> <limite inf> DO <Bloc> LOOP
```

```
Recréons le mot 1A10 de tout à l'heure (impression des
nombres de 1 à 10.) en l'appelant 1T010 pour les différencier.
 : 1T010          ;début de création du mot
 11 1 DO          ;dépot des limites et début de boucle
 I . LOOP ;       ;appel du compteur et impression, fin du mot
```


5.3.2 La structure DO... +LOOP

Cette structure permet le même type de boucles que les mots DO...LOOP, mais avec une variation du compteur qui n'est pas forcément égale à un (cela correspond donc à l'ajout de STEP dans les boucles FOR NEXT du Basic.). L'emploi est le même que pour la structure DO...LOOP en ce qui concerne les limites superieures et inferieures de la boucle. Par contre, la variation du compteur (le pas d'incrémentation du compteur) est à indiquer juste avant le mot +LOOP. La syntaxe est donc :

```
<limite sup> <limite inf> DO <Bloc> <Pas de variation> +LOOP
```

```
 Nous allons créer, par exemple, le mot PAIR, qui affiche tous
les nombres pairs entre 0 et 10.
 : PAIR           ;Début de définition du mot,
 11 0 DO          ;entrée des limites et début de la boucle,
 I .              ;affichage de la valeur du compteur,
 2 +LOOP ;        ;valeur du pas et fin de boucle et de mot.
```

5.4 *Les primitives de contrôle*

Les primitives du Forth en général sont des mots utilisés par le système Forth seulement, mais qui peuvent, à l'occasion, servir à l'utilisateur. Ces mots sont le plus souvent mis entre parenthèses comme (LOOP).(DO)... Pour le moment, nous vous conseillons de sauter ce chapitre qui est assez délicat et demande une bonne connaissance du Forth et de sa structure interne.

Considérons tout d'abord les deux primitives élémentaires BRANCH et 0BRANCH. Ces deux primitives sont équivalentes au GOTO du Basic, mais sont bien plus délicates à employer. Elles doivent être toutes les deux compilées pour être utilisables (par le mot COMPILE que nous verrons plus tard) et suivies par un nombre en simple précision qui représente un offset comme pour les instruction BNE du langage machine.

L'instruction BRANCH représente un branchement inconditionnel alors que l'instruction 0BRANCH est conditionnelle avec saut si l'élément sur la pile est nul au moment de l'exécution.

Ces deux mots (BRANCH et 0BRANCH) sont ennuyeux à utiliser tels quels car l'offset à compiler aprés eux n'est pas aisément calculable. C'est pourquoi il existe le mot BACK, qui bien que n'étant pas à vrai dire une primitive, n'est utilisé qu'en combinaison avec elles puisqu'il permet la calcul de l'offset.

Ce mot s'utilise juste aprés une compilation de BRANCH ou 0BRANCH, on suppose que sur la pile se trouve l'adresse de destination du saut, il suffit alors d'exécuter BACK qui se chargera de calculer et de compiler l'offset.

```
L'effet sur la pile est donc:
     addr     BACK      ----
```

A titre de curiosités, indiquons aussi les instructions (DO),(LOOP) et (+LOOP) qui ne sont guère utilisables et qui sont entièrement utilisées par DO,LOOP et +LOOP. (DO) est compilée par DO et se charge de déplacer les limites se boucles de la pile de données vers la pile de retour où elles pourront être conservées.

Quant aux mots (LOOP) et (+LOOP), ils sont tous deux compilés respectivement par LOOP et +LOOP et assure la gestion du compteur et des tests.

6.0 LE DICTIONNAIRE

La structure du langage Forth est trés differente de la plupart des langages en ce sens que l'ensemble des instructions forme ce que l'on appelle un dictionnaire extensible au gré de l'utilisateur et de la mémoire. Chaque instruction, constante, variable... créée est stockée dans le dictionnaire Forth.

6.1 La structure de chaque mot

Il est trés important pour dominer totalement le Forth de parfaitement connaître la structure interne du dictionnaire et la façon dont chacun des mots est compilé. Nous vous conseillons d'aborder ce chapitre tardivement car c'est un des plus difficile du manuel bien que sa compréhension soit le préalable indispensable à la compréhension totale du reste du Forth.

Le dictionnaire contient tous les mots Forth. Dans le cas du MO5, il s'étend vers le haut de la mémoire et le premier octet qui n'appartient pas au dictionnaire a son adresse stockée dans la variable système DP (Dictionnary Pointer) accessible par l'instruction HERE qui lit la variable DP.

Tout d'abord, nous allons voir la façon dont un mot est stocké dans le dictionnaire en considérant par exemple le mot CARRE qui a été créé au chapitre 4.1. .

	$85	Name Field adresse ou NFA
	C	
	A	
	R	
	R	
	E+$80	
high	LINK	Link field adresse ou LFA
low	LINK	
high	adresse de	Code field adresse ou CFA
low	DOCOLON	
high	adresse de	Parameter field adresse adresse ou PFA
low	DUP	
high	adresse de	
low	*	
high	adresse de	
low	;S	

On distingue pour chaque mot de vocabulaire, le header (ou en-tête) qui commence avec le name field (champ mémoire reservé au nom), le link field (champ reservé au lien), le code field (champ reservé au code) et la deuxième partie du mot qui n'appartient plus au header est appellée le parameter field (champ reservé aux paramètres). Nous allons examiner chacune de ces parties.

Le name field : Il commence avec un octet de longueur qui contient dans ses cinq bits les moins significatifs la longueur du nom du mot qui est donc limitée à 31. Le bit le plus significatif est toujours mis à un pour marquer le début du name field. Ensuite viens le nom lui-même, le dernier caractère ayant le bit le plus significatif mis à un afin de marquer la fin du name field.

Le link field : Il contient l'adresse du name field du mot créé précédemment. Ainsi l'ensemble des link field forment une chaîne à travers le dictionnaire ce qui permet une recherche rapide des mots. L' adresse du link field est appellée LFA (link field adress).

Le code field : Il contient l'adresse d'une routine directement exécutable en langage machine. La routine dépend de la façon dont le mot a été créé. Ainsi tous les mots créés à partir de ': ' auront dans leur code field l'adresse de DOCOLON.

Le parameter field : son contenu dépend de la manière dont le mot a été créé. Dans le cas d'une création à l'aide de ': ', le parameter field contient successivement l'adresse de tous les code field des mots utilisés dans l'écriture du mot. Ainsi CARRE contient successivement l'adresse du code field de DUP, puis de '*', puis enfin celui de ';S' qui est automatiquement compilé à la fin de chaque mot créé par ': '. Si le mot a été créé par un autre moyen, le parameter field a de bonnes chances de contenir tout autre chose.

6.2 Les vocabulaires

L'ensemble des link field parcourt tout le dictionnaire en chaînant tous les mots en une longue liste. Pour des propos de facilité d'utilisation et de rapidité de recherche, il est possible de séparer les mots en sous-ensembles nommés vocabulaires. Il en ainsi pour l'éditeur(EDITOR), les instructions musicales (MUSIC)...

L'utilisation de vocabulaires a deux effets: tout d'abord une augmentation de la vitesse de compilation en limitant la recherche des mots à une partie du dictionnaire. D'autre part, cela permet de d'avoir deux instructions avec le même nom dans deux vocabulaires différents qui soient distingués par le Forth.

Quand un nouveau vocabulaire est créé par l'instruction VOCABULARY, la chaîne des link field est détournée pour former une liste qui ne comporte qu'une partie de tout le dictionnaire.

6.3 Les instructions

FORGET : permet d'effacer de la mémoire tout les mots créés à partir d'un mot donné. Ainsi, si CUBE,CARRE et DRIAD ont été créés dans l'ordre précité, FORGET CARRE fera disparaître du dictionnaire CARRE et DRIAD.

VLIST : permet d'obtenir la liste de tous les mots du vocabulaire courant.

FORTH : nom du vocabulaire de base du Forth. Pour retourner sous ce vocabulaire, tapez simplement FORTH (ENTREE). A partir de ce moment, toutes les recherche de mots seront limitées à ce vocabulaire de base. Il faut remarquer que tous les vocabulaires créés par l'utilisateur contiennent le vocabulaire de base.

VOCABULARY : permet de créer un vocabulaire dans le dictionnaire. Une fois le vocabulaire créé, il suffit de taper le nom du vocabulaire puis (ENTREE) pour se trouver dans ce vocabulaire (c'est-à-dire que dorénavant, toutes les recherches seront effectuées dans ce vocabulaire). Pour que des définitions de mots soient placées dans un vocabulaire précis, il suffit de taper le nom de ce vocabulaire suivi de l'instruction DEFINITIONS. Ainsi FORTH DEFINITIONS (ENTREE) limite la recherche au vocabulaire de base FORTH et fait que toutes les créations de mots faites ultérieurement seront placées dans ce vocabulaire.

IMMEDIATE : ce mot placé à la fin de la création d'un mot (i.e. aprés le ';' dans le cas d'une instruction créée par ': ') permet de rendre ce mot immédiat, c'est-à-dire que ce mot ne peut plus être compilé car que ce soit en mode exécution ou en mode compilation, ce mot sera exécuté de toutes façons. Ainsi, les mots qui compilent une autre instruction qu'eux mêmes à la compilation sont immédiats (c'est le cas pour DO,LOOP,IF,ELSE,ENDIF....).

```
Exemple:     : HELLO ." BONJOUR" ; IMMEDIATE (ENTREE)
créé un mot immédiat.
Si maintenant nous tapons:  : TEST HELLO ;  (ENTREE)
Affichage: on obtient BONJOUR
```

C'est à dire que le mot HELLO a été exécuté au lieu d'être compilé. Ainsi, l'exécution du mot TEST ne fera rien du tout car il ne contient pas HELLO.

[COMPILE] : permet de compiler un mot immédiat. Il suffit pour cela d'écrier [COMPILE] nom-du-mot à la place du nom du mot pour que celui ci soit compilé dans l'instruction.

Exemple: : TEST2 [COMPILE] HELLO ; (ENTREE)

permet de compiler HELLO dans TEST2.

COMPILE : permet de compiler un mot dans le dictionnaire. Ainsi, COMPILE (DO) permet de compiler (DO). Cette instruction est en particulier utilisée dans les mots IF,DO,LOOP... qui sont immédiats et qui à la compilation s'exécutent pour compiler BRANC,(DO),(LOOP)....

LATEST : donne le NFA du dernier mot du vocabulaire utilisé.

HERE : donne l'adresse du premier octet libre au dessus du dictionnaire par une lecture de DP, variable système.

SMUDGE : Pour éviter que les mots qui comportent des erreurs soient exécutables par l'utilisateur (ce qui entrainerait un plantage du système), tous les mots, à la création sont "marqués" de façon à ne pas être exécutables. Ce n'est qu'à la fin du mot, lors de l'exécution de ';' que le mot est "démarqué" grâce à l'instruction SMUDGE. Ainsi, si vous avez créé un mot qui comportait une erreur et que vous vouliez l'oublier, le système vous indiquerait 'Not found' car le mot n'est pas "démarqué". Il suffirait d'exécuter alors SMUDGE, puis à nouveau FORGET pour pouvoir oublier ce mot.

LITTERAL : est une instruction utilisée par le système Forth. Elle compile dans le dictionnaire le nombre placé sur la pile si l'on est en mode compilation, elle ne fait rien sinon.

```
n  LITTERAL  ---  si mode compilation
                  si mode exécution
             n
```

DLITTERAL : est une instruction utilisée par le système Forth. Elle compile dans le dictionnaire le nombre double précision placé sur la pile si l'on est en mode compilation, elle ne fait rien sinon.

```
d  DLITTERAL  ---  si mode compilation
                   si mode exécution
              d
```

-FIND : permet de chercher l'emplacement mémoire d'un mot du dictionnaire dans le vocabulaire courant (CONTEXT) d'abord et de création (CURRENT) ensuite. Si le mot est trouvé, le PFA du mot, son octet de longueur et un drapeau logique vrai sont déposés sur la pile. Sinon, seul un drapeau logique faux est déposé.

```
-FIND name --  pfa octet-longueur tf   si trouvé
               ff                     si pas trouvé
```

(FIND) : est la primitive en langage machine de -FIND. La fonction et les résultats donnés sont les mêmes, seuls changent les arguments initiaux. Il faut fournir addr1, nfa à partir duquel il faut chercher le mot,et addr2 qui est l'adresse ou est stockée le mot à chercher.

```
addr1 addr2  (FIND) pfa  octet-longueur  tf   si trouvé
                    ff                       si pas trouvé
```

' : Cette instruction appellée 'tick' a deux modes d'utilisation. En mode exécution, elle cherche le pfa du mot dont le nom est indiqué et délivre un message d'erreur si le mot est absent des vocabulaires. En tant que directive de compilateur,elle compile le pfa comme littéral.

```
' name ---  pfa   ou message d'erreur
```

NFA : cette instruction permet de trouver le NFA d'un mot en connaissant son PFA.

```
pfa --- NFA  nfa
```

LFA : cette instruction permet de trouver le LFA d'un mot en connaissant son PFA.

```
pfa --- LFA  lfa
```

CFA : cette instruction permet de trouver le CFA d'un mot en connaissant son PFA.


```
pfa --- CFA  cfa
```

PFA : cette instruction permet de trouver le PFA d'un mot en connaissant son NFA.

```
nfa --- PFA  pfa
```

TRAVERSE : permet de se déplacer dans le name field d'un mot. Addr1 est l'adresse de l'une ou l'autre extrémité du name field Si n=1, le déplacement est effectué vers le haut de la mémoire; si n=-1, le déplacement est effectué vers le bas de la mémoire. L'adresse addr2 résultante est l'adresse de l'autre extrémité du name field.

```
addr1  n  TRAVERSE  ---  addr2
```

ALLOT : permet d'allouer de la mémoire au dictionnaire. Ainsi, 100 ALLOT déplace le sommet du dictionnaire de 100 octets vers le haut de la mémoire. Cette instruction permet essentiellement de reserver de la mémoire dans la création de mots du type <BUILDS, DOES>.

, : Cette instruction compile le nombre simple précision qui se trouve au sommet de la pile et alloue donc deux octets de plus pour le dictionnaire.

C, : Cette instruction compile le nombre simple précision de valeur inférieure à 255 au sommet du dictionnaire et alloue donc un octet de plus pour le dictionnaire.

7.0 LES D2 WORDS

Nous allons maintenant examiner une caractéristique du Forth qui le distingue des autres langages. Nous avons étudié au chapitre 4.0 les defining word ou mots de définition: ces mots permettent de définir le type du mot que l'on va créer. Nous allons voir que le Forth offre la possibilité de nouveaux types c'est à dire de créer de nouveaux defining word. Les mots qui permettent de réaliser cette opération sont les defining defining word : D2 words dans le jargon des programmeurs Forth.

Un defining word comporte deux parties bien distinctes: tout d'abord comment compiler le nouveau mot dans le dictionnaire et ensuite quelle opération effectuer quand le mot est exécuté. Un exemple avec les defining word déjà connus facilitera la compréhension de ce processus.

Commençons par exemple par le defining word CONSTANT. Au moment de la création d'une constante, il faut créer un entrée dans le dictionnaire avec le mot fourni par l'utilisateur (le nom de la constante), puis compiler la valeur de la constante dans le PFA du nouveaux mot. Ceci constitue la partie création ou construction. Ensuite lorsque le mot créé sera utilisé, il devra déposer le contenu de son PFA sur la pile. La routine qui effectue cette opération est la même pour toutes les constantes: c'est la partie exécution de CONSTANT

Considérons maintenant le mot VARIABLE. Au moment de la compilation celui-ci crée une entrée dans le dictionnaire et met la valeur initiale dans le PFA. Lorsqu'un mot de type VARIABLE est exécuté une routine se charge de pousser sur la pile l'adresse où est stockée la variable.

Il apparait donc que pour créer de nouveaux defining words, il faut définir deux choses bien distinctes: d'abord comment compiler le mot et ensuite quelle opération faire quand un mot (créé avec ce defining word) est exécuté . IL existe deux maniéres d'effectuer ce travail: l'une avec <BUILDS et DOES> qui fait uniquement appel au Forth, l'autre avec ;CODE qui fait appel au langage machine (ce qui autorise une vitesse supérieure). Nous allons examiner successivement ces deux possibilités.

7.1 Construction avec <BUILDS et DOES> :

<BUILDS crée une entrée dans le dictionnaire (NFA, LFA et CFA) avec le nom fournit aprés. Quand le mot crée sera exécuté le sommet de la pile contiendra le PFA du mot et l'exécution sera faite conformément à la partie DOES>. DOES> contiendra l'ensemble des mots à exécuter quand le mot sera exécuté sachant que le PFA est sur la pile. Un exemple va permettre d'y voir plus clair.

Commençons par un exemple sans intérêt pratique, mais assez simple à comprendre. Nous allons recréer le defining word CONSTANT à partir du Forth sans utiliser le langage machine. (Nous l'appelerons CONSTE)

Examinons la partie compilation: il faut créer l'entrée dans le dictionnaire, <BUILDS se charge de cette opération. Il faut ensuite prendre la valeur qui est au sommet de la pile et la compiler dans le PFA du mot; ceci s'effectue avec l'opérateur ',(cf chapitre 6.3 pour cet opérateur).

Partie exécution: il faut lire la valeur de la constante dans le PFA du mot et la déposer sur la pile. Or au moment de l'exécution de la partie DOES> le PFA se trouve sur la pile, il suffit donc de faire @

Doù finalement: CONSTE <BUILDS , DOES> @ ;

Lorsque l'on va taper 10 CONSTE DIX, une entrée va être ajoutée au dictionnaire avec le mot DIX et la valeur 10 va être mise dans le PFA. Lorsque l'utilisateur va taper DIX, le PFA va être lu (il contient la valeur 10) par @ (Il ne faut pas oublier que DOES> pousse sur la pile PFA donc pour lire son contenu il suffit de faire @

Nous allons maintenant créer un mot plus utile. Le Forth ne permet pas dans sa version de base de définir des constantes double précision. Pour créer une constante double précision, il faut tout d'abord

compiler sa valeur dans le dictionnaire, ceci peut être réalisé en compilant successivement deux nombres simples précision. A l'execution, il faudra restituer ces deux nombres (dans l'ordre correct) pour reconstituer la constante double précision.

On a donc:

: CONSTANTE2 <BUILDS , , DOES> DUP @ SWAP 2+ @ SWAP ;

, , compile les deux nombres du haut de la pile (2 simple précision = un double précision). A l'exécution l'adresse du PFA est sur la pile, on la duplique, on lit le contenu de la premiere partie, on swap ce qui ramène l'adresse au sommet de la pile, on ajoute 2 à celle-ci afin d'accéder à l'autre partie du nombre double précision, on lit et on swap pour restituer l'ordre initial.

Nous allons maintenant utiliser ce nouveau defining word pour créer une constante double précision: 123456789. CONSTANTE2 TOTO .Ceci crée le mot TOTO du type constante double précision. Vérifions le fonctionnement avec TOTO D.

Vous trouverez d'autres exemples de D2 words dans le screen EXTENS (tableau à une ou 2 dimensions, chaines etc).

7.2 ;CODE

Dans certains cas il est souhaitable de pouvoir écrire la partie exécution en langage machine afin de diminuer le temps d'exécution. Nous vous conseillons de relire la partie du chapitre 4.5 consacrée au defining word CREATE.

Il faut tout d'abord créer l'entrée dans le dictionnaire avec CREATE et ensuite indiquer le processus à suivre pour la compilation. Ensuite l'instruction ;CODE permet de revenir en mode exécution et de compiler la routine en langage machine à l'aide de l'intruction , .Un exemple va permettre de mettre en application ces principes.

Nous allons créer la constante double précision avec cette fois-ci la partie exécution écrite en langage machine. D'où :

: CONSTANT2M CREATE , , SMUDGE ;CODE HEX EC04 , 3606 , EC02 , 0EB4 ,

Examinons la partie compilation: CREATE va permettre de créer une entrée dans le dictionnaire avec le mot fournit aprés. Ensuite , , compile la valeur de la constante double précision dans le dictionnaire (processus analogue à l'exemple précédent). SMUDGE permet de "démarquer" le mot afin de le rendre exécutable (Cf chapitre 6.3).

Les valeurs hexadécimales compilées aprés correspondent au code suivant:

```
LDD     4,X     EC04    lecture de la partie low
PSHU    D       3606    on la pousse sur la pile
LDD     2,X     EC02    lecture de la partie high
JMP     PUSH    0EB4    D va être  poussé sur la pile
                        et passage à l'instruction suivante
```

Nous vous rapellons que à l'entrée de la routine donnée aprés ;CODE le registre X pointe sur le CFA du mot. Reportez-vous à CREATE (chapitre 4.5) pour les restrictions d'utilisation des registres internes.

Le mot (;CODE) est une primitive du Forth : c'est la run-time routine (partie exécution) de ;CODE

8.0 LA GESTION CASSETTE ET LES ÉCRANS

Sur le Forth comme sur tout autre langage, il est possible de sauvegarder et de recharger en mémoire des programmes et données. Ceci est possible grâce aux écrans ou plus communément screens en anglais.

Un écran représente une zone mémoire qui peut contenir exactement ce que peut contenir un vrai écran (donc 25 lignes de 40 caractères). Il y a 9 écrans en Forth, numérotés de 1 à 9. Les programmes se stockent sur les écrans exactement comme en mode normal, mais en employant l'éditeur (programme sur cassette stocké aprés le Forth lui-même).

8.1 Ordres s'adressant aux écrans

LIST : cette instruction permet de lister le contenu d'un écran. A l'initialisation du Forth, tous les écrans sont remplis de caractères ASCII 0. Aprés un LIST, tapez n'importe quelle touche pour retourner au mode de départ.

```
 n1 LIST  ---
Exemple: 4 LIST  (ENTREE)
Affichage: Liste de l'écran 4.
```

INDEX : cette instruction imprime la première ligne de de tous les écrans indiqués. La première ligne servant souvent de ligne de remarque pour indiquer le contenu d'un screen, on comprend l'utilité de cette instruction.

```
 n1 n2 INDEX ——
Exemple: 1 3  INDEX  (ENTREE)
Affichage: Affichage des premières lignes des écrans 1,2 et 3.
```

.LINE : Cette instruction imprime la ligne de l'écran indiquée par les deux nombres plaçés sur la pile.

```
 ligne écran  .LINE
Exemple: 1 3  .LINE  (ENTREE)
Affichage: Affichage de la première ligne de l'écran 3.
```

(LINE) : Cette instruction donne l'adresse et la longueur de la ligne indiquée par les deux nombres plaçés sur la pile. Il est donc possible d'utiliser l'instruction TYPE immédiatement aprés.

```
 ligne écran  (LINE)  addresse longueur
Exemple: 1 1  (LINE) HEX . . (ENTREE)
Affichage: 28 2100 (adresse et longueur en hexadécimal).
```

BLOCK : Cette instruction donne l'adresse de début de l'écran dont le numéro est présent sur la pile.

```
 écran  BLOCK  addresse
Exemple: 1  BLOCK  HEX . . (ENTREE)
Affichage: 2100 (adresse de début du premier écran en hexa).
```

C/L : Une constante égal au nombre de caractères par lignes sur un écran.


```
 C/L  n
Exemple: C/L  DECIMAL .  (ENTREE)
Affichage: 40
```

L/S : Une constante égal au nombre de lignes par écran.

```
 L/S  n
Exemple: L/S  .  (ENTREE)
Affichage: 25
```

LOAD : Cette instruction trés importante n'a pas le même sens que l'instruction LOAD du Basic. Elle permet l'interprétation de l'écran dont le numéro est indiqué sur la pile, c'est à dire que toutes les instructions contenues dans l'écran seront exécutées (y compris les instructions de création de mots).

```
 écran  LOAD  ---
```

--> : Cette instruction ne peut être utilisée que dans un écran. Elle permet d'indiquer au système qu'il faut continuer l'interprétation sur l'écran suivant.

```
 --->
Exemple: Si une --> est placée en fin d'écran 6, un 6 LOAD
(ENTREE) interprètera l'écran 6 et 7.
```

8.2 Les instructions cassettes

CLOAD : Cette instruction correspond au LOAD du Basic. Elle permet de charger les écrans indiqués sur la pile. Elle doit être suivie du nom désiré.

Tous les fichiers rencontrés sont affichés. Une recherche infructueuse peut être arrétée par appui sur (CNT)+C en permanence (La scrutation clavier n'est effectué qu'entre deux fichiers successifs).

```
 n1 n2 CLOAD name ---
Exemple: 1 3 CLOAD EDITOR permet de charger l'éditeurqui a été
sauvegardé sur les écrans 1,2 et 3.
```

CSAVE : Cette instruction correspond au SAVE du Basic. Elle permet de sauvegarder sur cassette les écrans indiqués sur la pile. Elle doit être suivie du nom désiré.

```
 n1 n2 CSAVE name ---
Exemple: 1 3 CSAVE TEST permet de sauvegarder sous le nom TEST le
programme placé dans les écrans 1,2 et 3.
```

MOTOR : Cette instruction permet de commander le moteur du lecteur de cassetes. Il doit être précédé d'un drapeau (un 0 indiquant un arrêt alors que le 1 indique un démarrage).

```
 n  MOTOR ---
Exemple: 1 MOTOR (ENTREE) fait démarrer la cassette.
```

Chaque face de la cassette est organisée de la façon suivante:

nom	screens	contenu
FORTH	Binaire	Système Forth
EDITOR	1 2 3	Editeur (cf chapitre
MUSIC	4	Extension musique
EXTENS	5	Extensions chaines et tableau
GAME	1 à 9	Exemple de programme Forth (Jeu vidéo)

9.0 L'ÉDITEUR

9.1 But de l'éditeur

Lors de la mise au point d'un programme,il est fastidieux d'avoir à retaper entierement des définitions pour les modifier. Il est donc souhaitable de disposer d'un outil permettant de corriger facilement un programme, c'est le rôle de l'éditeur.

L'éditeur permet donc de modifier le contenu des SCREENS. La mémoire du MO5 contient 9 screens (écran), c'est a dire 9 pages que l'on peut utiliser pour l'écriture des programmes. De plus, il est possible de sauvegarder ces screens sur cassette. Pour plus de détails reportez-vous au chapitre 7.0.

9.2 Utilisation de l'éditeur

9.2.1 Chargement de l'éditeur

Il faut tout d'abord charger l'éditeur à partir de la cassette. Pour cela on utilise l'instruction CLOAD : 1 3 CLOAD EDITOR . Sur la cassette le fichier EDITOR se trouve immédiatement aprés le code Forth.

Il est nécessaire de compiler l'éditeur, pour cela on utilise l'instruction LOAD : 1 LOAD . Dès que que la compilation est terminée, le message "Editor loaded" s'affiche à l'écran.

9.2.2 Fonction de l'éditeur

Une fois l'éditeur compilé, il faut taper EDITOR pour placer le Forth sous le vocabulaire EDITOR qui contient l'éditeur.

-Effacement d'un screen (écran):

```
n1 CLEAR          efface le screen n1
```

-Copie de screens

```
n1 n2 COPY        copie le screen n1 dans le screen n2
```

-Edition:

```
n1 EDIT
```

Cette instruction permet de se placer en mode édition dans le screen n1. Le contenu du screen n1 s'affiche et le curseur se place en haut à gauche de l'écran. Il est alors possible de taper les definitions etc et d'effectuer les corrections. On peut se déplacer sur tout l'écran à l'aide des flèches. Lorsque l'on sort d'une des limites de l'écran le curseur réapparait du coté opposé. Pour faciliter le travail de correction sous éditeur les fonctions suivantes sont disponibles:

EFF : efface le caractère se trouvant à la position courante du curseur et décale le reste de la ligne d'un caractère vers la gauche (si un mot est à cheval sur la fin de ligne celui-ci est décalé)

INS : insère un blanc à la position courante du curseur. Il y a troncature en fin de ligne.

BASIC+D : détruit la ligne courante et toutes les lignes inférieures se déplacent d'une ligne vers le haut.

BASIC+I : insére une ligne blanche au niveau de la ligne courante. Toutes les lignes inférieures se déplacent d'un ligne vers le bas. La dernière ligne du screen est perdue.

BASIC+N : remplit d'espaces la ligne courante.

BASIC+X : efface la fin de ligne à partir de la position courante du curseur.

BASIC+M : mémorise la ligne courante.

BASIC+R : rappelle la ligne qui a été mémorisée par la commande précédente . Celle-ci remplace la ligne courante.

BASIC+J : place à partir de la position courante du curseur la ligne immédiatement dessous (joint les deux lignes).

BASIC+S : coupe la ligne courante à partir de la position courante du curseur. La partie à gauche du curseur est inchangée. La partie à droite est insérée en dessous. La dernière ligne de l'écran est perdue (sépare la ligne en deux lignes).

BASIC+T : tabulation : déplace le curseur de 6 caracteres à droite.

BASIC+Q : permet de quitter le mode édition, on se retrouve sous Forth, vocabulaire EDITOR.

La meilleure méthode pour assimiler ces commandes est l'expérimentation.

-Edition aprés une erreur: EREDIT . Cette fonction doit être utilisée aprés une erreur survenue au moment de la compilation d'un screen (LOAD). En effet aprés une erreur l'interpréteur dépose sur la pile le numéro du screen et la position du mot qui a provoqué l'erreur. Ainsi la commande EREDIT permet de passer en mode édition pour le screen où l'erreur s'est produite. De plus le curseur est placé sur la fin du mot qui a provoqué l'erreur. Les possibilités de corrections aprés EREDIT sont les mêmes que celles décrites pour EDIT.

10.0 ENTRÉES SORTIES ÉCRAN

On peut classer ces instructions en trois grandes catégories: les entrées sorties numériques, les entrées sorties alphanumériques et les instructions permettant de changer les différents paramètres de fonctionnement de l'écran (c'est dans ce groupe que vous allez retrouvez l'essentiel des fonctions du Basic du type CONSOLE...).

10.1 Les entrées sorties numériques

Rappelons tout d'abord que toutes les entrées sorties de ce type s'effectuent toujours dans une base donnée (base d'ailleurs stockée dans la variable système BASE et accessible par @ et !) et que vous pouvez changer de base et vous placer en base dix en utilisant l'instruction DECIMAL ou en base seize en utilisant l'instruction HEX. Les autres bases sont accessibles par les instructions:

```
n  BASE  !    (ENTREE)    qui vous place en base n.
```

Pour plus de détail sur les bases reportez vous au chapitre 2.2.

10.1.1 Les sorties numériques

Ces instructions sont celles qui, à partir d'un nombre sur la pile, permettent d'afficher sa valeur. La plus usuellement employée est naturellement la fonction '.' qui affiche le contenu de la pile en simple précision.

```
. :        n  .  ----

Exemple :        3  .  (ENTREE)
Affichage:       3  bien sûr.
```

Des instruction équivalentes existent pour afficher des nombres en double précision ou pour formatter l'affichage. Il s'agit de :

.R' : permet d'afficher un nombre n1 en simple précision justifié à droite par n2 espaces. C'est particulièrement utile pour afficher des tableaux de chiffres où les colonnes doivent être verticales.

```
          n1  n2  .R  ----

Exemple:      3  4  .R  (ENTREE)
Affichage:    3  (avec une tabulation de quatre caractères)
```

D. : permet d'afficher un nombre d1 en double précision (il faut bien sûr que se trouve sur la pile un nombre en double précision).

```
          d1  D.  ----

Exemple:      700000.  D.   (ENTREE)
Affichage:    700000
```

D.R : permet d'afficher un nombre d1 en double précision (il faut bien sûr que se trouve sur la pile un nombre en double précision) justifié à droite par n2 espaces.


```
          d1  n2  D.R  ----

Exemple:      910000.  10  D.R   (ENTREE)
Affichage:    910000  (avec une tabulation de dix caractères)
```

C'est tout pour les instructions courantes d'emploi. Nous allons maintenant voir les instructions de base qui permettent de convertir des nombres et de les afficher avec un format pratiquement quelconque. L'utilisation en est un peu délicate, nous terminerons donc avec un exemple.

PAD : délivre l'adresse du PAD (buffer de sortie de texte) qui est situé au dessus du dictionnaire et dans lequel s'effectue toutes les manipulations de caractères destinés à la sortie.

<# : Cette instruction sert à indiquer au sytème Forth que le début d'une conversion va être réalisé. Les seules conversions possibles sont sur des entiers en double précision (si vous employer un entier en simple précision, il suffit d'utiliser l'instruction S->D).

: Cette instruction génère, à partir d'un nombre en double précision d1, le prochain caractère ASCII qui sera placé dans la chaîne de sortie. Elle laisse sur la pile d2, qui est le quotient de la division de d1 par la BASE courante. Cette valeur d2 est poussée sur la pile pour servir pour de futures utilisations de #. Signalons enfin que cette instruction doit être utilisée entre <# et #> pour être utile.

```
d1  #   d2
```

#S : Cette instruction génère les caractères ASCII de sortie et les stocke dans le PAD (Cf la carte mémoire du Forth) par une utilisation itérative de # jusqu'à ce qu'un zéro en double précision soit obtenu sur la pile (en d'autres termes, #S achève totalement la conversion d'un nombre dans la BASE courante).

```
d1  #S   d2=0
```

SIGN : permet de placer un signe moins si désiré dans la chaîne de caractères qui sera affichée à la sortie. Deux argument sont attendus: d'une part le nombre d qui en en cours de conversion, et en dessous sur la pile un entier n en simple précision qui indique le signe qu'il faut indiquer dans la conversion (si n<0, un signe '-' sera placé).

```
n  d  SIGN  d
```

HOLD : Cette instruction permet d'insérer un caractère alphanumérique quelconque dans la chaîne de caractères de sortie (par exemple un ': ' si l'on convertit des heures,minutes,secondes) . Comme toutes les instructions précédentes, elles n'est utile que si elle est employée entre <# et #> .

```
code ASCII  HOLD  ----

Exemple:     34  HOLD permet d'insérer des guillemets.
```

#> : dernière instruction de la série, indique que la conversion est terminée et laisse sur la pile l'adresse et la longueur de la chaîne de caractères créée lors de la conversion ce qui permet l'emploie de TYPE immédiatememt aprés.

```
d  #>  addr. long.
```

Aprés avoir saisi l'ensemble de ces instructions, nous allons voir un exemple afin d'en clarifier l'utilisation. Nous allons commencer par recréer le mot D. qui permet d'afficher un nombre en double précision.

```
: AFFI          ;début de création
SWAP OVER       ;permet de créer un entier simple précision
                ;qui est du signe du nombre initial
                ;sans perdre ce nombre.
DABS            ;on prend la valeur absolue de d car
                ;son signe a été conservé par SWAP OVER
<#              ;début de conversion
#S              ;génération de tous les ASCII
SIGN            ;génération du signe
#>              ;fin de conversion
TYPE    ;       ;affichage et fin de mot.
```

10.1.2 Les entrées numériques

Ces instructions permettent de convertir une chaîne ASCII de caractères en un nombre double précision. Le premier octet de la chaîne doit être la longueur de celle-ci. Si un point décimal était rencontré dans la chaîne, sa position sera retournée dans la variable système DPL, sinon DPL contient -1.

La première de ces instructions est DIGIT. Elle convertit le caractère ACCII c (en utilisant la base n1) en son équivalent binaire n2 et un drapeau logique. Si la conversion est réussie, le drapeau est vrai (donc différent de 0) sinon il est faux (donc nul).

```
c  n1  DIGIT  n2  tf  (réussie)
c  n1  DIGIT  ff      (raté)
```

(NUMBER) : convertit une chaîne ASCII commencant à addr1 + 1 en utilisant la BASE courante. La nouvelle valeur est accumulée au nombre en double précision d1 (concaténé) et donne d2. De plus, l'adresse addr2 du premier chiffre inconvertible est déposée sur la pile.

```
d1  addr1  (NUMBER)  d2  addr2
```

NUMBER : convertit (en utilisant la base courante) la chaîne stockée à l'adresse addr (où doit se trouver sa longueur) en un entier double précision.Si un point décimal est rencontré, sa position est stockée dans DPL. Si la conversion numérique est impossible, un message d'erreur sera délivré.

```
addr   NUMBER   d
```

10.2 Les entrées sorties alphanumériques

Ce sont toutes les instructions de saisie au clavier, d'impression de chaînes, et dans une trés faible part, de traitement de chaînes (pour plus de détail sur ces dernières instruction allez voir le chapitre 13.2,1).'

10.2.1 Les sorties alphanumériques

Commencons par l'instruction la plus élémentaire qui permet d'émettre un caractère (qu'il soit de contrôle ou non) à savoir EMIT. Le code ASCII du caractère à émettre doit être déposé sur la pile.

```
c   EMIT   ——

Exemple:    65  EMIT (ENTREE)
Affichage:  A
```

Si vous avez besoin de plus de renseignements sur ce qu'est le code ASCII des caractères du MO5, reportez vous au manuel de ce dernier.

TYPE : permet l'impression d'une chaîne de caractères de longueur l stockée à l'adresse addr. Attention, tous les caractères, même les caractères de contrôle, sont affichés.


```
addr 1   TYPE   ----
```

COUNT : Cette instruction laisse sur la pile l'adresse addr2 et la longueur l d'une chaîne commençant à addr1 et dont le premier octet est sa longueur. Typiquement, COUNT'est suivie par TYPE

```
addr1   COUNT   addr2  1
```

." : Cette instruction est l'équivalent du PRINT Basic. Elle doit être suivie d'une chaîne terminée par un '"' et permet l'impression de cette chaîne.

```
." chaîne"

Exemple:   ." FORTH MO5"  (ENTREE)
Affichage:    FORTH MO5
```

(.") : Cette instruction est une primitive de l'instruction ." . Elle est compilée par cette dernière et n'est pas utilisable autrement.

SPACES : Cette instruction permet d'afficher n espaces.

```
n  SPACES  ——
```

SPACE : Cette instruction permet d'afficher un espace.

BL : Cette instruction dépose sur la pile le code ASCII de l'espace. Donc BL EMIT est équivalent à SPACE.

CR : Cette instruction permet d'emettre un 'Carriage Return' ou retour de chariot et donc de passer à la ligne.

MESSAGE : Cette instruction affiche le message d'erreur dont le numéro est déposé sur la pile, ceci sans qu'aucun traitement n'ait lieu pour autant.

ID. : Cette instruction permet d'afficher le nom de l'instruction dont le NFA (Name Field Adress) est déposé sur la pile.

```
pfa   ID.
```

-TRAILING : est plutôt une instruction de traitement de chaîne et permet de supprimer les espaces restants à la fin d'une chaîne de longueur n1 située à l'adresse addr1 pour donner une chaîne de longueur n2 située à l'adresse addr2 où les espaces auront été supprimés en fin de chaîne.

```
addr1  n1  -TRAILING   addr2  n2
```

10.2.2 Les entrées alphanumériques

Commençons par l'instruction élémentaire KEY. Cette instruction attend l'appui d'une touche (de contrôle ou non) et délivre son code ASCII sur la pile. Elle est donc semblable à l'instruction INPUT$(1) du Basic.

```
----   KEY  ASCII code

Exemple:   KEY      (ENTREE) , puis appuyer sur  'A'
Affichage: 65
```

KEYF : est l'équivalent de KEY si ce n'est que d'une part il n'y a pas attente d'une touche mais scrutation rapide du clavier, et d'autre part le nombre délivré sur la pile n'est pas le code ASCII de la touche mais un code interne au MO5. Le mieux, pour connaître le code d'une touche est d'utiliser cette instruction KEYF.

```
----   KEYF   code MO5

Exemple:   KEYF      (ENTREE)
Affichage:   48   (si aucune touche n'est appuyée)
```

?TERMINAL : permet de scruter rapidement un groupe de touches particulier, à savoir le groupe (CNT)+C, qui sert traditionnellement à l'arrêt d'un programme. Un drapeau logique (0=faux) est déposé sue la pile et permet de savoir si ces touches ont été appuyées.

```
?TERMINAL    1     (CNT)+C appuyées
             0     (CNT)+C relachées
```

EXPECT : lit une chaîne au clavier et la stocke à l'adresse addr ceci jusqu'à ce que la touche (ENTREE) ait été frappée ou un nombre n de touches appuyées (de contrôle ou non) ait été atteint.

```
addr  n   EXPECT   ——
```

QUERY : a exactement la même fonction que EXPECT si ce n'est que l'adresse est fixée par le système et est stockée dans la variable système TIB (Terminal Input Buffer) et que la longueur attendue est forcément égale à 80.

```
----   QUERY   ——
```

ENCLOSE : est une primitive en langage machine qui est accessible à l'utilisateur. Elle permet de rechercher des chaînes dans la mémoire si celles ci sont entourées de séparateurs. Il faut lui fournir l'adresse de départ de la recherche et le caractère délimiteur (de toutes façons, zéro est un caractère délimiteur inconditionnel). Elle fournit au retour la même adresse, l'offset jusqu'au premier caractère qui n'est pas un délimiteur, l'offset jusqu'au premier délimiteur aprés la chaîne et l'offset pour le premier caractère non inclus dans la chaîne.

```
addr1  c  ENCLOSE  addr1  n1  n2  n3
```

WORD : est une instruction utilisant la primitive ENCLOSE. Elle lit les caractères stockés dans le Terminal Input Buffer par QUERY jusqu'au délimiteur spécifié puis transporte cette chaîne précédée d'un octet contenant sa longueur au sommet du dictionnaire. Pour plus d'indications, reportez vous au chapitre 6.0.

10.3 Instructions diverses

Ce chapitre comprend toutes les instructions de formattage de l'écran, de réglage des couleurs, taille des caractères ...

INK : applique à tous les caractères et points présents sur l'écran la couleur dont le numéro est sur la pile. (Les codes sont les mêmes qu'en basic). De plus les caractères seront dorénavant affichés en cette couleur (pas les graphiques).

```
n   INK   ----
```

PAPER : applique à tout le fond de l'écran la couleur dont le le numéro est sur ia pile. De plus les caractères seront dorénavant imprimés avec cette couleur de fond (pas les graphiques).

```
n PAPER  ----
```

COLOR : permet de changer la couleur d'affichage des caractères sans changer la couleur de tout l'écran. La pile doit contenir le code de couleur de fond et de forme désiré.

```
n1  n2  COLOR   ----
```

FRAME : permet de changer la couleur du cadre de l'écran. Le code de couleur désirée doit être sur la pile.

```
n  FRAME   ----
```


CLS : permet d'effacer l'écran. Aucun argument n'est nécessaire.

CONSOLE : permet de changer la taille de l'écran logique. Il faut fournir la limite supérieure et inférieure de l'écran.

```
n1  n2   CONSOLE  ----

Exemple:    5 12 CONSOLE  (ENTREE)  limite l'écran à 8 lignes.
```

LOCATE : permet de placer le curseur en une position donnée de l'écran. Le numéro de ligne puis de colonne doivent être poussés sur la pile.

```
n1  nc  LOCATE  ----
```

CURPOS : permet de connaître la position actuelle du curseur. Le numéro de colonne est au dessus du numéro de ligne aprés l'appel.

```
CURPOS   ----      n1  nc
```

GET : permet d'obtenir le code du caractère présent à la position indiquée sur la pile par nl numéro de ligne et nc pour la colonne.

```
n1  nc  GET  ----   c   (code ASCII)
```

ATTRB : permet de fixer les différents modes de travail de l'écran. L'appel doit être précédé d'un code dont la liste suit en décimal.

```
00 -----> caractères taille normale
01 -----> caractères double hauteur
02 -----> caractères double largeur
03 -----> caractères double dimension
04 -----> mode caractères et couleurs affichés
05 -----> mode caractères affichés seulement
06 -----> mode non incrusté
07 -----> mode incrusté
08 -----> scrolling normal
09 -----> scrolling lent
10 -----> mode page
11 -----> vidéo inverse
Les codes suivants sont illicites.
```

DEFGR : permet de définir des caractères utilisateurs tout comme DEFGRS pour le Basic. Pour l'utiliser, il faut calculer tout comme pour l'instruction Basic les huit codes de chacune des lignes du caractère désiré. Il faut ensuite pousser ces huits nombres sur la pile suivis du numéro du caractère. Pour afficher ultérieurement le caractère créé, il suffit d'aditionner 128 et le numéro du caractère créé et de l'émettre (par EMIT). Notez qu'afin de permettre la sauvegarde des caractères créés, ceux ci sont stockés sur l'écran 9 qui est donc inutilisable si l'on emploie DEFGR.

```
n1 n2 n3 n4 n5 n6 n7 n8 n°  DEGFR

Exemple: HEX FF FF FF FF FF FF FF FF 0 DEFGR (ENTREE)
suivi de 80 EMIT (ENTREE) affiche un rectangle plein.
```

11.0 LE GRAPHISME HAUTE RÉSOLUTION

Ce chapitre décrit les fonctions graphiques de type haute résolution. Il existe d'autres commandes agissant sur l'écran texte qui sont décrites dans le chapitre 10.3. Vous allez retrouver des fonctions analogues a celle du Basic, plus un certain nombre de fonctions nouvelles.

Toutes les fonctions graphiques à l'exception des sprites utilise l'encre graphique courante. Cette encre graphique se sélectionne de la manière suivante:

```
n  INKG  (ENTREE)
```

Si n est positif, les tracés graphiques se font en allumant les points et en appliquant la couleur indiquée par n (la numérotation est la même que celle du Basic)

Si n est négatif, les tracés graphiques se font en éteignant les points et en appliquant la couleur indiquée par n (la numérotation est la même que celle du Basic)

11.1 Points et traits:

PSET : Cette fonction prend les coordonnées du point sur la pile et agit selon la valeur de l'encre graphique.

```
 x y  PSET      ----
Exemple:4 INKG 300 30 PSET     (ENTREE)
Affichage:  Un point bleu s'allume dans le coin supérieur droit.
Exemple:-2 INKG 300 30 PSET    (ENTREE)
Affichage:  Le point précédement allumé s'éteint et la couleur
de fond de la cellule de 8 pixels devient rouge.
```

POINT : Cette fonction retourne la couleur du point dont les coordonnées sont sur la pile. Si le nombre retourné est positif, le point est allumé et sa couleur est donnée par le nombre (codes identiques à ceux du Basic). Si le nombre est négatif, le point est éteint et sa couleur est donnée par le nombre (comme pour le Basic).

```
 x y  POINT     ----     code
Exemple:4 INKG 300 50 PSET 300 50 POINT  .     (ENTREE)
Affichage:  Un point bleu s'allume dans le coin supérieur droit.
            L'interpréteur affiche 4 .
```

LINE : Cette fonction trace un segment de droite entre les deux points dont les couples de coordonnées se trouvent sur la pile.

```
 x1 y1 x2 y2   LINE      ----
Exemple: 10 30 310 30 LINE
Affichage: un segment horizontal se trace en haut de l'écran.
```

-LINE : Cette fonction trace un segment de droite entre le le dernier point allumé et le point dont le couple de coordonnées se trouvent sur la pile.

```
 x1 y1  -LINE      ----
```


11.2 Figures

BOX : Cette fonction trace un rectangle. Il suffit de déposer sur la pile les deux couples de coordonnées de deux coins diagonalement opposés (l'ordre dans lequel on donne le deux couples est sans importance).

```
 x1 y1 x2 y2   BOX     ----
Exemple:100 10 20 80 BOX
Exemple:100 150 10 50 BOX
```

BOXF : Cette fonction trace un rectangle plein . Le passage de paramètres est le même que pour BOX.

```
 x1 y1 x2 y2   BOXF    ----
Exemple:100 10 20 80 BOXF
Exemple:100 150 10 50 BOXF
```

CIRCLE : Cette fonction trace un cercle dont les coordonnées du centre et le rayon sont placés sur la pile.

```
 x1 y1 r  CIRCLE    ----
Exemple:160 100 50  CIRCLE
Affichage: Un cercle de centre 160 100 et de rayon 50
```

PAINT : Cette fonction permet de colorier une surface fermée. Pour cela on doit déposer sur la pile les coordonnées d'un point à l'intérieur de cette surface fermée. La frontière de cette surface est constituée par l'ensemble des points allumés quelque soit leur couleur. Si la frontière n'est pas fermée le coloriage va sortir et risque de remplir l'écran.

```
 x1 y1     PAINT     ----
Exemple: CLS 160 100 50  CIRCLE 160 100 PAINT
Affichage: Un disque de centre 160 100 et de rayon 50
           (par coloriage du cercle)
```

11.3 Le stylo optique et les manettes de jeu

INPUTPEN : Cette fonction dépose sur la pile les coordonnées du point et laisse un flag indiquant la validité de la mesure. La mesure n'est déclenchée que par appui sur l'interrupteur du stylo optique.

```
INPUTPEN    -----   x  y  tf        mesure valide
INPUTPEN    -----   ff             mesure invalide  (ff=0)
```

On est souvent ammené à utiliser cette instruction de la façon suivante: BEGIN INPUTPEN UNTIL . La boucle est répétée tant que la mesure est invalide et aprés le UNTIL les coordonnées se trouvent sur la pile.

INPEN : Cette fonction dépose sur la pile les coordonnées du point et laisse un flag indiquant la validité de la mesure. La mesure est faite instantanément sans attendre l'appui sur l'interrupteur du stylo optique.

```
INPEN    -----   x  y  tf        mesure valide
INPEN    -----   ff             mesure invalide  (ff=0)
```

On est souvent ammené à utiliser cette instruction de la façon suivante: BEGIN INPEN UNTIL .

PTRIG : Cette fonction dépose un flag indiquant si l'interrupteur du stylo optique est fermé ou pas.

```
PTRIG    -----   tf        interrupteur fermé (appuyé)
PTRIG    -----   ff        interrupteur ouvert (relaché)
```

JOYSTK : Cette fonction dépose sur la pile l'orientation du joystick et un flag qui indique si la touche de mise à feu est enfoncée. Avant d'eppeller cette fonction il est nécessaire de déposer sur la pile le numéro du joystick que l'on désire tester.

```
JOYSTK   -----   p  tf          touche action enfoncée
JOYSTK   -----   p  ff          touche action non enfoncée
p indique la position du joystick:

                    1
                  8   2
                7   0   3
                  6   4
                    5
```

11.4 Sprite

Le Forth MO5 permet de définir des sprites, c'est à dire des dessins d'objets que l'on peut placer où l'on veut sur l'écran. De plus les sprites n'altèrent pas les dessins sur lesquels ils se déplacent (sauf s'il y a scrolling de la page écran). Ils prennent la couleur de l'encre des cellules écran où ils sont affichés. Les sprites peuvent sortir partiellement de l'écran. Pour effacer un sprite il suffit de le placer entiérement en dehors de l'écran. Les sprites sont particulierement utiles lors de la réalisation de jeux vidéo: ils permettent en effet de réaliser des animations rapides sans exiger l'écriture de programme en assembleur. De plus le nombre de sprites utilisables est seulement limité par la capacité du dictionnaire. Nous allons maintenant détailler leur mise en oeuvre.

11.4.1 Définition du sprite

Le dessin du sprite se fait dans un rectangle dont la longeur est un multiple de 8. Il convient donc de réaliser le dessin sur une grille. Il faut ensuite convertir numériquement le dessin. Sur ce point le Forth est plus souple que le Basic en ce qu'il permet l'utilisation de la base 2. Le dessin est entré octet par octet dans la pile des données (en suivant l'ordre des lignes du dessin). Puis on pousse sur la pile la largeur exprimée en nombre de cellules de 8 pixels, puis on pousse sa hauteur. Ensuite on va créer un mot qui sera utilisé par la suite pour manipuler le sprite: ceci se fait à l'aide du Defining word SPRITE. On tape donc : SPRITE NOM—DU—SPRITE. Un exemple facilitera la compréhension de la méthode à suivre.

```
On veut créer un sprite ayant la forme d'un rectangle de 14
pixels de large et 6 pixels de haut.
Taper 2 BASE ! pour passer en base 2, puis:
 11111111   11111100      (ENTREE)  (premiere ligne du dessin)
 10000000   00000100      (ENTREE)  (deuxième ligne du dessin)
 10000000   00000100      (ENTREE)  (troisième ligne du dessin)
 10000000   00000100      (ENTREE)       . . .
 10000000   00000100      (ENTREE)
 11111111   11111100      (ENTREE)

DECIMAL
2          (ENTREE)        repasse en base 10
6          (ENTREE)        largeur de 2 cellules (14<2*8)
SPRITE RECTANGLE (ENTREE)  hauteur de 6 pixels
                           ceci crée un sprite qui est rangé
                           dans le dictionnaire sous le nom
                           RECTANGLE. De plus les données
                           introduites précédement ont été
                           dépilées.
VLIST    (ENTREE)          Le mot RECTANGLE figure en début
                           du dictionnaire
```

D'autres exemples de définition de sprites sont disponibles dans le jeu: se reporter au chapitre 14.1.

11.4.2 Utilisation du sprite

Le fait de déplacer un sprite d'un endroit à un autre ne modifie pas les dessins déjà existants sur l'écran. (Sauf s'il y a eu scrolling de l'écran)

```
Pour placer un sprite à une position données de l'écran,
     il suffit de mettre sur la pile les coordonnées du point
     de l'écran où doit s'afficher le coin supérieur gauche
     du dessin:  x    y     nom-du-sprite
Exemple: avec le sprite crée précédement:
CLS 200 20 RECTANGLE      (ENTREE)
Puis:
310 20 RECTANGLE     (ENTREE)
Puis:
1000 20 RECTANGLE    (ENTREE)   Le sprite disparaît (hors écran)
```

WHERE : cette instruction permet de trouver les coordonnées courantes d'un sprite connaissant sont PFA.

```
   pfa       WHERE  -----    x   y
Pour connaitre le pfa connaissant le nom du sprite il suffit
de faire:  ' nom-du-sprite     ceci dépose sur la pile le pfa
du sprite (pour plus de précisions sur cette fonction cf
chapitre 6.3).
Exemple: CLS 10 250 RECTANGLE    (ENTREE)     affichage du sprite
         ' RECTANGLE    (ENTREE)     le pfa est sur la pile
         WHERE   (ENTREE)  . .
Affichage:   250 10            (y puis x)
```

12.0 LES MOTS DU SYSTÈME

12.1 Les variables systèmes

Le forth travaille dans un environnement toujours variable, à savoir que la base de travail, la taille du dictionnaire, le vocabulaire actuellement utilisé.... peuvent varier au gré de l'utilisateur. Tous les paramètres de fonctionnement sont donc stockés dans des variables dites systèmes qui sont accessibles à l'utilisateur comme toutes les variables. Il faut noter que toutes les variables systèmes sont définies comme des variables utilisateur (Cf USER pour plus de détail). En voici une liste avec leur utilité.

BASE : C'est la variable système qui contient la base de travail courante. Pour changer son contenu, il suffit d'employer les instructions classiques (@ pour lire et ! pour écrire).

BLK : C'est la variable système qui contient le numéro de l'écran actuellement en cours d'interprétation. Ainsi, si l'instruction 5 LOAD a été employée, BLK contient 5. En mode direct, BLK est mis à 0.

CONTEXT : Cette variable système contient un pointeur vers le vocabulaire à l'intérieur duquel les recherches de mots commenceront. Voir VOCABULARY,DEFINITIONS et CURRENT.

CURRENT : Cette variable système contient un pointeur vers le vocabulaire dans lequel les définitions seront stockées. Il faut savoir aussi que le vocabulaire CURRENT est celui dans lequel les recherches sont effectuées si un mot n'a pas été trouvé dans le vocabulaire CONTEXT. Voir VOCABULARY et DEFINITIONS

CSP : est la variable système qui permet un stockage temporaire de la position de la pile de données (par exemple au début de la création d'un mot) pour vérifier qu'elle ne s'est pas déplacée (à la fin de création du mot). Voir aussi !CSP et ?CSP.

DP : est la variable système qui pointe sur le sommet du dictionnaire (d'ou son nom Dictionnary Pointer). Elle contient l'adresse de la prochaine position mémoire libre au sommet du dictionnaire. Sa valeur peut être altérée par ALLOT et lue par HERE.

DPL : est la variable système qui contient le nombre de chiffres à la droite du point décimal dans l'entrée de nombre en double précision . Sa valeur par défaut dans le cas de l'emploi de nombres en simple précision est -1.

FENCE : Cette variable système contient une adresse en dessous de laquelle l'instruction FORGET est inactive (dictionnaire protégé). Pour oublier un mot en dessous de FENCE, il faut modifier son contenu.

HLD : Cette variable système contient l'adresse du dernier caractère de texte dans le cas d'une sortie numérique. Bien qu'accessible à l'utilisateur, son utilisation n'est pas fréquente.

IN : Cette variable système contient le décalage à l'intérieur du bloc de données actuellement utilisé (écrans ou mode direct) à partir duquel le prochain mot sera interprété. L'instruction WORD déplace et utilise la variable IN.

R0 : Cette variable système contient la valeur de départ de la pile de retour (i.e. la valeur à laquelle sera initialisée le pointeur de pile de retour lors de la prochaine initialisation par COLD par exemple).

S0 : Cette variable système contient la valeur de départ de la pile de données (i.e. la valeur à laquelle sera initialisée le pointeur de pile de données lors de la prpchaine initialisation par COLD ou lors de la prochaine erreur de l'utilisateur).

SCR : Cette variable système contient le numéro de l'écran actuellement utilisé (essentiellement valide lorsque qu'un LIST ou un EDIT a été exécuté).

STATE : Cette variable système contient l'état dans lequel est actuellement la système Forth. Si son contenu est nul, cela signifie que l'on est en mode exécution alors qu'un contenu non nul indique un mode compilation (mode utilisé entre : et ; par exemple).

TIB : Cette variable système contient l'adrese du Terminal Input Buffer. Ce buffer contient, en mode direct, la ligne d'instruction actuellement en interprétation.

VOC-LINK : Cette variable système contient l'adresse d'un champ dans le parameter field du vocabulaire le plus récemment créé. Toutes les définitions de vocabulaires sont reliées entre elles grâce à la variable VOC-LINK, ce qui permet des FORGET à travers de multiples vocabulaires.

WARNING : Cette variable système permet de modifier le traitement d'erreur du Forth. Sa valeur initiale est un. Elle indique que lorsqu'une erreur est présente, un message d'erreur sera affichée puis l'instruction ABORT qui redémarre le système sera exécuté. Si sa valeur est nulle, aucun message d'erreur ne sera donné, seul un numéro d'erreur sera indiqué, puis ABORT sera exécuté. Enfin, si WARNING est égal à -1, l'instruction (ABORT) qui est là pour être redéfinie par l'utilisateur s'il désire un nouveau traitement d'erreur sera exécuté. Voir MESSAGE et ERROR.

WIDTH : Cette variable système contient la longueur maximale des noms d'instruction du Forth (ici limitée à 31 caractères). En aucun cas WIDTH ne doit contenir un nombre supérieur à 31, mais il est possible, pour des propos d'économie de mémoire de limiter la longueur des noms d'instruction à une valeur inférieure. Dans ce cas, bien sûr, le nom des instructions trop longs sera tronqué.

Signalons enfin que toutes les variables systèmes sont réinitialisées par le système par les instructions COLD et WARM. Les valeurs utilisées pour reinitialiser les variables systèmes peuvent être atteintes grâce à l'instruction +ORIGIN qui donne l'adresse de stockage de la n-ième de ces valeurs:

```
n  +ORIGIN  addr
```

L'ordre dans lequel sont stockées ces valeurs (appelées Boot-up) est le suivant:

```
 0 +ORIGIN  ----> FENCE
 2 +ORIGIN  ----> DP
 4 +ORIGIN  ----> VOC-LINK
 6 +ORIGIN  ----> S0
 8 +ORIGIN  ----> TIB
10 +ORIGIN  ----> R0
12 +ORIGIN  ----> WIDTH
14 +ORIGIN  ----> WARNING
16 +ORIGIN  ----> NFA du dernier mot du FORTH
```

Il est donc possible de modifier ces Boot-up afin de rendre par exemple une vocabulaire inoubliable en tapant:

```
HERE DUP 0 +ORIGIN !
2 +ORIGIN !
LATEST 16 +ORIGIN !
HERE FENCE !
```

12.2 Le traitement d'erreur

Bien que le langage Forth n'offre pas comme le Basic un environnement protégé, un traitement d'erreur existe en Forth qui permet d'éviter la trés grande majorité des plantages et permet une localisation facile de la cause de l'erreur.

Tout d'abord, signalons qu'une erreur est signalée généralement par un ? suivi du message d'erreur. Dans ce cas, la pile des données est vidée de son contenu et le système dépose dessus deux nombres simple précision qui sont la valeur de BLK et de IN au moment de l'erreur. Ceci permet de connaitre ou l'erreur a été commise. Ainsi, si aprés une erreur on trouve BLK = 5 et IN = 20, on sait que l'erreur est due au mot qui est à la vingtième position de l'écran 5.

ERROR : exécute le traitement d'erreur notifié par WARNING (cf ce mot au chapitre précédent). Si WARNING est différent de -1 (cas du traitement d'erreur spécifié par l'utilisateur) Le message d'erreur

dont le numéro est sur la pile est affiché, l'instruction ABORT est ensuite exécutée puis le contenu actuel de IN et BLK sont déposés sur la pile.

```
numéro erreur   ERROR  in  blk
```

?ERROR : donne le message d'erreur numéro n si le drapeau logique fl situé sur la pile est vrai. L'instruction est inactive sinon.

```
fl  n  ?ERROR  in  blk
```

(ABORT) : est l'instruction exécuté par ERROR si WARNING est égal à -1. Elle permet à l'utilisateur de redéfinir son traitement d'erreur en recréant le mot (ABORT).

?COMP : donne un message d'erreur si l'on est pas en mode compilation (STATE nul).

?CSP : donne un message d'erreur si la position actuelle de la pile de données n'est pas égale à celle stockée dans CSP par !CSP.

?EXEC : donne un message d'erreur si l'on est pas en mode exécution (STATE différent de zéro).

?LOADING : donne un message d'erreur si l'on est pas en train de compiler un écran par LOAD.

?PAIRS : donne un message d'erreur si les deux nombres situés au sommet de la pile ne sont pas égaux. Cette instruction est utilisée pour vérifier que les structures de contrôles sont bien construites.

?STACK : donne un message d'erreur si la pile de données sort des limites qui lui sont imposées.

12.3 *Les instructions interpréteur*

Le langage Forth est un langage mi-compilé, mi-interprété, mais en fait ces deux aspects du Forth sont regroupés dans l'interpréteur qui suivant l'état de STATE interprètera ou compilera les instructions qu'il rencontre. Nous allons voir ici les différentes instructions qui forment l'interpréteur.

INTERPRET : Ce mot représente l'instruction utilisée en permanence par le Forth pour interpréter les instructions venues du clavier ou des écrans. Si le mot rencontré ne peut être trouvé ni dans le vocabulaire CONTEXT, ni dans le vocabulaire CURRENT, il est convertit en nombre en utilisant la base courante. Si ceci est également impossible, un message d'erreur 'Not found', précédé du mot en question est délivré. Si un point décimal est trouvé durant l'interprétation d'un nombre, une valeur en double précision sera considérée. Le point décimal n'a d'autres propos que cette distinction.

QUIT : Vide la pile de retour, arrête la compilation si il y en a une en ce moment et rend le controle à l'utilisateur.

ABORT : Vide la pile de données, puis exécute un QUIT.

WARM : redémarrage à chaud du système Forth. Les piles de retour et de données sont vidées. Les variables systèmes suivantes sont réinitialisées à partir des Boot-up (valeurs initiales de toutes les variables systèmes): WIDTH, TIB, WARNING, STATE, BASE, S0, R0...

COLD : redémarrage à froid du système Forth. Toutes les variables systèmes sont réinitialisées aux valeurs stockées dans les Boot-up (pour modifier ces valeurs, voir +ORIGIN). Tous les mots créés par l'utilisateur sont donc oubliés.

EXECUTE : exécute la définition dont le CFA est sur la pile de données.

```
cfa   EXECUTE  ---
```

CLIT : cette instruction est compilée par le Forth devant tout nombre en demi-précision, il permet d'indiquer au Forth que l'octet qui suit doit être interprété comme un nombre et non comme une instruction.

LIT : tout comme CLIT, cette instruction est compilée par le Forth devant tout nombre en simple précision pour indiquer que les deux octets qui suivent doivent être interprétés comme un nombre et non comme une instruction. : SP@ : Délivre la position de la pile des données avant que le résultat de l'instruction elle-même ne soit déposée sur la pile.

!CSP : Sauvegarde la position courante de la pile des données dans la variable système CSP pour être ultérieurement vérifiée par ?CSP.

SP! : Réinitialise la valeur du pointeur de pile des données en utilisant la valeur indiquée par les Boot-up (Cf +ORIGIN).

RP! : Réinitialise la valeur du pointeur de pile de retour en utilisant la valeur indiquée par les Boot-up (Cf +ORIGIN).

(: indique le début d'une remarque. Cette remarque doit se terminer par ')'. Exemple: (Editeur Forth) est une remarque.

[: permet de forcer le système Forth en mode exécution. Cette instruction est utile pour faire exécuter des mots durant une compilation. Pour plus de détails, voir] .

] : permet de forcer le système Forth en mode compilation. Cette instruction est utile pour faire exécuter des mots durant une compilation en l'utilisant conjointement avec [. Ainsi [HEX] placé au milieu d'une définition de mot permet de faire interpréter le début de la définition dans une base et la fin en hexadécimal.

Les caractères] et [n'existent pas sur le clavier du MO5. Pour accéder à ces caractères, il faut utiliser les touches CNT A et CNT Z.

13.0 LES EXTENSIONS (MUSIC ET EXTENS)

13.1 Le programme MUSIC

Ce programme est contenu dans l'écran 4 sur la cassette sous le nom MUSIC, juste aprés l'éditeur. Il faut donc le charger et le compiler:

```
 4 4 CLOAD MUSIC (ENTREE)
4 LOAD (ENTREE)
Le message "Music loaded" doit alors s'afficher.
```

Les notes sont directements accessibles par leur nom suivi d'un point. Ainsi DO. permet d'émettre un DO, FA#. permet d'émettre un FA dièse. De plus, l'instruction PAUSE permet de faire une pause.

Il est possible, comme pour le Basic, de choisir l'octave, le tempo, la durée et l'attaque. Les instructions correspondantes sont:

OCTAVE : permet de se placer sur l'octave dont le numéro est placé sur la pile.

```
 n  OCTAVE ---
Exemple: 3 OCTAVE (ENTREE) vous place sur l'octave 3.
```

TEMPO : Permet de fixer le tempo musical. L'utilisation est la même que pour OCTAVE.

DUREE : Permet de régler la durée de chaque note. La syntaxe est toujours identique.

ATTAQUE : Permet de régler l'attaque de chaque note. L'utilisation est identique aux précédentes.

Signalons aussi l'existence de NOTE qui est une primitive totalement exploitée par le système et qui permet de définir DO.,RE. . . .

13.2 Le programme EXTENS

Ce programme se trouve sur l'écran 5 sur la cassette sous le nom EXTENS, juste aprés le programme MUSIC. Il contient diverses extensions du Forth dans les types de données (ici les tableaux à plusieurs dimensions, les chaînes et le traitement des chaînes).

13.2.1 Tableaux à une et deux dimensions

Les structures de données apportées par EXTENS sont les tableaux de nombres en simple précision et les tableaux d'octets (demi précision) à une ou deux dimensions.

L'instruction ARRAY1 permet de créer un tableau de nombres en simple précision à une dimension. La syntaxe est la suivante :

```
dimension ARRAY1 nom

Exemple: 5 ARRAY1 TEST (ENTREE)
crée un tableau de 6 éléments (de 0 à 5).
```

Une fois le tableau créé (attention, il n'est pas initialisé à 0), on peut stocker des nombres dedans par:


```
 n1 n2 TEST !
Alors n1 est stocké comme n2-ième élément de TEST..
On peut relire le n-ième élément par :
  n TEST  @
```

La procédure de création et d'utilisation est exactement la même pour CARRAY1 qui crée des tableaux d'octets à une dimension, si ce n'est qu'il faut utiliser C! au lieu de ! et C@ au lieu de @.

```
Exemple 7 CARRAY1 OCTETS crée le tableau OCTET à 8 éléments,
4 1 OCTETS C! stocke 4 comme premier élément,
3 OCTETS C@ lit le troisième élément.
```

L'instruction ARRAY2 permet de créer des tableaux de nombres en simple précision et à deux dimensions. La syntaxe est :

```
        dimension1 dimension2 ARRAY2 nom

Exemple: 4 4 ARRAY2 2DARRAY (ENTREE)
crée un tableau 5 5 de nom 2DARRAY. Le stockage s'effectue par:
         n1 n2 n3 2DARRAY !
qui stocke n1 en (n2,n3). La lecture s'effectue par:
         n1 n2 2DARRAY @
```

La procédure de création et d'utilisation est exactement la même pour CARRAY2 qui crée des tableaux d'octets à deux dimensions,si ce n'est qu'il faut utiliser C! au lieu de ! et C@ au lieu de @.

```
Exemple 7  2 CARRAY2 2OCTTS crée le tableau  8*3  2OCTTS.
4 1 1 OCTETS C! stocke 4 en (1,1),
3 1 OCTETS C@ lit l'élément stocké en (3,1).
```

13.2.2 Les chaînes et leur traitement

Contrairement au Basic qui est un langage interprété qui permet l'allocation dynamique des variables, le Forth est en partie compilé et ne permet donc (comme le Fortran ou le Pascal) que l'emploi de chaînes de longueurs fixes.

La création d'une chaîne s'effectue grâce à STRING. La syntaxe est la suivante:

```
  STRING nom chaîne"

Exemple:  STRING CHAIN LORICIELS" (ENTREE)
crée une chaine de nom CHAIN, contenant "LORICIELS".
  STRING STR FORTH" fait de même avec STR et "FORTH".
```

Il est possible de transférer une chaîne dans une autre grâce à l'instruction TFR. Si les chaînes sont de longueurs différentes, le Forth se charge de tronquer ce qu'il est nécessaire de tronquer.

```
  Syntaxe:        nom1 nom2 TFR

Exemple: STR CHAIN TFR (ENTREE)
transfère STR dans CHAIN.
```

L'affichage d'une chaîne une fois créée se fait directement par TYPE.

```
Exemple:    STR TYPE (ENTREE)
Affichage: FORTH
```

Une fois une chaîne créée, il est possible de faire changer son contenu par l'utilisateur grâce à la fonction INPUT qui s'emploie comme la fonction INPUT du Basic. Ainsi STR INPUT (ENTREE) permet de rentrer une nouvelle valeur de STR.

IL est aussi possible, comme pour le Basic, d'extraire des parties chaînes avec les instructions LEFT, RIGHT et MID.

L'instruction LEFT extrait les caractères de gauche d'une chaîne. Ainsi, STR 2 LEFT TYPE (ENTREE) donnera : 'FO'. La syntaxe est donc:

```
      nom de chaîne     n       LEFT
Comme tout autre chaîne, les résultats d'une extraction peuvent
être transférés(TFR), affichés(TYPE)....
```

L'instruction RIGHT à exactement la même syntaxe que LEFT, mais agit sur les caractères de droite.

```
 Exemple: STR 2 RIGHT TYPE (ENTREE)
donnera 'TH'.
```

L'instruction MID extrait n2 caractères à partir du n1-ème et correspond donc exactement à l'instruction MID$ du Basic.

```
      nom de chaînes   n1  n2  MID

Exemple: CHAIN 3 2 MID TYPE (ENTREE)
donnera à l'affichage 'RI' (2 caractères à partir du 3ème).
```

Il faut remarquer la puissance de ces instructions par rapport avec celles du Basic car elles permettent non seulement l'extraction mais aussi le transfert.

Il est ainsi possible de transférer les trois premières lettres d'une chaîne dans une autre chaîne, insérée à partir du troisième caractère. Voyons ceci sur quelques exemples (les deux chaînes STR et CHAIN étant définies comme précédemment):

Nous allons extraire les trois premières lettres de STR (donc les lettres 'FOR') et les insérer dans CHAIN à partir du cinquième pour finalement obtenir dans CHAIN la chaîne 'LORIFORLS'.

```
 STR  3 LEFT (ENTREE)     ;extraction des trois lettres
 CHAIN 5 3 MID (ENTREE)  ;calcul de la position d'insertion
 TFR (ENTREE)             ;transfert
Essayons maintenant:
 CHAIN TYPE (ENTREE)
On obtient bien 'LOROFORLS'.
```


14.0 L'EXEMPLE: UN JEU VIDÉO

Charles Moore, l'inventeur du langage Forth l'utilisait pour l'astronomie, pourquoi ne pas l'utiliser pour la réalisation d'un jeu vidéo ?

Vous trouverez un exemple de jeu que l'on peut réaliser avec le Forth. Il utilise largement le vocabulaire graphique et les sprites. De plus vous pourrez comparer la vitesse d'exécution avec celle du basic. Le programme se trouve sur la cassette aprés le fichier EXTENS. Pour le charger en mémoire faites 1 9 CLOAD GAME. Une fois le chargement fait , il faut compiler le programme en tapant 1 LOAD. Une fois la compilation finie, pour jouer faites JEU. Les touches A et Z permettent de se déplacer latéralement et la barre d'espace permet de tirer. Le jeu commence par un mode de présentation, pour jouer utilisez la barre d'espace et pour quitter le jeu tapez Q (pendant le mode démonstration). Afin que vous puissiez modifier le jeu ou vous en inspirer voici la définition des mots utilisés par le jeu.

14.1 Les sprites: screens 1 à 3.

```
VAIS : il s'agit de votre vaisseau (en bas de l'écran)
SOUC : c'est la soucoupe (volante)
TIRV : tir de votre vaisseau.
TIRS : tir de la soucoupe.
METEOR : dessin du météore
EXPLO : explosion de la soucoupe.
```

14.2 .Les variables :

```
XV : position de votre vaisseau en x
XS : position de la soucoupe en x
YS : position de la soucoupe en y
DXS : incrément sur la position de la soucoupe en x
DYS : incrément sur la position de la soucoupe en y
XM : position du météore en x
YM : position du météore en y
XTV : position du tir du vaisseau en x
YTV : position du tir du vaisseau en y
XTS : position du tir de la soucoupe en x
YTS : position du tir de la soucoupe en y
EXPS: compteur et flag utilisé pour l'explosion.
```

14.3 Déplacement et tests:

```
MOVV : déplace le vaisseau plus tests aux limites
MOVS : déplace la soucoupe et gère son explosion
TRAJ : générateur de position pour la soucoupe
       (fournit la prochaine position ). Ce mot est
       utilisé par MOVS
MOVTV : déplace le tir vaisseau et scrute le clavier.
MOVTS: déplace le tir de la soucoupe
FIRE? : prend la décision de tir pour la soucoupe
HITS : teste si la soucoupe est touchée (met à jour EXPLS)
HITV: teste si le vaisseau est touché par le tir de la soucoupe
      et dépose un flag sur la pile.
MOVM: déplace le météore et dépose sur la pile un flag indiquant
      si le vaisseau est touché.
```

14.4 Divers:

```
CSR: commutateur pour le clignotement du curseur
INIT:initialise les variables et les sprites.
DECOR: dessine le chef d'oeuvre que constitue le décor
DEMO: boucle de démonstration
MAIN: boucle de jeu principale
EXPLOV: explosion du vaisseau (aprés MAIN)
JEU: boucle BEGIN WHILE REPEAT qui constitue l'ensemble du jeu.
```


ANNEXE A. LES MESSAGES D'ERREURS

Les message d'erreurs sont affichés en anglais pour plus de concision.

EMPTY STACK : on a dépilé plus de nombre que la pile n'en contenait.

DICTIONARY FULL : indique que le dictionnaire est plein et que l'on ne peut plus compiler de mots. Lorsque l'on met au point un programme (avec l'éditeur en particulier), il faut oublier (FORGET) les anciennes versions des mots que l'on a créé afin de ne pas remplir inutilement le dictionnaire.

XXXX NOT FOUND : indique que le mot XXXX n'est pas trouvé dans le vocabulaire CONTEXT et dans le vocabulaire CURRENT, il peut cependant être dans un autre vocabulaire.

CONDITIONALS NOT PAIRED : des structures de contrôles sont mal imbriquées : Exemple BEGIN ... IF ... ELSE ... UNTIL

EXECUTION ONLY : il y a eu tentative d'utilisation en mode compilation d'un mot réservé au mode exécution (Exemple :)

COMPILATION ONLY : tentative d'exécution d'un mot réservé au mode compilation (Exemple IF BEGIN ...).

LOADING ONLY : utilisation de --> en dehors du mode load.

ISN'T UNIQUE : on vient de créer un mot qui existe déjà.

DEFINITION NOT FINISHED : on a commencé la définition d'un mot sans la terminer.

IN PROTECTED DICTIONARY : on a essayé de d'oublier (FORGET) un mot se trouvant dans le dictionnaire protégé (Forth de base). Cette protection évite de détruire par erreur une partie du vocabulaire de base. : I/O ERROR : il s'agit d'une erreur avec le lecteur enregistreur de cassettes (mauvais format etc)

ANNEXE B. LA CARTE MÉMOIRE:

Adresse hexadécimale	Contenu
$2100	Début du premier écran.
$444B	Fin du dernier écran.
$444C	Buffer de cassette.
$454C	Début du Forth.
.	
.	
HERE	Fin du Forth.
.	
.	
.	
.	Pile de données
$9DFF	Début de la pile données.
$9E00	Pile de retour et TIB.
$9FFF	Début de la pile retour.

INDEX

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

NOTES

53, rue de Paris 92100 BOULOGNE – Tél. (1) 825.11.33 +