TO7 MO5

LE MEMENTO FORTH

TO 7/MO 5

S.E.F.I.

— Si vous abordez le langage FORTH pour la première fois, reportez-vous au manuel "Initiation au FORTH TO7/MO5" (CEDIC/TO TEK INTERNATIONAL).

— Pour approfondir les instructions FORTH, consultez le "Manuel de Référence FORTH TO7/MO5" (CEDIC/TO TEK INTERNATIONAL).

— Ce mémento ainsi que les deux manuels contiennent les mots du D.O.S. FORTH.

MISE EN ROUTE DES ÉQUIPEMENTS

1) TO7/TO7-70

a) Insérez la cartouche programme FORTH dans le lecteur de cartouche de votre unité centrale ;

b) Mettez sous tension, dans l'ordre : les périphériques, le téléviseur ou le moniteur puis l'ordinateur.

c) Le menu apparait sur l'écran : choisissez FORTH ou réglez au préalable votre crayon optique.

2) MO 5

a) et b) : IDEM TO7/TO7-70

c) Pour régler votre crayon optique : tapez "TUNE", puis pointez votre crayon sur la mire.

N.B. : Si vous souhaitez utiliser une disquette reportez-vous au paragraphe 10 (stockage de masse).

PRINCIPAUX MOTS DU FORTH

1 - Manipulation et affichage des nombres de la pile.

0	**(...n)** laisse 0 sur la pile
1	**(...n)** laisse 1 sur la pile
2	**(...n)** laisse 2 sur la pile
3	**(...n)** laisse 3 sur la pile
>R	**(n...)** transfère n sur la pile des retours
?DUP	**(n...(n) n)** duplique n s'il n'est pas nul
DDROP	**(d...)** supprime d du sommet de la pile
DDUP	**(d...d d)** duplique d
DOVER	**(d1 d2...d1 d2 d1)** copie au sommet de la pile le 2e nombre double longueur
DROP	**(n...)** supprime n du sommet de la pile
DROT	**(d1 d2 d3 ...d2 d3 d1)** permutation circulaire sur d1, d2, d3
DSWAP	**(d1 d2 ...d2 d1)** échange d1 et d2 (nombres double longueur)
DJP	**(n...n n)** duplique n
OVER	**(n1 n2 ...n1 n2 n1)** recopie le 2e nombre au sommet de la pile
PICK	**(n1...n2)** recopie le n1-ième nombre au sommet de la pile (n1 non compris)
R>	**(...n)** transfère sur la pile le sommet de la pile des retours. Inverse de >R
R@	**(...n)** copie sur la pile le sommet de la pile des retours
ROLL	**(n1 n2...n_n n...n2...n_n n1)** permutation circulaire sur n éléments
ROT	**(n1 n2 n3...n2 n3 n1)** permutation circulaire sur 3 éléments
SWAP	**(n1 n2...n2 n1)** échange des 2 éléments du sommet de la pile
SWAPDROP	**(n1 n2...n2)** supprime le 2e élément à partir du sommet de la pile

---

><	**(n1....n2)** échange octet haut et octet bas de n1
DEPTH	**(....n)** n est le nombre de nombres sur la pile (n non compris)
HILO	**(n...n1 n2)** n1 = octet bas de n, n2 = octet haut de n
R0	**(...adr)** adresse du début de la pile des retours
RP!	**(...)** initialise la pile des retours
RP0	**(...adr)** variable contenant l'adresse initiale de la pile.des retours (pour reconfiguration)
RP@	**(...adr)** valeur du pointeur de la pile des retours (sommet de la pile des retours)
S0	**(...adr)** adresse du début de la pile (des données)
SP!	**(...)** initialise la pile des données
SP0	**(...adr)** variable contenant l'adresse initiale de la pile des données (pour reconfiguration)
SP@	**(...adr)** adresse du sommet de la pile
SQUISH	**(n1 n2 ...n)** n a pour octet haut l'octet bas de n2, et pour octet bas l'octet bas de n1

Conventions de notation de la pile (pour tout le memento).

n	nombre entier signé (-32768 ≤ n ≤ 32767) (en 2 octets)
un	nombre entier non signé (0 ≤ un ≤ 65535) (en 2 octets)
d	nombre double longueur signé (-2147483648 ≤ d ≤ 2147483647) (en 4 octets)
ud	nombre double longueur non signé (0 ≤ ud ≤ 4294967295) (en 4 octets)
adr	adresse (nombre entier en 2 octets)
str	paramètre pour chaîne de caractère (adresse et longueur) = 2 nombres
condition (ou cond)	nombre entier 0 signifie "faux", tout autre nombre signifie "vrai"
c	caractère (en fait, nombre de 0 à 255, représentant son code ASCII étendu)
pile	= pile des données (sauf indication contraire : pile des retours)

2 - Calculs arithmétiques et logiques.

*****	**(n1 n2 ...n)** n est le produit de n1 et n2
***/**	**(n1 n2 n3 ...n)** n est le quotient par n3 du produit n1 x n2. Le produit est un nombre double longueur
***/MOD**	**(n1 n2 n3 ... n4 n5)** n5 est le quotient par n3 de n1n2, n4 est le reste
+	**(n1 n2 ... n)** n est la somme de n1 et n2
−	**(n1 n2 ...n)** n = n1 − n2
/	**(n1 n2 ...n)** n est le quotient entier de n1 par n2
/MOD	**(n1 n2 ...n3 n4)** n4 est le quotient de n1 par n2, n3 est le reste de la division
0<	**(n ...cond)** cond vaut 1 si n < 0, 0 sinon
0=	**(n ...cond)** cond vaut 1 si n est nul, 0 sinon
0>	**(n ...cond)** cond vaut 1 si n > 0, 0 sinon
1+	**(n1 ...n2)** n2 = n1 + 1
1−	**(n1 ...n2)** n2 = n1 − 1
2*	**(n1 ...n2)** n2 = 2 n1
2+	**(n1 ...n2)** n2 = n1 + 2
2−	**(n1 ...n2)** n2 = n1 − 2
2/	**(n1 ...n2)** n2 est le quotient entier de n1 par 2
<	**(n1 n2 ...cond)** cond vaut 1 si n1 < n2, 0 sinon
=	**(n1 n2 ...cond)** cond vaut 1 si n1 = n2, 0 sinon
>	**(n1 n2 ...cond)** cond vaut 1 si n1 > n2, 0 sinon
ABS	**(n1 ...n2)** n2 est la valeur absolue de n1
AND	**(n1 n2 ...n3)** réalise le ET logique (bit à bit) de n1 et n2
D*	**(d1 d2 ...d3)** d3 est le produit de d1 et d2
D+	**(d1 d2 ...d3)** d3 est la somme de d1 et d2
D−	**(d1 d2 ...d3)** d3 est la différence d1 − d2
D0=	**(d ...cond)** cond vaut 1 si d = 0, 0 sinon
D<	**(d1 d2 ...cond)** cond vaut 1 si d1< d2, 0 sinon

D= **(d1 d2...cond)** cond vaut 1 si d1 = d2, 0 sinon
DABS **(d1...d2)** d2 est la valeur absolue de d1
DMAX **(d1 d2...d3)** d3 est le plus grand des deux nombres d1 et d2
DMIN **(d1 d2...d3)** d3 est le plus petit des deux nombres d1 et d2
DN* **(d1 n...d2)** d2 est le produit du nombre double longueur d1 par le nombre n
DNEGATE **(d1...d2)** d2 l'opposé de d1
DU< **(ud1 ud2...cond)** cond vaut 1 si ud1 < ud2, 0 sinon
M* **(n1 n2...d)** d est le produit (double longueur) de n1 et n2
M/ **(d n...n1 n2)** n2 est le quotient entier de d par n, n1 est le reste de la division
M/MOD **(ud1 un1...un2 ud2)** ud2 est le quotient de ud1 par un1, un2 est le reste
MAX **(n1 n2...n)** n est le plus grand des deux nombres n1 et n2
MIN **(n1 n2...n)** n est le plus petit des deux nombres n1 et n2
MOD **(n1 n2...n)** (modulo) n est le reste de la division de n1 par n2
NEGATE **(n1...n2)** n2 est l'opposé de n1
NOT **(n1...n2)** n2 vaut 1 si n1 = 0, et 0 si n1 est non nul (synonyme de 0 =)
OR **(n1 n2...n3)** réalise le OU logique bit à bit de n1 et n2
S->D **(n...d)** transforme n en nombre double longueur
U* **(un1 un2...ud)** produit non signé de 2 nombres 16 bits. Résultat en 32 bits.
U/MOD **(ud1 un2...un3 un4)** division non signée de ud1 par un2, un3 et le reste ; un4 est le quotient
U< **(un1 un2...cond)** Cond vaut 1 si un1 < un2 ; sinon 0
UDN* **(ud1 un...ud2)** produit non signé d'un nombre 32 bits par un nombre 16 bits

---

+- **(n1 n2...n)** n vaut n1 si n2 > 0, et —n1 si n2 < 0
D+- **(d1 d2...d)** analogue à +-, pour les nombres double longueur
XOR **(n1 n2...n)** réalise le OU Exclusif bit à bit de n1 et n2

3 - Structures de contrôle.

BEGIN **(...)** répétition non indicée. Se présente sous l'une des 3 formes :

1) **BEGIN actions AGAIN**
actions est répété indéfiniment

2) **BEGIN actions1 (condition) WHILE actions2 REPEAT**
actions1 s'exécute une première fois (laisse une condition sur la pile)
puis, actions2 et actions1 se répètent **tant que** la condition est "vrai"

3) **BEGIN actions (condition) UNTIL**
répétition de actions **jusqu'à** ce que condition soit "vrai"

CASE **(n...)** structure de sélection ; s'utilise sous la forme suivante :

```
n CASE
  n1 OF  actions1  ENDOF   si n = n1, actions1 s'exécute
  n2 OF  actions2  ENDOF   sinon, si n = n2, actions2 s'exécute
   ...                     sinon, ...
```

nx OF actionsx ENDOF
autres actions si n n'est égal ni à n1, ni à n2, . . ., ni à nx

END CASE **autres actions** (facultatif) s'éxécute.

DO (n1 n2 . . .) répétition indicée. n2 = indice de début ; 3 formes possibles

1) n1 n2 DO actions LOOP
l'indice est augmenté de 1. La répétition s'arrête quand l'indice est égal à n1

2) n1 n2 DO actions n +LOOP
l'indice est augmenté de n. La répétition s'arrête pour
indice ≥ n1 si n> 0
indice < n1 si n< 0

3) n1 n2 DO actions un /LOOP
l'indice est non signé et augmente de **un** à chaque tour (autorise des boucles de 0 à 65535)

GODO **(n . . .)** structure de sélection calculée.
n GODO mot1 mot2 mot3 . . . motx THEN
exécute le mot motn (si n< 1, exécute mot1. Si n > x, exécute motx)

IF **(cond . . .)** structure conditionnelle

1) condition IF actions THEN
si la condition est réalisée, **actions** s'exécute

2) condition IF actions1 ELSE actions2 THEN
si la condition est réalisée, **actions1** s'exécute
sinon, c'est **actions2** qui est exécutée

EXIT **(. . .)** force la fin de l'exécution d'un mot

I **(. . .n)** dépose sur la pile l'indice de la boucle la plus interne (dans une boucle DO)

J **(. . .n)** dans les cas de boucles DO imbriquées, donne l'indice de l'avant-dernière boucle ouverte

LEAVE **(. . .)** force la fin d'une boucle DO (au prochain passage par LOOP, +LOOP ou/LOOP)

Tous ces mots, sauf I et J, ne peuvent être utilisés qu'à l'intérieur d'une définition par : . . .;
Actions désigne une suite de mots Forth.

4 Définition de nouveaux mots. Compilation et interprétation.

, **(n . . .)** compile le nombre n à la fin du dictionnaire

: **(. . .)** s'utilise sous la forme : XX . . . ; pour définir le mot XX

; **(. . .)** termine une définition commencée par :

ARRAY **(n . . .)** n ARRAY XX crée un tableau XX pouvant contenir n nombres

ARRAY$ **(n1 n2 . . .)** n1 n2 ARRAYS XX crée un tableau XX pouvant contenir n2 chaînes de longueur maximum n1

C, **(n . . .)** compile l'octet bas de n à la fin du dictionnaire

CARRAY **(n . . .)** n CARRAY XX crée un tableau XX pouvant contenir n octets

CONSTANT **(n . . .)** n CONSTANT XX crée une constante XX de valeur n

CREATE **(. . .)** CREATE XX crée une en-tête dans le dictionnaire pour le mot XX

CTABLE **(. . .)** CTABLE XX crée une table d'octets (A remplir, par exemple par C,)

DCONSTANT **(d . . .)** d DCONSTANT XX crée une constante double longueur XX de valeur d

DEFINITIONS **(. . .)** positionne le vocabulaire actuellement sélectionné pour les prochaines définitions

DOES> **(. . .adr)** utilisé sous la forme : XX . . . CREATE . . . DOES> . . . ; pour créer un mot de définition XX

DVARIABLE **(. . .)** DVARIABLE XX crée une variable numérique double longueur. (Initialisée à 0)

EDITOR **(. . .)** sélectionne le vocabulaire de l'éditeur

EXECUTE **(adr . . .)** exécute le mot dont le cfa est sur la pile

FORTH **(. . .)** sélectionne le vocabulaire Forth

IMMEDIATE **(. . .)** rend immédiat le dernier mot créé au dictionnaire

INTERPRET **(. . .)** interprète le flot d'entrée (exécute ou compile les mots)

STRING **(n . . .)** n STRING XX crée une variable chaîne pouvant contenir au maximum n caractères

TABLE **(. . .)** TABLE XX crée une table de nombres (à remplir, par exemple par ,)

VARIABLE **(. . .)** VARIABLE XX crée une variable de nom XX, initialisée à 0

VOCABULARY **(. . .)** VOCABULARY XX crée un nouveau vocabulaire de nom XX. (Suivre par XX DEFINITIONS)

[**(. . .)** passe en mode exécution

[COMPILE] **(. . .)** dans une définition, [COMPILE] XX force la compilation de XX s'il est immédiat

] **(. . .)** positionne le mode compilation

;S **(. . .)** stop l'interprétation d'un écran de l'éditeur

ALLOT **(n . . .)** alloue n octets à la fin du dictionnaire

COMPILE **(. . .)** dans une définition, compile la valeur 16 bits qui suit dans le dictionnaire

CONTEXT **(. . . adr)** variable mémorisant le vocabulaire actuellement sélectionné

CSP **(. . . adr)** variable système

CURRENT **(. . . adr)** variable spécifiant le vocabulaire qui contiendra les prochaines définitions

STATE **(. . . adr)** variable système spécifiant le mode (compilation ou interprétation)

VOC-LINK **(. . . adr)** variable pointant le dernier vocabulaire créé

5 Caractères et chaînes de caractères. Conversion entre données numériques et alphanumériques.

"	**(...str)** utilisé sous la forme " xxxx... " pour définir une chaîne temporaire xxxx...
" "	**(...str)** donne les paramètres de la chaîne nulle (de longueur 0)
$!	**(str1 str2 ...)** copie la chaîne (str1) dans la chaîne (str2)
#	**(ud1 ...ud2)** génère le code ASCII suivant de la conversion commencée par <#
#>	**(ud ... str)** termine une conversion commencée par <#
#S	**(ud ...00)** convertit ud en caractères ASCII à l'intérieur de <##>
$+	**(str1 str2 ...)** ajoute la chaîne (str1) à la fin de (str2)
$<	**(str1 str2 ...cond)** cond vaut 1 si (str1) < (str2) (dans l'ordre lexicographique), 0 sinon
$=	**(str1 str2 ... cond)** cond vaut 1 si (str1) = (str2), 0 sinon
−TRAILING	**(str1 ...str2)** modifie la longueur de (str1) de manière à ignorer les espaces en fin de chaîne
<#	**(ud ... ud)** initialise le processus de conversion d'un nombre en chaîne de caractères
ARRAY$	**(n1 n2 ...)** n1 n2 ARRAY $ XX crée un tableau XX pouvant contenir n2 chaînes de longueur maximum n1
ASC	**(str ... n)** donne le code ASCII du premier caractère de (str)
CHARTAB	**(n ...)** n CHARTAB XX définit une table XX pouvant contenir n caractères graphiques
CHR$	**(n ... str)** convertit le code ASCII n en chaîne temporaire de 1 caractère
DEFGR	**(n1 n2 ... n8 n)** crée le caractère graphique de code ASCII n
HOLD	**(c ...)** insère le caractère c dans la chaîne obtenue par une conversion du type <#...#>
INSTR	**(str1 str2 ... n),** donne la position de la première occurence de (str1) dans (str2). (0 si non trouvé)
LEFT$	**(str1 n ... str2)** donne la sous-chaîne str2 constituée par les n caractères de gauche de (str1)
LEN	**(str ...n)** donne la longueur n de la chaîne (str)
MLEN	**(str ... n)** donne la longueur maximum de la chaîne (str)
MID$	**(str1 n1 n2 ... str2)** donne la sous-chaîne (str2) constituée par n2 caractères de (str1) à partir du numéro n1
RIGHT$	**(str1 n ... str2)** donne la sous-chaîne (str2) constituée par les n caractères de droite de (str1)
SIGN	**(n ...)** insère le signe "—" si n<0 dans une conversion du type <#..#>
STR$	**(d ... str)** convertit le nombre double longueur d en chaîne de caractères
STRING	**(n ...)** n STRING XX définit une variable XX pouvant contenir une chaîne de n caractères maximum

SUB$! **(str1 str2 ...)** transfère (str1) dans la sous-chaîne (str2)
USTR$ **(ud ... st2)** convertit le nombre double longueur ud (non signé) en chaîne de caractères
VAL **(str ...d)** convertit la chaîne (str) en nombre double longueur

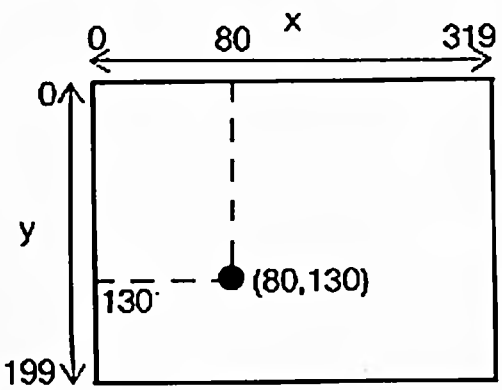
---

COUNT **(adr ... str)** donne les paramètres de la chaîne qui commence en adr. (L'octet en adr doit être l'octet de longueur de la chaîne)
HLD **(... adr)** contient l'adresse du dernier caractère convertit par #ou#S
NUMBER **(adr ... d)** convertit en nombre double longueur la chaîne qui commence en adr
PAD **(... adr)** donne l'adresse du début de la zone où sont rangées les chaînes temporaires
TEXT **(c ...)** dépose dans PAD la chaîne du flot d'entrée délimitée par c
WORD **(c ... adr)** donne le prochain mot du flot d'entrée délimité par le caractère c

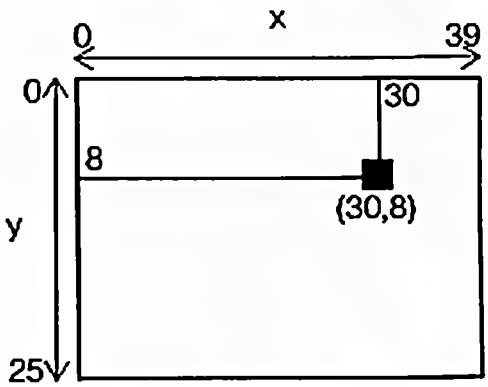
6 Gestion de l'écran (texte et graphique).

BLACK **(... n)** constante qui donne le code de la couleur noire (0)
BLUE **(... n)** constante qui donne le code de la couleur bleue (4)
BOX **(n1 n2 n3 n4 ...)** trace un rectangle de sommets (n1, n2) et (n3, n4)
BOXF **(n1 n2 n3 n4 ...)** trace un rectangle plein de sommets (n1, n2) et (n3, n4)
BOXFTO **(n1 n2 ...)** trace un rectangle plein de sommets (n1, n2) et le dernier point antérieurement tracé
BOXPEN **(n1 n2 n3 n4 n5 ...)** définit la zone de saisie numéro n5 passera le crayon optique
BOXTO **(n1 n2 ...)** trace un rectangle de sommets (n1, n2) et le dernier point antérieurement tracé
CLS **(...)** efface l'écran et replace le curseur en haut à gauche
COLOR **(n ...)** positionne la couleur pour le texte
COLOROFF **(...)** lie les couleurs à la position-écran. Les attributions de couleurs sont inactives
COLORON **(...)** libère les couleurs
CSRLIN **(...)** donne le numéro de la ligne où est le curseur (mode texte)
CSROFF **(...)** rend le curseur invisible
CSRON **(...)** rend le curseur visible
CYAN **(... n)** constante qui donne le code de la couleur cyan (6)
DARK attributs de couleur saturés (*)
FRAME **(n ...)** positionne la couleur du pourtour de l'écran
GCOLOR **(n ...)** positionne la couleur en mode graphique
GLOBAL **(...)** les attributions ultérieures de couleur sont valables pour tout l'écran
GREEN **(... n)** constante qui donne le code de la couleur verte (2)
HOME **(...)** place le curseur en haut à gauche de l'écran
INITSCR **(...)** initialise les paramètres écran (taille, curseur, défilement, fenêtre)

INK	**(...)** les attributions de couleurs ultérieures agissent sur la forme ou les caractères
INVCOLOR	**(...)** inverse les couleurs du fond et des points (ou des caractères)
LINE	**(n1 n2 n3 n4 ...)** trace une ligne droite joignant les points (n1, n2) et (n3, n4)
LINETO	**(n1 n2 ...)** trace une ligne droite joignant le point (n1. n2) au dernier point tracé antérieurement
LOCAL	**(...)** les attributions de couleurs ultérieures seront locales (pas plein-écran)
LOCATE	**(n1 n2 ...)** place le curseur en colonne n1, ligne n2
MAGENTA	**(...n)** constante donnant le code de la couleur violette (5)
MASK	**(...)** sélectionne le mode masqué
NOMASK	**(...)** desélectionne le mode masqué (si LOCAL), ou démasque (si GLOBAL)
PAGE	**(...)** supprime le défilement (mode " page ")
PALE	attributs de couleurs pastels (*)
PAPER	**(...)** les attributions de couleurs postérieures agissent sur le fond
PEN	**(n1 n2 n3...)** définit la zone de saisie n3 par le crayon optique par la zone du caractère en (n1, n2) (position texte)
POINT	**(n1 n2 ...n)** donne la couleur du point de coodonnées (n1, n2) (si n < 0, couleur fond - n-1)
PSET	**(n1, n2 ...)** affiche un point en (n1, n2)
RED	**(...n)** constante donnant le code de la couleur rouge (1)
SCREEN	**(n1 n2 ...c)** donne le code ASCII en position (n1, n2) de l'écran texte
SCROLL	**(n ...)** règle la vitesse du défilement (0 = rapide, 1 = lent)
SIZE	**(n1 n2 ...)** définit la taille des caractères (voir **)
WHITE	**(...n)** constantes qui donne le code de la couleur blanche (7)
WINDOW	**(n1 n2 ...)** la fenêtre de travail comporte les lignes n1 à n2
YELLOW	**(...n)** constante qui donne le code de la couleur jaune (3)



Mode graphique



Mode texte

** pour TO7-70 et MO5 seulement*

**** attributs de SIZE**

0	0	Taille normale
1	0	Double hauteur
0	1	Double largeur
1	1	Double taille

7 - Musique.

.#	(...) altère (augmente d'un demi-ton) la dernière note compilée dans la table en cours
.b	(...) altère (diminue d'un demi-ton) la dernière note compilée dans la table en cours
DUR	(n ...) sélectionne la durée n en mode immédiat pour les notes à jouer (voir*)
INITMUS	(...) réinitialise la table musicale sélectionnée
MUSTAB	(n ...) n MUSTAB XX crée une table musicale XX de n octects
NOTE	(n ...) joue la note de code n (voir ci-dessous **)
OCT	(n ...) sélectionne l'octave n en mode immédiat pour les notes à jouer
PITCH	(n ...) sélectionne l'enveloppe n pour les notes à jouer
PLAY	(...) joue la table musicale sélectionnée
TEMP	(n ...) sélectionne le tempo n pour les notes à jouer
do	(...) compile un do dans la table sélectionnée
do +	(...) compile un do de l'octave supérieure dans la table sélectionnée
dur	(n ...) sélectionne la durée en mode compilé
fa	(...) compile un fa dans la table sélectionnée
la	(...) compile un la dans la table sélectionnée
mi	(...) compile mi dans la table sélectionnée
oct	(n ...) sélectionne l'octave en mode compilé
pause	(...) compile un silence (pause, 1/2 pause, soupir, ... suivant la durée) dans la table sélectionnée
pitch	(n ...) sélectionne l'enveloppe en mode compilé
re	(...) compile un ré dans la table en cours de définition
si	(...) compile un si dans la table en cours de définition
sol	(...) compile un sol dans la table en cours dedéfinition
temp	(n ...) sélectionne le tempo en mode compilé

***Paramètres musicaux:**

	Par défaut
Octave: 1 à 5	4
Tempo: 1 à 255	5
Durée: 1 à 96 (noire 24)	24
Enveloppe: 0 à 255	0

****Pour les notes en mode direct :**

48	silence
49	do
50	do +
51	re
52	mi b
53	mi
54	fa
55	fa #
56	sol
57	sol #
58	la
59	si b
60	si
61	do (octave supérieure)

8 - Mots concernants le dictionnaire et la mémoire.

!	**(n adr ...)** range le nombre n à l'adresse adr (en 2 octets)
'	**(... adr)** ' XX laisse sur la pile l'adresse du champ paramètre (pfa) de XX
+!	**(n adr ...)** ajoute n au nombre rangé à l'adresse adr.
1 +!	**(adr ...)** incrémente le nombre rangé aux adresses (adr, adr +1)
<CMOVE	**(adr1 adr2 n ...)** déplace n octets de adr1 à adr2 en commençant par la fin
?	**(adr ...)** affiche la valeur du nombre rangé aux adresses (adr, adr+1)
@	**(adr ... n)** laisse sur la pile la valeur du nombre rangé aux adresses (adr, adr+1)
BANK	**(n ...)** permet de sélectionner la banque numéro n (0≤ n ≤ J) de l'extension mémoire (pour TO7-70 seulement)
BLANKS	**(adr n ...)** met à blanc (remplit d'espaces) la zone de n octets commençant à l'adresse adr
C!	**(n adr ...)** range l'octet bas de n à l'adresse adr
C@	**(adr ... n)** laisse sur la pile la valeur n de l'octet à l'adresse adr
CMOVE	**(adr1 adr2 n)** déplace n octets de adr1 à adr2 (en commençant par le début)
D!	**(d adr ...)** range le nombre d à l'adresse adr (en 4 octets)
D@	**(adr ... d)** donne la valeur d du nombre double longueur commençant à l'adresse adr
DUMP	**(adr n ...)** affiche les contenus (hexa et ASCII) de n octets à partir de l'adresse adr
EDITOR	**(...)** sélectionne le vocabulaire éditeur (voir §4)
ERASE	**(adr n ...)** met à zéro (remplit de caractères de code ASCII 0) la zone de n octets commençant à l'adresse adr
FENCE	**(... adr)** variable contenant l'adresse sous laquelle le dictionnaire est protégé
FORTH	**(...)** sélectionne le vocabulaire Forth
FILL	**(adr n c ...)** remplit avec le caractère c la zone de n octets à partir de l'adresse adr
FIND	**(... adr)** FIND XX donne le cfa de XX, ou 0 si XX n'est pas trouvé
FORGET	**(...)** FORGET XX supprime du dictionnaire le mot XX et tous les mots qui suivent
MOVE	**(adr1 adr2 n ...)** déplace n nombres (2n octets) de l'adresse adr1 à adr2
VLIST	**(...)** affiche la liste des noms du dictionnaire

?FIND	**(... adr)** ?FIND XX donne le cfa de XX, et un message d'erreur si XX n'est pas trouvé
CFA	**(adr1 ... adr2)** donne le cfa d'un mot à partir de son pfa
CONTEXT	**(... adr)** variable mémorisant le vocabulaire sélectionné
DP	**(... adr)** contient l'adresse du premier octet après la fin du dictionnaire
FIRST	**(... adr)** variable donnant l'adresse du premier buffer
HERE	**(... adr)** donne la prochaine adresse disponible dans le dictionnaire
ID.	**(adr ...)** affiche le nom d'un mot à partir de son NFA
LATEST	**(... adr)** donne l'adresse du dernier mot compilé dans le vocabulaire positionné
LFA	**(adr1 ... adr2)** donne le lfa d'un mot connaissant son pfa
LIMIT	**(... adr)** donne l'adresse du dernier octet disponible pour un buffer (en fonction de la mémoire)

NFA **(adr1 ... adr2)** donne le nfa d'un mot de son pfa
PFA **(adr1 ... adr2)** donne le pfa d'un mot à partir de son nfa
TOGGLE **(adr c ...)** modifie le caractère en adr en réalisant son "ou exclusif" avec c
TRAVERSE **(adr1 n ... adr2)** donne l'adresse d'une extrémité du nom en fonction de l'autre. n vaut 1 ou – 1 pour préciser le sens du déplacement
WIDTH **(... adr)** variable contenant le nombre de caractères du nom qui sont pris en compte pour les définitions ultérieures

---

9 - Entrées-Sorties (sauf cassettes ou disquettes): Clavier. Crayon optique. Manettes de jeux. Imprimante.

. **(n ...)** affiche le nombre n sur le périphérique de sortie en cours
." **(...)** utilisé sous la forme ." xxxx" pour afficher la chaîne xxxx
.R **(n1 n2 ...)** affiche le nombre n1, cadré à droite dans une zone de n2 espaces
?STOP **(... cond)** arrête l'exécution si on a tapé une touche. Si la prochaine touche tapée est la touche STOP, cond est "vrai" sinon cond est "faux"
?TERMINAL **(... cond)** cond est vrai si on a tapé une touche
BASE **(... adr)** variable contenant la base utilisée pour les conversions numériques
BL **(... n)** constante donnant le code du caractère "espace" (32)
BS **(... n)** constante donnant le code du caractère "back-space" (8)
BUTTON **(n ... cond)** cond est vrai si le bouton de la manette de jeux n° n est enfoncé
CONSOLE **(...)** retourne au périphérique de sortie positionné avant l'exécution de PRINTER, en général l'écran
CR **(...)** envoi un "passage à la ligne" au périphérique de sortie CR + LF
D. **(d ...)** affiche le nombre d sur le périphérique de sortie en cours
D.R **(d n ...)** affiche le nombre d, cadré à droite dans une zone de n espaces
DECIMAL **(...)** positionne la base à dix
DPL **(... adr)** variable contenant la position du point décimal lors de l'entrée d'un nombre 32 bits en mode direct
EMIT **(c ...)** envoie le caractère c au périphérique de sortie
HEX **(...)** positionne la base à seize (système hexadécimal)
INKEY **(... c)** lecture non bloquante du clavier
INPEN **(... n1 n2)** lecture non bloquante des coordonnées du point visé par le crayon optique
INPUT **(... d)** lecture d'un nombre double longueur sur le périphérique en entrée
INPUT$ **(... str)** lecture d'une chaîne de caractères (80 car maxi)
INPUTPEN **(... n1 n2)** lecture bloquante des coordonnées du point visé par le crayon optique
KEY **(... c)** lecture d'un caractère sur le périphérique d'entrée
LINPUT **(n ... d)** lecture d'un nombre (double longueur) pouvant comporter au maximum n chiffres
LINPUT$ **(n ... str)** lecture d'une chaîne pouvant comporter au maximum n caractères

ONPEN **(...n)** donne le numéro de la zone (définie par PEN ou BOXPEN) pointée par le crayon optique

PTRIG **(...cond)** cond est "vrai" si le bouton poussoir à l'extrémité du crayon optique est enfoncé

PRINTER **(...)** sélectionne l'imprimante comme périphérique de sortie

SPACE **(...)** affiche un espace sur le périphérique de sortie

SPACES **(n ...)** affiche n espaces sur le périphérique de sortie

STICK **(n1 ... n2)** donne la position n2 de la manette de jeux numéro n1

TYPE **(str ...)** affiche la chaîne de caractères (str) sur le périphérique de sortie

U. **(un ...)** affichage d'un nombre interprété comme non signé

U.R **(un n ...)** affichage non signé de un cadré à droite dans une zone de n espaces

UD. **(ud ...)** affichage d'un nombre double longueur son signé

UD.R **(ud n ...)** affichage d'un nombre double longueur non signé, cadré à droite dans une zone de n espaces

---

>IN **(...n)** donne l'offset (adresse relative) du caractère traité dans le flot d'entrée

>OUT **(...n)** contient un nombre incrementé par EMIT, pour contrôler le formattage en sortie

EXPECT **(adr n ...)** lit n caractères sur le périphérique en entrée, et les transfère à partir de l'adresse adr

IN? **(...adr)** adresse du prochain caractère du flot d'entrée

QUERY **(...)** lit 80 caractères en entrée, et les dépose dans le TIB

TIB ..... **(....adr)** variable contenant l'adresse du TIB (Buffer d'entrée du Terminal)

SCRPRT **(...)** réalise une copie de l'écran graphique sur l'imprimante thermique

---

10 - Stockage de masse. Cassettes. Disquettes

Disquettes

Pour utiliser les disquettes, vous devez d'abord charger le DOS FORTH avant de commencer à travailler :

- pour le MO5, après avoir démarré FORTH, mettre la disquette DOS dans le lecteur 0 et taper DOS (attention : DOS réinitialise FORTH) ;
- pour le TO7 ou TO7-70, mettre la disquette DOS dans le lecteur 0 avant de choisir [1] pour FORTH dans le menu principal.

BACKUP **(n1 n2 ...)** copie physique du contenu de la disquette dans l'unité n1 sur celle de l'unité n2.

=BUF **(...n)** n est le nombre de buffers alloués

B/BUF **(...n)** constante laissant le nombre d'octets par block (1024)

BLOCK **(n ...adr)** laisse l'adresse en mémoire du block n. Si le block n'est pas en mémoire, il est lu sur le stockage de masse

BUFFERS **(n ...)** sélectionne le nombre de buffers utilisables

CASS **(...)** positionne la cassette comme unité de stockage de masse.

COPY-BLOCKS **(n1 n2 n3)** copie les blocks logiques n1 à n2 d'une disquette, en les renumérotant à partir de n3. A l'exécution, il est demandé le lecteur source et le lecteur destination.

DIR **(n ...)** affiche le catalogue de la disquette dans l'unité n.

DISK **(...)** positionne l'unité de disquette comme unité de stockage de masse.

DOS **(...)** charge le DOS FORTH et réinitialise FORTH.

DRIVE **(n ...)** positionne l'unité de disquette n comme unité courante.
EMPTY-BUFFERS **(...)** vide les blocks en mémoire
FORMAT **(n ...)** formatte la disquette dans n.
KILL **(n ...)** détruit le block logique n dans l'unité de disquette courante.
LOAD **(n ...)** lit éventuellement le block n et l'interprète
MAXBUF **(...n)** donne le nombre maximum de buffers utilisables compte tenu de la mémoire disponible
RENUM **(n1 n2 n3 ...)** numérote les blocks logiques n1 à n2 de l'unité de disquette courante comme étant des blocks dont le premier est numéroté n3.
SAVE-BUFFERS **(...)** sauvegarde les blocks modifiés
SKIP **(...)** positionne la cassette après le premier fichier trouvé
SKIPB **(n ...)** positionne la cassette après le block n
THRU **(n1 n2 ...)** lit les blocks de n1 à n2 (s'ils ne sont pas en mémoire) et les interprète
UPDATE **(...)** marque le dernier block référencé comme modifié

BBLOCK **(n ... adr)** laisse sur la pile l'adresse du block n. Si le block n'est pas en mémoire, il est créé (à blanc)
BLK **(... adr)** variable contenant le numéro du block en cours d'interprétation. (Si n = o, interprète le périphérique en entrée)
BUFFER **(n ... adr)** assigne un buffer (adresse adr) au block n
D-ERR **(... adr)** variable contenant le code erreur après une opération d'entrée-sortie (*), variable sur un octet
DMA **(... adr)** variable contenant l'adresse du buffer d'entrées-sorties cassettes ou disquettes
DNAM **(... adr)** variable contenant l'adresse du nom de fichier utilisé
EOF **(... cond)** indique "vrai" pour une fin de fichier en lecture
FASC! **(n ...)** positionne le type de données du fichier en cours (0 pour binaire, 255 pour ASCII)
FNAM! **(str ...)** positionne la chaîne (str) comme nom de fichier
FNB! **(n1 n2 ...)** modifie le nom de fichier en incorporant le nombre n1 en position n2
FTYP! **(n ...)** positionne le type du fichier ($0 \leqslant n \leqslant 3$)
GETC **(... c)** lit le caractère c sur le stockage de masse
MOTOROFF **(...)** arrête le déroulement de la cassette
MOTORON **(...)** met en route la cassette
PREV **(... adr)** variable contenant l'adresse du buffer du block le plus récemment référencé
PUTC **(c ...)** écrit le caractère c sur le stockage de masse
R/W **(adr n1 n2 ...)** lit (si n2 = 1) ou écrit (si n2 = 0) le block n1 du stockage de masse, à l'adresse adr en mémoire
RS-B permet de fixer le nombre de bits de la transmission série RS-232 du port de communication. (**)
RS-S permet de fixer la vitesse
USE **(... adr)** variable contenant l'adresse du prochain buffer à utiliser

* **Codes erreur de D-ERR**

0 - L'opération s'est bien déroulée
1 - Code erroné
2 - Erreur de lecture
3 - Erreur en écriture
4 - Fichier non trouvé
5 - Disque plein
8 - Fichier existant en écriture disquette
255 - Fin de données en lecture

** **RS-B (n ...)** n = 7 : 7 bits
n = 8 : 8 bits

RS-S (n ...)

n = 1	110 bauds
2	300
3	600
4	1200
5	2400
6	4800
7	9600

11 - Utilitaires et mots divers.

(	**(...)** utilisé sous la forme (XXX ...) pour inclure des commentaires dans une définition
(LINE)	**(n1 n2 ... str)** donne les paramètres de la ligne n1 du block n2
.S	**(...)** affichage non destructif du contenu des piles
ABORT	**(...)** mot exécuté lors d'une erreur (vide les piles et rend la main au clavier)
AUTORUN	**(...)** AUTORUN XX positionne XX comme exécuté lors du chargement du dictionnaire
BELL	**(...)** fait un bip
C/L	**(... adr)** variable utilisée par VLIST et par l'éditeur pour le nombre de caractères par ligne
EDIT	**(n ...)** entre en mode édition dans le block n
L/SCR	**(adr ...)** variable utilisée par l'éditeur pour le nombre de lignes par écran
LIST	**(n ...)** affiche le block n sur le périphérique de sortie
ONERR	**(...)** ONERR XX positionne XX comme mot à exécuter en cas d'erreur
REDIT	**(...)** retourne en mode édition dans le dernier écran utilisé antérieurement
RND	**(n ...)** donne un nombre au hasard entre 0 et n-1
RUN	**(n ...)** lit sur le périphérique d'entrée le dictionnaire sauvegardé avec le numéro n, et exécute le mot positionné par AUTORUN (le cas échéant)
SAVE	**(n ...)** sauvegarde le dictionnaire actuel avec le numéro n
WHERE	**(...)** retourne sous l'éditeur après une erreur d'interprétation et se positionne sur le mot qui suit l'erreur

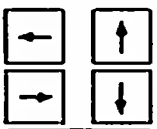
---

.LINE	**(n1 n2 ...)** affiche la ligne n1 de l'écran n2
79-STANDARD	**(...)** positionne WIDTH à 31
CALL	**(adr ...)** exécute le sous-programme à l'adresse adr. Les paramètres sont passés par l'intermédiaire des variables REGD, REGU, ...
COLD	**(...)** redémarrage à froid du système
MESSAGE	**(n ...)** affiche le message d'erreur numéro n
QUIT	**(...)** initialise la pile des retours, positionne le mode exécution, et rend le contrôle au terminal
REGD	
REGU	**(adr ...)** variables permettant de passer les paramètres pour les registres D, U, X, Y à un sous-programme en langage machine
REGX	
REGY	
SCR	**(... adr)** variable contenant le numéro du dernier écran accédé par l'éditeur

---

12 - L'éditeur.

Pour donner des ordres à l'éditeur, on utilise les touches spéciales suivantes du clavier :



déplacement curseur.

ENTREE place le curseur en début de la ligne suivante.

RAZ	efface tout.
STOP	arrête l'édition, sans exécuter le contenu de l'éditeur.
EFF	efface le caractère à la position du curseur.
INS	insère un espace à la position du curseur.
CNT Y	arrête l'édition, et exécute le contenu de l'éditeur.
CNT D	place le curseur alternativement en début ou en fin de ligne.
CNT F	tabulation à gauche (les positions sont 1, 6, 11, 16, 21, 26, 31, 36).
CNT G	tabulation à droite.
CNT S	scinde en deux la ligne courante à la position du curseur.
CNT U	réunit deux lignes consécutives.
CNT X	détruit la ligne du curseur (cette ligne est conservée dans le tampon d'édition, à condition que celui-ci soit vide.
CNT E	insère la ligne contenue dans le tampon d'édition devant la ligne du curseur.
CNT C	copie dans le tampon d'édition la ligne du curseur, sans modifier l'écran (à condition que le tampon soit vide).
CNT T	vide le tampon d'édition.
CNT O	passe à l'écran précédent.
CNT Q	passe à l'écran suivant.
CNT P	recopie l'écran sur imprimante.
CNT W	sauvegarde l'écran courant sur cassette.
CNT R	charge l'écran courant depuis la cassette.

MESSAGES D'ERREUR

0	**UK**	UNKNOWN
1	**ES**	EMPTY STACK
2	**DF**	DICTIONNARY FULL
3	**BA**	BAD ARGUMENT
4	**NU**	NOT UNIQUE
5	**TB**	TOO BIG
6	**DR**	DISK RANGE
7	**FS**	FULL STACK
8	**LO**	LOAD ONLY
9	**CO**	COMP ONLY
10	**XO**	EXEC ONLY
11	**CD**	CONDITIONAL?
12	**UF**	UNFINISHED
13	**PR**	PROTECTED
15	**DL**	DELIMITER ?
17	**I>**	INPUT > 255
18	**VF**	VOCABULARY FULL
19	**IO**	IO ERROR