TYPHON

"TAIFUN - BASIC-Compiler"

This programm has been developped in West-Germany by:

Imperial SOFTware Systems Germany
-Gerdes KG-
Rochus-Center

Lessenicher Str. 9

D - 5300 B o n n 1

Préface:

Ce programme est probablement le programme le plus universel pour les ordinateurs AMSTRAD CPC 464, 664 et 6128. A l'aide de ce programme, vous pouvez compiler des programmes que vous avez écrits en code BASIC (lequel est relativement simple) en code machine (un code beaucoup plus rapide).Le code compilé devient -selon le choix des instructions- de 20 à 100 fois plus rapide que la version BASIC. Ainsi, votre programme gagne énormement en qualité. En plus, le Typhon vous offre presque la totalité du répertoire des instructions du code BASIC (sauf les fonctions des nombres en virgule flottante) et y ajoute une ample extension du répertoire des instructions.

Le fonctionnement de ce programme:

Le compilateur (traducteur) travaille de manière interactive, c'est à dire, et le compilateur et le code traduit et le programme BASIC se trouvent en mémoire simultanément.
Ainsi, on ne perd pas de temps pour le chargement de cassette ou disquette, comme cela est le cas pour des programmes comparables qui existent pour d'autres ordinateurs.
Au cours de l'élaboration de ce programme, nous avons mis l'accent sur son efficacité. C'est pour cela que le compilateur peut réaliser la compilation,même de programmes assez longs, en 6 secondes seulement. En plus, on peut selon le genre de programmation - faire compiler des programmes BASIC jusqu'à 12 kilo-octets de longueur. Sans perdre du temps (p.ex. par mémorisation sur cassette ou disquette), on peut mettre au point ces programmes immédiatement en mémoire pour effectuer une correction rapide et simple au cas où il y aurait des erreurs.

Le répertoire du compilateur:

Un compilateur n'est pas fait pour réaliser des calculs exacts, car l'ordinateur AMSTRAD CPC dispose déjà de routines de calcul très rapides dans son système d'exploitation.Une intégration de la totalité des instructions de calcul aurait sensiblement diminué l'efficacité du compilateur en causant un besoin beaucoup plus grand de place en mémoire ainsi qu' une vitesse de traitement beaucoup moins grande. Le compilateur devrait être utilisé en première ligne pour le développement d'algorithmes de traitement (traitement de textes, traitement de l'information) ainsi que pour d'autres programmes utiles qui comportent le désavantage d'être asservis au temps. Vous obtiendrez de très bons résultats surtout en intégrant les instructions complémentaires du compilateur directement dans l'élaboration de vos programmes. Nous sommes convaincus de vous offrir avec le Typhon un outil de programmation très efficace.

2.) Vous n'avez le droit de copier le Typhon que pour votre utilisation privée. Une transmission à des personnes tiers n'est donc pas permisè
3.) Vous n'avez le droit de transmettre à des personnes tiers le "Runtime Package" ainsi que des routines faisant partie de celui-ci qu' en relation avec les programmes ayant été produits à l'aide du Typhon. Il n'est donc pas non plus permis de vendre des programmes qui ne font que définir des instructions supplémentaires du compilateur en forme de RSX.
4.) Dans les programmes compilés qui sont vendus ou transmis à des personnes tiers, le titre que produit le Typhon après la compilation doit être cité au moins une fois pour signaler que vous vous êtes servi de ce programme utilitaire.
5.) Le copyright des programmes publiés dans cette documentation (programmes source et programmes compilés) restera auprès de la société Imperial SOFTware Systems Gerdes KG Germany. Vous n'avez donc pas le droit de vendre ces programmes, sauf si ces programmes ne constituent qu' une partie minime de vos propres programmes.

Chargement des programmes:

cassette:
Introduisez la face A de votre cassette originale dans votre lecteur de cassette et appuyez sur:¥CTRL¥ et le petit ¥ENTER¥. Répondez avec ¥ENTER¥ à la question que vous posera le système d'exploitation.

disquette:
Introduisez la disquette originale dans l'unité de disquette.Après, introduisez RUN"Typhon" ¥ENTER¥.

Le processus de chargement se réalise.
Si le processus de chargement est terminé de manière correcte, le titre du compilateur et une question (changer paramètre? (O/N)) apparaissent sur l'écran. Au début, répondez avec ¥N¥ à cette question, s'il vous plaît, pour vous entrainer.
Si ce titre n'apparaît pas après un certain temps d'attente, il s'agit probablement d' une erreur de chargement. Essayez encore une fois de réaliser le processus de chargement.

Quelques remarques concernant cette documentation:

Nous supposons que vous avez lu le manuel pour le CPC et que vous connaissez du moins la structure de base d' un programme BASIC. Comme la plupart des instructions du compilateur sont identiques à celles du code BASIC du CPC, nous n'avons pas pris la peine de vous donner des explications supplémentaires concernant ces instructions. C'est pour cela que nous vous recommandons d'avoir votre manuel pour le CPC à côté de vous en étudiant à fond cette documentation. En outre, il vous sera utile de tester chaque instruction directement après avoir lu les explications concernant cette instruction pour s'accoutumer tout de suite au fonctionnement des commandes.

Attention: Dans cette documentation, nous avons mis un point d'exclamation (!) devant les instructions élargies ou les instructions RSX-BASIC. Mais cela ne représente pas - dans ce contexte - un point d'exclamation mais la barre verticale que vous obtenez en appuyant sur ¤SHIFT¤ et ¤ ¤.

Par la suite, cette documentation se divise en les parties suivantes:

I. Explication de la totalité des instructions
II. Intégration du code machine dans votre programme
III. Explication de la fonction de compilation
IV. Les paramètres et l'occupation de la mémoire
V. Le déplacement (relocation) de votre programme
VI. Traitement multitâche en relation avec le Typhon
VII. Production de RSX à l'aide du Typhon
VIII. Des conseils pour l'adaptation de programmes BASIC
IX. Messages d'erreur donnés par le Typhon
X. Exploitation à disquette
XI. Particularités concernant le CPC 664 et le CPC 6128
XII. Exemples pour l'utilisation du Typhon

Nous vous recommandons de consulter d'abord le chapitre III, car, autrement, vous ne pouvez pas tester les instructions. S'il y a, dans ce chapitre, des parties que vous ne comprenez pas tout de suite, ne vous en préoccupez pas, car la compréhension complète ne viendra qu'après la lecture de la documentation entière.

I. Explication de la totalité des instructions

Les instructions sont énumérées par ordre alphabétique pour vous faciliter la recherche d'instructions déterminées si vous vous servez du programme plus tard.

Pour les explications, nous utilisons:

ST1,ST2,ST3,...	des STRING's, c'est à dire: des variables de chaînes de caractères
N1,N2,N3,.....	variable de nombres entiers (de -32768 à 32767)
M1,M2,M3.....	valeur d'un octet. Un nombre entre 0 et 255 ou une variable avec une valeur qui se trouve dans cette echelle de nombres

Nous indiquons chaque fois la syntaxe (format de l'instruction). Ensuite, nous vous donnons des explications ou nous vous recommandons de consulter le mode d'emploi pour le CPC.

En ce qui concerne les formats de nombres acceptés par le compilateur:

Le compilateur comprend des nombres dans toutes les structures de nombres qu'on trouve aussi en code BASIC, tant qu'il s'agit de nombres entiers (de -32768 à 32767): p.ex. nombres binaires, hexadécimaux et, bien entendu, décimaux. Ces formats de nombres peuvent apparaître et dans le programme et dans les instructions INPUT.

Les opérations de calcul admissibles sont les suivantes: opérations de calcul fondamentales, opérations logiques, opérations de comparaison et les fonctions nommées ci-dessous.

En ce qui concerne des types de variables, vous pouvez utiliser des nombres entiers (A%), nombres réels (A!) et chaînes de caractères (STRING's), aussi en forme de variables matricielles. Les nombres réels sont traités comme les chaînes de caractères pour que vous ayez à votre disposition les cinq octets dont vous aurez besoin pour la mémorisation de nombres en virgule flottante.Normalement, il n'y a que les chaînes de caractères et les nombres entiers qui vous seront utiles. Des variables sans indication de type sont traitées comme des variables de nombres entiers jusqu'à ce que vous ayez réalisé un DEFINT, DEFREAL ou DEFSTR.

Ce signe, mis directement devant une variable, fournit l'adresse de cette variable dans la mémoire. Cela fonctionne d'ailleurs sans restriction en code BASIC, même si cela n'a été publié la plupart du temps qu'en relation avec des extensions d'instructions (RSX).
Quand il s'agit de variables de nombres, il vous est fournie une adresse, où on peut directement lire le contenu des variables à l'aide de PEEK W ou le modifier à l'aide de POKE W; cela correspond donc au code BASIC.
Lorsqu'il s'agit de variables de chaînes de caractères, il vous est fournie - autrement qu'en BASIC directement l'adresse de rangement où est enregistré le texte terminant en un signe ASCII 0.
Lorsqu'il s'agit de variables matricielles (arrays), on reçoit toujours l'adresse du premier élément - aussi autrement qu'en BASIC.

ABS (N1) Change la valeur de N1 en une valeur positive, si celle-là est d'abord négative.
Attention: ABS(-32768)=0 !

AFTER N1,M1 GOSUB
Correspond au code BASIC. Voir aussi "EVERY" et le chapitre sur le traitement multitâche.

N1 AND N2 Instruction logique AND avec N1 et N2.

ASC (ST1) voir BASIC.

AUTO N1 Boucle d'attente. Halte de N1 * 0.00001 sec.

BIN$ (N1) Voir BASIC. Un deuxième paramètre n'est pas possible.

BORDER N1,N2 Voir BASIC.

CALL N1 Voir BASIC.

CAT Voir BASIC.

CHR$ (M1) Voir BASIC.

CINT (N1,M1,N1) Fonction comparable à celle de ROUND (voir ROUND); mais on cherche en avant au lieu d' en arrière.

CLEAR Efface toutes les variables. Si derrière CLEAR est indiquée encore une autre valeur, l'ordinateur ne tient pas compte de cette valeur.

CLEAR INPUT Efface toute la mémoire intermédiaire pour le clavier. Il est utile de se servir de cette instruction p.ex. avant l'introduction du texte par INPUT dans un jeu, car les interrogations de touches par INKEY ou JUY ne sont pas effacées de la mémoire intermédiaire.

CLG Voir BASIC.

CLOSEIN Voir BASIC.

CLOSEOUT Voir BASIC.

CLS #M1 Voir BASIC.

CLS #M1) En tant que fonction (A=CLS #0), cette instruction vous fournit la hauteur (en unités de caractères) du WINDOW indiqué.

CONT C'est le moment où vous avez la possibilité de traiter des évènements synchrones (consultez aussi le chapitre sur le traitement multitâche).

COS M1,M2 Si M1 est zéro, le curseur du système d'exploitation, qui apparaît pendant le processus du PRINT, est mis hors service. Autrement, il est mis en service.M2 fait la même chose pour le curseur pour l'opérateur. On peut omettre un des deux paramètres M1 ou M2.

CREAL (N1,N2,N3) Instruction pour l'appel du code machine. Dans N1, vous trouvez l'adresse de la routine. N2 est intégré dans le registre DE et A. HL contient N3. Si on exécute une instruction "RET Z80", cette fonction du compilateur fournit la valeur en HL.
Exemple: A=CREAL(&BB00,0,1)

DATA..... Voir BASIC.
Mais il faut mettre des guillemets (") devant et derrière les chaînes de caractères'

DEF B Change toutes les instructions PRINT en extraction en forme HEX. L'ordinateur extrait toujours l'octet de poids faible en format HEX.

DEF DATA.... Définit RSX. Voir chapitre sur RSX.

DEF N	Toutes les extractions par PRINT sont représentées comme nombres entiers de -32768 à 32767.
DEF READ....	Adopte des valeurs de l'appel RSX. Vous trouvez d'autres explications concernant cette instruction au chapitre sur RSX.
DEF U	Toutes les extractions par PRINT sont représentées dans l'echelle de nombres de 0 à 65535.
DEF W	Toutes les extractions par PRINT sont représentées comme nombres hexadécimaux de 2 octets.
DEFINT	Syntaxe comme en code BASIC. Cette instruction doit apparaître dans le programme avant qu'on ne puisse utiliser une des variables.
DEFREAL	Syntaxe comme en code BASIC. Les variables sont définies en tant que variables de chaînes de caractères.
DEFSTR	Voir DEFREAL
DEG N1,N2,M1,M2	Dessine un cercle du rayon M1 autour du point ayant pour abscisse N1 et pour ordonnée N2. Vous pouvez désigner M2, mais cela n'est pas obligatoire. M2 indique la couleur du cercle. En MODE 2, le rayon peut atteindre 127 au maximum, dans les autres modes, il peut représenter des valeurs jusqu'à 255. En MODE 2, la routine est aussi beaucoup plus lente que dans les autres modes.
DELETE N1	Boucle d'attente. Si vous introduisez cette instruction, le programme attend N1/50 secondes.
DI	Disable Interrupts. Fonctionne comme code BASIC pour les évènements synchrones (autrement, cette instruction ne fonctionne pas du tout).
DIM N1 (xx),...	Voir BASIC. Mais xx doit être un nombre.
DRAW N1,N2,M1	Voir BASIC.
DRAWR N1,N2,M1	Voir BASIC.
EDIT N1	Halte de N1/1000 secondes.
EI	Enable Interrupts. Fonctionne comme code BASIC pour des évènements synchrones (autrement, cette instruction ne fonctionne pas du tout). (Consultez aussi le chapitre sur le traitement multitâche).

ELSE — Voir BASIC.
On ne peut utiliser qu' un seul ELSE dans une ligne de programme.

END — A l'aide de cette instruction, vous retournez au BASIC.

ENT — Voir BASIC.

ENV — Voir BASIC.

EOF — Voir BASIC.
Cette interrogation est réalisée aussi pour disquette de manière que la fin du fichier puisse être reconnue correctement.

ERL (N1,N2,N3) — Instruction Block-Move. Une zone de mémoire est déplacée en avant (comme LDIR en Z80).N1 indique , combien d'octets doivent être remplacés. N2 indique, où doivent se trouver les données qui commencent à N3.
Exemple:
10 POKE &C000,255:ERR (&3FFF,&FFFE,&FFFF)

ERR — Voir ERL.
Mais la zone de mémoire est déplacée en arrière

EVERY N1,M1 GOSUB — Appèle toutes les N1/50 secondes le sous-programme qui est indiqué derrière l'instruction GOSUB. En principe, l'exécution de cette instruction correspond à celle du code BASIC; mais on court le risque d' un arrêt du système quand il s'agit d'évènements asynchrones; et en ce qui concerne des évènements synchrones, on doit utiliser CONT.
(Consultez aussi le chapitre sur le traitement multitâche).

FIX (N1,M1,N2) — Cette instruction exécute une commande BLOCK-XOR. Si on commence par N2, une instruction XOR est exécutée N1 fois entre M1 et chaque octet.

FOR — Voir BASIC.

FRE — Fournit la position en mémoire sur laquelle le programme est en train de travailler.

GOSUB nn — Voir BASIC.

GOTO nn — Voir BASIC.

HEX$(N1) — Voir BASIC.
Un deuxième paramètre n'est pas permis, de manière que vous deviez utiliser DEF B et DEF W pour des extractions de nombres mises en format.

HIMEM — Voir BASIC.
HIMEM est mis sur la position de mémoire qui se trouve directement en dessous de celle du début du programme au moment de la mise en action du compilateur et au moment de l'appel du programme compilé. A partir de cette position de mémoire, des mémoires intermédiaires (p.ex. pour SYMBOL AFTER et OPENIN de même que pour OPENOUT) s'étendent vers le bas , de façon qu'on doive tenir compte de cet encombrement en mémoire .

IF — Voir BASIC.
Mais lorsqu'il s'agit de comparaisons de chaînes de caractères, ne doivent apparaître que des chaînes de caractères simples sans opérations-String quelconques.
p.ex.: BASIC: IF MID$(A$,1,1)="XAX" THEN.....
Typhon: X$=MID$(A$,1,1):IF X$="XAX" THEN.....

INK — Voir BASIC.

INKEY (M1) — Voir BASIC.

INKEY$ — Voir BASIC.

INP (M1) — Voir BASIC.

INPUT — Voir BASIC.
Exactement comme en BASIC, plusieurs variables pour INPUT sont possibles, et des chaînes de caractères et des variables de nombres. Lorsqu'on utilise plusieurs variables pour INPUT, les introductions au cours du déroulement du programme doivent être structurées par des virgules. De même qu'en code BASIC, la virgule indique en tout cas la fin d'une chaîne de caractères de façon qu'on doive éventuellement utiliser LINE INPUT. INPUT a d'ailleurs une limitation effective de longueur. Si vous mettez celle-là sur '0' p.ex. lorsque vous indiquez les paramètres, le compilateur n'accepte que 8 caractères au maximum en mémoire intermédiaire. Au-delà de ces introductions, l'ordinateur n'en accepte plus, comme c'est aussi le cas en code BASIC après l'introduction de plus de 256 caractères.

En outre, il y a encore quelques fonctions Editor peu connues:

CTRL+*↑* met le curseur au début de la ligne d'introduction
CTRL+*↓* met le curseur à la fin de la ligne d'introduction
CTRL+*←* met le curseur au début de la ligne d'écran
CTRL+*→* met le curseur à la fin de la ligne d'écran

£CTRL£+£TAB£Transfert du mode d'insertion au mode de recouvrement (fonctionne aussi sur le 664 si on se sert du Typhon (autrement, cette instruction ne fonctionne pas à cause de ROMs défectueux).

INSTR — Voir BASIC.

INT (N1) — Voir BASIC.

JOY (M1) — Voir BASIC.

KEY — Voir BASIC.

KEY DEF — Voir BASIC.

LEFT$ — Voir BASIC.

LEN (ST1) — Voir BASIC.

LET — Voir BASIC.

LINE N1 — Vous offre la possibilité de dérouler l'écran (scrolling) de la manière que vous souhaitez. Le début de la mémoire de l'écran est déplacé de N1/40 largeurs d'écran; les sections de l'écran disponibles après ce déplacement ne sont pas effacées. Ainsi, LINE 40 p.ex. déroule l'écran vers le haut et LINE-1 vers la droite sans effacer les marges.

LINE INPUT ST1 — Voir BASIC.

LOAD ST1,N1 — Voir BASIC.
Ne fonctionne que pour code machine ou pour fichiers binaires. N1 est l'adresse de chargement qu'on peut omettre (ne pas indiquer ",B")

LOCATE — Voir BASIC.

LOWER$ — Voir BASIC.

MAX — Voir BASIC.

Memory N1 — Voir BASIC.

MERGE — Intégration de code machine dans votre programme. Consultez le chapitre II. de cette documentation, s'il vous plaît.

MID$ — Voir BASIC.

MID$ (ST1,M1,M2)=ST2
Assigne la partie de ST2 définie par M1 et M2 à la variable ST1. Cette instruction fonctionne aussi en BASIC AMSTRAD.

MIN — Voir BASIC.

MOD	Voir BASIC.
MODE M1	Voir BASIC.
MODE	En tant que fonction (p.ex. A=Mode), cette instruction fournit le mode de l'écran
MOVE	Voir BASIC.
MOVER	Voir BASIC.
NEW	Effacer toute la mémoire.
NEXT nn	Voir BASIC.
NOT	Voir BASIC.
ON M1 GOSUB	Voir BASIC.
ON BREAK XXXX	Exécute l'instruction XXXX, si on appuye sur *ESC* au moment de l'interrogation
ON SQ(M1) GOSUB	Voir BASIC.
OPENIN ST1	Voir BASIC.
OPENOUT ST1	Voir BASIC.
N1 OR N2	Voir BASIC. Opérateur logique.
ORIGIN	Voir BASIC.
OUT M1,N1	Voir BASIC.
PAPER M1	Voir BASIC.
PAPER (#M1)	Fournit comme fonction (A=...) la couleur d'arrière-plan du WINDOW M1
PEEK (M1)	Voir BASIC.
PEEK W (N1)	Fournit la valeur de 2 octets des positions de mémoire N1 et N1+1
PEN M1	Voir BASIC.
PEN (#M1)	Fournit comme fonction (A=...) la couleur d'ecriture du WINDOW M1.
PLOT	Voir BASIC.
PLOT PAPER M1	Choisit M1 en tant que couleur d'arrière-plan
PLOT PEN M1	Choisit M1 en tant que couleur de caractères.
PLOTR	Voir BASIC.
POKE	Voir BASIC.

POKE W(N1),N2	Met aux positions de mémoire N1 et N1+1 la valeur de N2
POS	Voir BASIC.
POS D	Déroule une ligne de l'écran vers le bas et remplit la première ligne par la couleur d'arrière-plan.
POS L	Déroule 1/40 de la largeur de l'écran vers la gauche et remplit la marge droite par la couleur d'arrière-plan.
POS R	Déroule 1/40 de la largeur de l'écran vers la droite et remplit la marge gauche par la couleur d'arrière-plan.
POS U	Déroule une ligne vers le haut et remplit la dernière ligne par la couleur d'arrière-plan.
PRINT	Voir BASIC. Des opérations spéciales de PRINT comme TAB,SPC, "," et ";" sont admises. TAB et , sont même beaucoup plus rapides qu'en BASIC. Mais il n'y pas de USING. Si on introduit PRINT CHR$, aussi le caractère ASCII 0 est extrait; autrement, ce caractère désigne la fin de la chaîne de caractères et, pour cela, n'est pas traité.
PRINT	Comme fonction (A=PRINT), cette instruction fournit l'état de l'imprimante. 0 indique que l'imprimante est en train de travailler; tout autre nombre indique qu'elle peut recevoir des caractères.
RAD M1	Remplit la section à partir de la position actuelle du curseur de graphique par la couleur M1. La section peut avoir n'importe quelle forme. Pour limiter la section, l'ordinateur choisit toute sorte de couleur différente de celle de la première position. Si le premier point a déjà la couleur M1, l'ordinateur ne remplit rien du tout. Bien entendu, cette instruction est beaucoup plus rapide en MODE 0 qu'en MODE 2.
RANDOMIZE	Voir BASIC (mais ne pas indiquer de paramètre')
READ	Voir BASIC.
READ CHR$(#M1)	En tant que fonction, cette instruction lit un caractère sur l'écran. Le caractère est lu par la position momentanée du curseur de l'appareil entrée/sortie indiqué. Si l'ordinateur n' a pas pu reconnaître le caractère, il signale " "
RELEASE M1	Voir BASIC.
REM ou '	Voir BASIC.

REMAIN (M1) Voir BASIC.

RESTORE Voir BASIC.
Mais il n'est pas possible d'indiquer un numéro de ligne.

RETURN Voir BASIC.

RIGHT$ Voir BASIC.

RND (M1) Fournit un nombre aléatoire entre 1 et M1.

ROUND (N1,M1,N2) Il s'agit d'une fonction de recherche. N1 indique, combien d'octets doivent être scrutés. M1 est l'octet qui doit être cherché. N2 indique la première adresse à partir de laquelle on doit chercher l'octet M1. A partir de N2, on compte N1 octets en arrière. Si l'ordinateur trouve l'octet M1, cette fonction fournit l'adresse correspondante en mémoire. (comparable à CPDR en Z80)

RUN ST1 Charge et met en action un programme en code machine de cassette/disquette. Le nom du programme est ST1.

Met de nouveau le programme en action et efface toutes les variables.

SAVE ST1,N1,N2,N3
Enregistre le programme en code machine (fichier binaire) sous le nom ST1 sur cassette/disquette. N1 désigne l'adresse de départ, N2 la longueur et N3 (qu'on ne doit pas forcément indiquer) est l'adresse de départ automatique.

Voir BASIC.

SOUND Voir BASIC.

SOUND #M1,M2,M3,M4
Cette instruction vous offre la possibilité d'une application simple des fonctions SOUND du CPC. Elle est très efficace et facile à manier. Elle existe aussi pour d'autres ordinateurs sous l'instruction PLAY.
M1 désigne le canal (1-3), M2 l'octave (1-8), M3 indique la hauteur du son (1-12; do, do dièse, ré, ré dièse, mi....) et M4 (qui n'est pas obligatoire) indique l'intensité sonore. On ne peut omettre l'intensité sonore que si on l'a déjà indiquée avant. Si vous voulez arrêter un son, réalisez un SOUND, pour lequel vous désignez l'intensité sonore de 0; ensuite, le son ne sera plus audible.

SOUND TO M1,M2 Cette instruction écrit la valeur M2 dans le registre PSG M1 (programmable sound generator). Le PSG du CPC a les registres suivants:

Fréquences:
R0 Réglage de précision canal A
R2 canal B
R4 canal C
R1 Réglage approximatif canal A
R3 canal B
R5 canal C

R6 Fréquence de bruit (0-31)
R7 Mise en activité:
Bits: 0 son canal A 0=activé
1 B "
2 C "
3 bruit canal A "
4 B "
5 C "
6 et 7 doivent être mis à zéro
R8 Intensité sonore canal A
R9 B
R10 C
R8-R10: (0-15, 16: mise en activité de la courbe enveloppante)
R11 Réglage de précision de la fréquence de la courbe enveloppante
R12 Réglage approximatif de la fréquence de la courbe enveloppante.
R13 Forme de la courbe enveloppante

SPACE$ Voir BASIC.

SPEED INK M1,M2 Voir BASIC.

SPEED KEY M1,M2 Voir BASIC.

SPEED WRITE M1 Voir BASIC.
Les valeurs admissibles pour M1 fournissent les rapports BAUD suivants:

Valeur	rapport BAUD
0	1000
1	2000
2	3000
3	3600

SQ Voir BASIC.

SQR Voir BASIC.

STEP Voir BASIC.
(mais ne désignez pas de variables pour la progression)

STOP Voir BASIC.

STR$ Voir BASIC.

STRING$	Voir BASIC.
SYMBOL	Voir BASIC.
SYMBOL AFTER	Comparable au BASIC. Pour rendre possible une exécution sans problèmes, cette instruction a été modifiée un peu,comparé au BASIC. Vous devez seulement faire attention lors de la mise au point, car la place pour les nouveaux caractères est réservée sous HIMEM, et au moment du retour au BASIC, HIMEM est mis en dessous du tableau des caractères définis. Donc, tenez compte du HIMEM lors de votre mise au point en mémoire, car, si on compile le programme plusieurs fois, le tableau des caractères définis peut s'étendre tellement vers le bas que le système d'exploitation s'arrête.
TAG	Voir BASIC.
TAGOFF	Voir BASIC.
TAN	Attend jusq'à ce que le faisceau d'électrons ait produit la prochaine image, de façon que des extractions sur écran puissent être synchronisées avec celle-ci. Correspond donc à l'instruction CALL &BD19.
TEST	Voir BASIC.
TESTR	Voir BASIC.
TIME	Voir BASIC.
TROFF	Voir BASIC. Arrête la fonction TRACE.
TRON	Voir BASIC. Cette instruction est intégrée dans le programme au cours de la compilation; c'est pour cela que le programme devient plus long. Faites attention lorsque vous intégrez le code machine dans votre programme, car un arrêt du système d'exploitation en peut être la conséquence.
UPPER$	Voir BASIC.
VAL	Voir BASIC. Comme si vous introduisiez INPUT, des blancs devant des nombres de même que les systèmes de nombres décimaux, hexadécimaux et binaires sont acceptés.
VPOS	Voir BASIC.
WAIT	Voir BASIC.
WEND	Voir BASIC.

WHILE	Voir BASIC. Chaque boucle WHILE doit être terminée par une instruction WEND.
WIDTH #M1)	En tant que fonction (A=...), cette instruction fournit la largeur du WINDOW M1 en caractères.
WINDOW	Voir BASIC.
WINDOW SWAP	Voir BASIC.
WRITE	Voir BASIC. En principe, cette instruction est un PRINT modifié, qui interprète , ne pas comme tabulatrice mais comme caractère et qui extrait des guillemets autour de chaînes de caractères. SPC, TAB....sont donc aussi admis, si on utilise l'instruction WRITE.
XOR	XOR logique de deux facteurs (voir BASIC).
XPOS	VOIR BASIC.
YPOS	VOIR BASIC.
ZONE	VOIR BASIC. ZONE a la valeur 16 au début du programme.
↑	Cette fonction n'est nécessaire que pour RSX (voir chapitre "Exploitation à disquette").
	RSX est traité par le Typhon comme il est traité en BASIC. (Consultez le chapitre "Exploitation à disquette",s'il vous plaît).

II. Intégration de code machine dans votre programme

Par les fonctions pour l'intégration de code Z80 dans votre programme, on peut encore améliorer des programmes qu'on écrit à l'aide du Typhon.
Mais pour réaliser cela, vous avez besoin de connaissances suffisantes du code machine. Si vous n'avez pas encore de connaissances dans ce domaine, nous vous recommandons la lecture de la littérature spéciale correspondante.

Il y a ,en principe, deux possibilités d'intégrer le code machine dans votre programme:

1.) On écrit une routine séparée qui peut être appelée par CALL ou CREAL
ou
2.) On intègre la routine directement dans le programme (par MERGE).

A cet endroit, nous ne voulons expliquer que l'instruction MERGE, car l'utilisation de CREAL et CALL ne devrait pas poser de problèmes.

A l'aide de l'instruction MERGE, vous pouvez introduire des OPCODES directement , indépendamment du format des nombres (binaires, hexadécimaux, décimaux). La routine que vous produisez de cette manière, se trouve plus tard à l'endroit du programme où vous l'avez introduite.

```
exemple:        MERGE &21,-1,&C9
cela signifie:  LD HL,&FFFF
                RET
```

Cette possibilité ne ferait pas encore preuve d'une efficacité trop grande, s'il n' y avait pas encore une fonction supplémentaire:

Vous pouvez transférer les valeurs des variables de votre programme dans un programme en code machine, car si lors d'une instruction MERGE apparaît au lieu d'un nombre - remplaçant un OPCODE ou un facteur - un nom de variable, au lieu de la variable est désignée par l'ordinateur la position de mémoire de cette variable.

exemple: Vous voulez transférer la valeur d'une variable XX dans le registre HL:

```
MERGE &2A,XX
```

cela signifie: LD HL, (XX)

Au cours du procéssus de compilation, le compilateur remplace automatiquement XX par l'adresse de la valeur de la variable XX.

Exemple pour une boucle d'attente:

```
  10 FOR I=1 TO 100:XX=100:GOSUB 1000:NEXT I:STOP
1000 MERGE &2A,XX      LD      HL,(XX)
1010 MERGE &2B  boucle DEC     HL
1020 MERGE &7C         LD      A,H
1030 MERGE &B5         OR      L
1040 MERGE &20,&FB     JR      NZ,boucle
1050 MERGE &C9         RET
```

Pour rendre votre routine deplaçable aussi quand il s'agit de sauts absolus et adressages absolus, vous pouvez vous servir de l'instruction FRE.TRON y ajoute toujours six octets. C'est pour cela que vous devez utiliser TROFF.

III. Explication de la fonction de compilation

Cette partie de la documentation est consacrée à la manière de laquelle on se sert du compilateur, donc de la traduction de votre programme introduit ou chargé.
Vour déclanchez le processus de compilation en appuyant sur le petit ¤ENTER¤. Sur l'écran, apparaissent de différentes données concernant votre programme (si vous n'avez pas fait d'erreurs!).

La longueur du programme BASIC en octets, la longueur du programme compilé en octets et la longueur de la mémoire de variables.
Ensuite, vous avez la possibilité d'appeler le programme traduit à l'aide de *A*, d'enregistrer le programme compilé à l'aide de *E* ou de retourner au BASIC à l'aide de *B*.
Si vous choisissez *E*, l'ordinateur vous posera toutes les questions nécessaires, de manière qu' une explication plus détaillée ne soit plus nécessaire. Le programme enregistré peut être, même s'il est déplacé, mis en action immédiatement par RUN"Nom" (le nom de votre programme).
Au cas où il y aurait encore des erreurs lors de la compilation de votre programme, le message d'erreur correspondant et la ligne erronnée sont indiqués. Pour connaître la signification des messages d'erreur, consultez le chapitre IX,s'il vous plaît.
Cette possibilité simple de compilation sera certainement celle que vous utiliserez la plupart du temps.

Mais le Typhon vous offre encore d'autres possibilités:

Vous pouvez,p.ex.,choisir la zone de mémoire dans laquelle devra opérer votre programme plus tard. Nous expliquons à cet endroit ce processus un peu plus compliqué que les autres:
Vous chargez Typhon de cassette ou disquette (voir ci-dessus). Puis, vous appuyez sur *O* (oui) lorsqu' apparaît la question: "Changer paramètre?"
Ensuite, l'ordinateur vous posera quelques questions. Vous répondez à celles-ci selon votre désir. Si vous ne désirez pas de modification des valeurs implicites, vous appuyez simplement sur *ENTER*.

Signification de la première question:
Vous pouvez indiquer, quelle longueur maximum peut avoir une chaîne de caractères définie comme matrice (String Array). Cette valeur devrait être calculée tout juste, car le programme compilé n'a pas de format String variable. Si vous indiquiez donc une longueur de 200 octets p.ex., cette matrice aurait toujours besoin de 20000 octets si vous avez 100 élements de matrice, même si les variables ne sont pas utilisées.
Mais, de l'autre côté, vous ne devez en aucun cas indiquer une valeur trop petite, car cela aurait probablement pour conséquence un arrêt du système d'exploitation.

La prochaine question concerne la longueur maximale des variables de chaînes de caractères.Dans ce domaine aussi, on ne devrait utiliser que peu de place en mémoire.

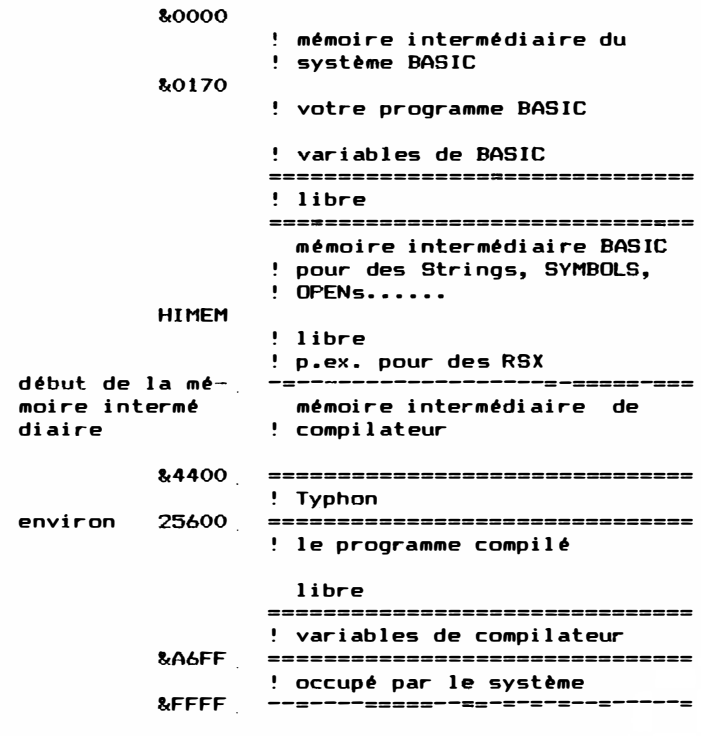
Les deux prochaines questions concernent la définition de la position de votre programme.
Mais d'abord quelques informations sur l'enregistrement du compilateur:

Au début de la mémoire se trouve votre programme BASIC. Ensuite, c'est le compilateur qui est enregistré. Celui-ci est suivi du code objet, c'est à dire , de votre programme traduit. Plus votre programme est long, plus cette zone devient grande. La mémoire du code objet s'étend donc du bas vers le haut.

Le compilateur provoque exactement le contraire dans la mémoire de variables, car celle-ci s'étend d'en haut vers le bas. Si votre programme et la mémoire de variables que vous utilisez sont trop grands, ces deux zones se revouvrent et le compilateur extrait un message d'erreur.

IV. Les paramètres et l'occupation de la mémoire

Quoique les paramètres aient déjà été expliqués en partie dans les chapitres III et V de cette documentation, nous les traitons encore une fois ici.Ce qui est essentiel pour la compréhension de la désignation des paramètres, c'est le plan d'occupation mémoire suivant:



Par modification des paramètres, vous pouvez déplacer en partie ces zones. Derrière la désignation des paramètres, l'ordinateur vous indique toujours la valeur antérieure introduite que vous pouvez adopter en appyant sur *ENTER*.

Il y a les paramètres suivants:

Longueur maximale de la matrice des chaînes de caractères	indique la chaîne de caractères la plus longue possible d'une matrice

Longueur maximale d'une chaîne de caractères	La place en mémoire réservée pour une chaîne de caractères.
Fin de la mémoire des variables	L'adresse de mémoire la plus haute des variables compilées
Adresse de début du code-objet	L'adresse la plus basse du programme compilé
Début de la mémoire intermédiaire au moment de la compilation	La position de mémoire la plus basse dont peut disposer le compilateur au moment de la compilation
Longueur maximale du texte au moment de INPUT	Nombre des caractères acceptés au moment de INPUT
Catégorie d'évènement	Introduisez *A* à cet endroit, et tous les évènements seront compilés comme asynchrones. Mais si, par contre, vous introduisez *S*, les évènements sont produits de manière synchrone. Mais dans ce cas-là, vous devez aussi utiliser CONT
Démarrage à froid (O/N) (oui ou non)	Si vous réalisez un démarrage à froid (O), les variables sont effacées et le titre du compilateur est extrait. Le contraire est le cas, si vous ne réalisez pas de démarrage à froid (N)

V. Le déplacement de votre programme

Si vous voulez maintenant déplacer votre programme, c'est à dire le mettre sur un position de mémoire qui ne correspond pas à celle définie d'avance par le compilateur, vous devez indiquer l'endroit où doit commencer le programme compilé, lorsque la quatrième question apparaît sur l'écran.

En plus, vous pouvez indiquer, quel encombrement en mémoire peuvent avoir le programme et les variables au maximum, si la troisième question apparaît sur l'écran. Vous obtiendrez la différence minimale en additionnant simplement pendant la compilation en mode normal - la longueur indiquée du code objet et l'envergure de l'encombrement en mémoire des variables.

Mais une fois réalisé ce déplacement par modification des valeurs nommées ci-dessus, vous ne pouvez plus tester votre programme par *A*. Vous pouvez seulement enregistrer votre programme compilé. C'est pour cela que vous deviez, avant de

réaliser ce processus de déplacement - avoir enregistré votre programme en forme correcte sur cassette/disquette en tant que programme BASIC pour vous épargner le travail pénible de tester le programme.

La question antépénultième s'informe de la longueur maximale d'introduction. Cette longueur détermine, à partir de quelle longueur une introduction faite au moment d'INPUT n'est plus acceptée par l'ordinateur. Cette fonction a la tâche d'éviter un recouvrement des variables de chaînes de caractères.

La dernière question concerne la manière d'initialisation de votre programme au moment de la mise en action. Normalement, cette valeur est mise sur ‡O‡ pour réaliser un démarrage à froid. Par cela, toutes les variables sont effacées au moment de la mise en action du programme.

Si, par contre, on introduit ‡N‡, le programme n'efface pas les variables, sauf si existe une instruction CLEAR.

On vous offre cette possibilité pour le cas où vous voulez faire coopérer deux programmes (à l'aide de "RUN"). Pour cela, il est nécessaire d'éviter que les variables soient effacées.

Mais pour un transfert correct des variables, il est nécessaire que les variables corresponantes soient localisées sur la même position de mémoire. Cela n'est le cas que si vous compilez les deux programmes avec les mêmes paramètres, immédiatement l'un après l'autre (sans les tester), en déclenchant le processus de compilation du deuxième programme en appuyant sur la touche de fonction ‡.‡ au lieu d'appuyer sur ‡ENTER‡.

Mais dans ce cas-là, ce programme devrait contenir une instruction CLEAR. Le premier programme peut très bien être compilé avec démarrage à froid; il n'est donc pas nécessaire que le premier programme contienne une instruction CLEAR.

VI. Traitement multitâche en relation avec le Typhon

Le Typhon dispose d'un traitement multitâche qui ressemble beaucoup à celui du BASIC. Les routines sont appelées en unités de temps de 1/50 sec. et il dispose de 4 sous-programmes de mesure de temps (Timer).

Une particularité du Typhon est la division en deux catégories d'évènements: des évènemenst synchrones et asynchrones.

En principe, les évènements synchrones sont les plus agréables, car EVERY, AFTER et ON SQ fonctionnent sans problèmes pour cette catégorie d'évènement. Le désavantage des évènements synchrones: ils ne sont traités que si le programme principal permet par CONT de les traiter. Dans le programme principal doit donc se trouver un "CONT" à plusieurs endroits. Il s'agit là d'une programmation simple, surtout si le programme principal consiste en une boucle.

Les évènements asynchrones sont prévus pour le cas où le traitement d'évènements synchrones ne peut pas être réalisé assez souvent ou assez vite. DI et EI ne coopèrent pas avec des évènements asynchrones, comme nous venons d'expliquer ci-dessus. En outre, ON SQ peut poser des problèmes sur le CPC 464 si on l'utilise pour des évènements asynchrones (à cause d'un défaut dans le système d'exploitation). Enfin, le moment de l'interruption n'est pas prévisible; elle peut se produire p.ex. aussi pendant le traitement d'une routine du système d'exploitation. Si la routine d'interruption réappèle la routine du système d'exploitation ou une autre qui utilise les mêmes variables, cela provoque l'arrêt du système d'exploitation.
C'est pour cela qu'on doit choisir très prudemment les instructions qu'on utilise dans la routine d'interruption.

VII. Production de RSX à l'aide du Typhon

Ce chapitre concerne la production de RSX. Les particularités de la compilation de RSX sont expliquées au chapitre X (Exploitation à disquette).

D'un côté, la longueur des programmes qui sont produits à l'aide du Typhon est limitée, de l'autre côté, le Typhon ne permet pas de calculs exacts. Cela fait supposer qu'il vaut mieux ne compiler à l'aide du Typhon que des routines asservies au temps et écrire le programme principal en BASIC. Pour pouvoir appeler sans problèmes les sous-routines, le Typhon utilise ce qu'on appèle des instructions RSX; ce sont des instructions devant lesquelles se trouve une barre verticale. Le compilateur connaît deux instructions pour le traitement de RSX:

1.) DEF DATA nom=ligne, nom1=ligne1, nom2=ligne2.....
Cette instruction définit les instructions supplémentaires, de façon que le BASIC puisse s'en servir. Lorsque, plus tard, on exécute l'instruction 'nom',le sous-programme 'ligne' est appelé. Celui-ci retourne au code BASIC par "RETURN". Certes, la table pour RSX est déjà produite au cours du processus de la compilation; mais les instructions RSX ne sont à votre disposition qu' après l'appel du programme compilé. D'ailleurs, on ne devrait compiler le programme qu'une seule fois lorsqu'on utilise RSX.

2.) DEF READ variable, variable$, variable1.....
Adopte des valeurs qui ont été indiquées par l'appel de la routine à l'aide de RSX. Et des variables de nombres et des variables de chaînes de caractères sont acceptées pour les valeurs adoptées.

p.ex.: dans la sous-routine est indiqué:
DEF READ A,B,A$,X

Le programme est appelé par:
|RSX,1,2,@Z$,100

Alors, en A se trouve la valeur 1, en B la valeur 2, en X la valeur 100 et en A$ le texte de Z$.

(si vous avez encore des problèmes de compréhension, consultez le chapitre des exemples de programmes, s'il vous plaît)

VIII. Des conseils pour l'adaptation de vos programmes BASIC

Il est pratiquement impossible de converter des programmes qui exécutent des calculs exacts. Mais, de toute façon, cela n'aurait pas pour conséquence de très grands avantages concernant la vitesse de traitement.

Il faut tenir compte du fait que la plupart des commandes et fonctions ne peuvent traiter que des chaînes de caractères simples (IF etc.). C'est pour cela qu'on doit - lorsqu'on veut réaliser des opérations compliquées - d'abord réaliser l'opération à l'aide d'une variable auxiliaire. Donc, n'utilisez pas INKEY pour IF, p.ex.

En outre, il est important de tenir compte de certaines restrictions nommées au chapitre I.(p.ex.: pas de paramètre pour RANDOMIZE).

Des calculs sont traités par le compilateur selon les lois de calcul générales. Seulement quand on utilise des opérateurs logiques pour des comparaisons par IF, il est utile de mettre des paranthèses.

Ce compilateur attend - comme tout compilateur à nombres entiers- de recevoir un nombre après un extrait commençant par "-" Il n'est donc pas possible d'introduire simplement des variables à préfixe négatif. A=-A et aussi A=SQR(100) p.ex. produisent des erreurs de syntaxe. A=0-A et A=100, par contre, sont admis.

Tenez aussi compte du fait que, pour des variables, uniquement les deux premières caractères sont importantes. Le compilateur ne pourrait donc pas distinguer entre "GRAND" et "GROS"; par contre, il pourrait très bien faire la distinction entre "MAXIMUM" et "MINIMUM".

Lorsque vous travaillez avec des routines d'interruption asynchrones, vous devez tenir compte du fait que le moment de l'interruption n'est pas prévisible.C'est pour cela que la routine principale et la routine d'interruption ne doivent pas se servir, les deux en même temps, du clavier, de l'écran ou de SOUND. Dans la routine principale doit être garanti - avant l'introduction de toutes les instructions PRINT, PLOT, INKEY et SOUND - qu'il n'y ait pas d'interruption. En outre, la routine d'interruption ne doit pas se trouver en position parallèle au ROM.C'est pour cela qu'on ne doit en aucun cas choisir pour le programme compilé une adresse de début qui soit en dessous de &4000. Vous ne perdez quand-même pas cette place en mémoire, car vous pouvez - avec beaucoup de précaution, bien entendu - y déposer des variables.

Tenez aussi compte du fait que le Typhon n'accepte pas de numéros de ligne qui dépassent 32767. Mais vous pouvez facilement résoudre de tels problèmes à l'aide d'un RENUM. Les instructions supplémentaires du compilateur ont, en partie, une syntaxe inhabituelle. Si vous avez des doutes concernant la syntaxe, consultez toujours cette documentation et introduisez au moins une fois les programmes d'exemple.

N'utilisez jamais plus d'un seul ELSE par IF; autrement, les résultats deviennent imprévisibles.

Des programmes compilés n'interrogent plus ¥ESC¥. C'est pour cela qu'on doit introduire l'instruction ON BREAK STOP chaque fois que vous voulez rendre possible la réalisation d'un BREAK. Chaque fois que le programme arrive à cette instruction, le compilateur interroge ¥ESC¥ et retourne , le cas échéant, au code BASIC. Enfin, tout programme qui doit retourner au BASIC, doit être terminé par un END ou un STOP. Si vous ne terminez pas vos programmes de cette manière, l'ordinateur vous signale ,au moment de l'appel du programme, votre erreur par le message "RETURN superflu", mais quand il s'agit d'un programme enregistré, le système d'exploitation s'arrête.

IX. Messages d'erreur donnés par le Typhon

Les messages d'erreur ne sont écrits qu'en anglais, car il peut y avoir aussi des messages d'erreur donnés par le code BASIC qui, en tout cas, sont écrits en anglais.

Les messages d'erreur sont extraits au cours du processus de compilation avec indication de la ligne erronnée; la plupart de ces messages sont extraits au cours du deuxième passage de compilation.

Syntax Error in line nnnn

Voir BASIC.

Line Error in line nnnn

Ce message d'erreur correspond au message "Line does not exist".

FOR/NEXT Error in line nnnn

1.) L'argument de STEP est une variable
2.) Il y a plus de NEXTs que de FORs dans le programme (et vice versa).
3.) La variable de déroulement opératoire n'est pas de variable de nombres entiers. La cause en peut être un DEFSTR ou DEFREAL,p.ex.

Illegal Statement Error in line nnnn

Comparable au SYNTAX-Error et au "Unknown Command"

Variable Error in line nnnn

Erreur dans le traitement de variables matricielles.

Variable Overflow Error in line nnnn

Signale que le Typhon n'a plus assez de place pour la compilation. En ce cas-là, le début de la mémoire intermédiaire doit être mis à une valeur moins grande, avant le processus de compilation.

WHILE/WEND Error in line nnnn

Comparable au FOR/NEXT Error, uniquement pour boucles WHILE/WEND.

Out of Memory Error in line nnnn

Si cette erreur se produit, vous devez diviser votre programme en plusieurs sous-programmes, car il ne reste plus de place en mémoire.

Outre ces messages d'erreur, le compilateur vous signale encore deux autres erreurs:

Chevauchement de code/variables

Ce message d'erreur signale, que la zone de mémoire des variables et la zone de mémoire du code traduit se recouvrent. Vous devez donc modifier les paramètres.

RETURN superflu

Ce message est extrait, si le programme se heurte à un RETURN sans introduction préalable de GOSUB, ou si le programme retourne au BASIC sans introduction de STOP. Cela mène normalement à un arrêt du système d'exploitation.

X. Exploitation à disquette

Si vous utilisez des disquettes, vous ne devriez travailler qu'avec des copies et garder la disquette originale à un endroit sûr. Cela est important surtout pour d'éventuelles réclamations, car si vous avez enregistré quelque chose sur la disquette originale ou laissé celle-là dans l'unité à disquette, au moment où vous avez débranché celle-là, cela est certainement la cause de problèmes de chargement et vous devez payer vous-même les frais qui en résultent (éventuellement les frais pour une nouvelle cassette, le port etc.....).

Nous tenons à signaler, que le Typhon n'efface pas de programmes qui pourraient se trouver en mémoire au moment du chargement.Vous avez donc, p. .ex. si vous avez oublié de charger le Typhon, la possibilité de charger celui-ci après avoir chargé le programme BASIC et ensuite, compiler le programme BASIC. Il peut seulement y avoir des problèmes, si votre programme BASIC a besoin de plus de place en mémoire que n'est à votre disposition après le chargement du Typhon. Si vous utilisez le Typhon, vous devriez d'ailleurs tenir compte du fait que vous pouvez vous servir d'une unité à disquette (drive). Ainsi, vous avez la possibilité d'enregistrer votre programme BASIC assez souvent et diminuer de cette manière le risque d'un arrêt du système d'exploitation (p. ex. lorsque vous tester des instructions compliquées). Le programme compilé est tout de suite enregistré en format AMSTRAD. Selon le genre de dispositif de sortie que vous choisissez (!TAPE/!DISC,SPEED WRITE), il s'agit d'un enregistrement sur cassette ou sur disquette.Vous devez donc chosir votre dispositif de sortie avant le processus de compilation.

Pour résoudre le problème de l'utilisation d'une unité à disquette, le compilateur compile toutes les instructions RSX. Cela veut dire que le Typhon n'utilise pas seulement la totalité du AMSDOS, mais qu'il peut aussi compiler toute sorte d'instructions RSX.
Celles-là doivent avoir les caractéristiques suivantes:

elles doivent se trouver en mémoire et elles doivent déja être initialisées au moment de la compilation

elles doivent se trouver en mémoire au moment de l'exécution du programme

elles ne doivent pas faire partie du groupe "!BASIC" (donc initialisation ROM), car celles-ci sont exécutées immédiatement au moment de la compilation

En ce qui concerne l'unité à disquette (et d'autres ROMs supplémentaires), le compilateur s'occupe lui-même de ce que ces caractéristiques soient garanties, si les appareils nécessaires sont branchés et allumés (l'unité à disquette est donc allumée). Ci-dessous, vous trouvez d'abord une énumération des instructions de disquette qui peuvent être compilées sans problèmes.

Les instructions suivantes sont acceptées aussi par l'unité à disquette:

CAT
CLOSEIN
CLOSEOUT
INPUT #9
LOAD
OPENIN
OPENOUT
PRINT #9
RUN ST1
SAVE

Les restrictions nommées dans cette documentation au chapitre I sont également vraies pour l'utilisation d'une unité à disquette.

Les instructions suivantes coopèrent également avec le compilateur comme cela est décrit dans le manuel AMSDOS:

```
!CPM
!DISC
!DISC.IN
!DISC.OUT
!TAPE
!TAPE.IN
!TAPE.OUT
!A
!B
!DRIVE,N1
!USER,N1
'DIR
```

Des adaptations ne sont nécessaires que pour des instructions qui ont pour paramètres des chaînes de caractères, car des chaînes de caractères sont enregistrées par le compilateur d'une manière différente que par le BASIC. C'est pour cela que " " ne fournit pas de valeur qui puisse être utilisée par DOS quand il s'agit de chaînes de caractères compilées. Les paramètres nécessaires peuvent être produits par la fonction "↑". Les instructions suivantes se servent également de cette fonction sur disquette:

```
 DIR,↑ST1
!ERA,↑ST1
```

Il n'y a plus de problèmes que pour !REN. !REN a besoin de 2 chaînes de caractères comme paramètres, mais vous n'en pouvez pas produire qu'un seul à l'aide de "↑". Le deuxième paramètre doit donc être produit par une des méthodes expliquées ci-dessous:

a.)
```
POKE &4000,LEN(ST2)
POKE W(&4001),@ST2
!REN,↑ST1,&4000
```

b.)
```
ST3=CHR$(LEN(ST2))+CHR$((@ST2)AND 255)+CHR$((@ST2)\256)
'REN,↑ST1,@ST3
```

XI. Particularités concernant le CPC 664 et 6128

Sur le CPC 664 et le 6128, le compilateur peut encore traiter les instructions supplémentaires suivantes:

COPYCHR$	Voir BASIC.
CURSOR M1,M2	Voir BASIC.
FILL M1	Voir BASIC.
FRAME	Voir BASIC.
GRAPHICS PAPER	Voir BASIC.
GRAPHICS PEN	Voir PLOT PEN.
MASK M1,M2	Voir BASIC.

XII. Exemples pour l'utilisation du Typhon

Dans ce chapitre, vous trouvez des exemples pour des programmes qui ont été écrits à l'aide du Typhon. Peut-être, vous pouvez même vous servir de quelques unes de nos idées.

Le programme suivant p.ex. démontre, comment on se sert des instructions de graphique et comment on évite le préfixe négatif devant des variables.

```
10 '** Demonstration for normal graphic-commands
20 '** Clear screen
30 FOR N=1 TO 3
40 IF N>1 THEN A=40 ELSE A=0
50 MODE 2:CLG N-1
60 ORIGIN 320,218
70 '** Draw frames
80 PLOT PEN N
90 FOR T=0 TO 180 STEP 10
100 MOVE A,T:DRAW T,0-A:DRAW 0-T,0-A
110 DRAW 0-A,T:DRAW A,T
120 IF N<3 THEN A=A+10 ELSE A=A+5
130 NEXT T
140 '** Draw cross
150 IF N=1 THEN MOVE 180,180:DRAW -180,-180:MOVE -180,180:DRAW 180,-180
160 CALL &BB06:NEXT N
170 STOP
```

L'instruction DEG (CIRCLE sur d'autres ordinateurs) est utilisée dans les programmes suivants pour la production de ce qu' on appèle graphique Moirée.

```
10 '** Create Moiree-graphics using circles
20 MODE 0
30 FOR r=4 TO 255 STEP 4
40 DEG 320,200,r
50 NEXT r
60 CALL &BB06:END
```

Le programme suivant est une modification intéressante de l'antérieur.

```
10 '** Create several Moiree-graphics using circles
20 MODE 2
30 FOR n=1 TO 127 STEP 2
40 DEG 30,30,n:DEG 286,30,n:DEG 542,30,n:DEG 30,286,n:DEG 286,286,n:DEG 542,286,
n
50 NEXT n
60 END
```

L'instruction RAD peut remplir toutes les figures compliquées comme démontre le programme suivant:

```
10 '** Draw circles
20 PLOT PEN 1:MODE 1
30 FOR m=80 TO 560 STEP 120
40 FOR n=0 TO 120 STEP 10
50 DEG m-1,(n*3)/2,n
60 DEG m+1,(n*3)/2,n
70 DEG m-1,400-(n*3)/2,n
80 DEG m+1,400-(n*3)/2,n
90 NEXT n
100 NEXT m
110 '** Fill lines of circles
120 MERGE &31,&6000'Stackpointer in free area of RAM
130 RAD 3
140 MERGE &31,&C000'Stackpointer back to normal
150 END'No information on Stack => jump to BASIC-Ready
```

Probablement, vous ne savez pas exactement, ce que signifient les instructions spéciales de son du Typhon. Donc, nous vous donnons quelques exemples pour les expliquer:

Le programme suivant vous démontre, comment on peut réaliser l'instruction ON SQ sur le CPC464.

```
10 '## SQ-Demo for asynchronous events
20 GOSUB 40
30 ON BREAK STOP:GOTO 30
40 SOUND 1,RND(400)+1
50 IF SQ(1)>0 THEN 40 ELSE ON SQ(1) GOSUB 40'Branch back if sound-stack empty
60 RETURN
```

Le programme suivant vous donne un exemple de l'utilisation de l'instruction SOUND TO du Typhon.

```
10 '## Demonstration for hardware-sound
20 FOR A=0 TO 15:PRINT"Je joue courbe enveloppante No. : ";A
30 SOUND TO 13,A
40 SOUND TO 12,15
50 SOUND TO 11,255
60 SOUND #1,5,5,16
70 CALL &BB06'Wait for key
80 NEXT A
90 PRINT"Courbe enveloppante No. 14 - un peux    plus vite et avec bruit"
100 SOUND TO 13,14
110 SOUND TO 12,0
120 SOUND TO 11,255
130 SOUND #1,7,7,16
140 SOUND TO 7,55
150 END
```

Le programme suivant vous montre, qu'on peut aussi faire de la musique à l'aide du Typhon.

```
10 '** Demonstration for SOUND
20 '** Read melody
30 ok=1
40 DIM a(100,3,2)
50 pt=0
60 READ a(pt,1,1):IF a(pt,1,1)=999 THEN 90
70 READ a(pt,2,1),a(pt,3,1)
80 pt=pt+1:GOTO 60
90 pt=0
100 READ a(pt,1,2):IF a(pt,1,1)=999 THEN 140
110 READ a(pt,2,2),a(pt,3,2)
120 pt=pt+1:GOTO 100
130 '** Start playing
140 p1=0:p2=0:GOSUB 230:GOSUB 280
150 PRINT"Je joue la premiere partie du ":PRINT"DEPOSUIT de Gilles Binchois":E
ND
160 '** One channel
170 DATA 4,10,4,4,11,2,4,10,2,5,3,2,4,11,2,5,1,4,5,3,3,0,0,1,5,3,1,0,0,1,5,3,2,5
,1,4,4,11,4,4,10,4,0,0,4,5,3,6,5,1,2,5,6,4,5,5,4,5,1,1,0,0,1,5,1,2,5,3,2,5,3,4,5
,6,3,5,5,2,5,3,1,5,6,4,0,0,4,5,5,2,5,3,1,5,1,1,4,11,2,4,10,1,4,11,1,5,1,2,4,5,2,
4,10,2,4,8,2
180 DATA 4,6,4,0,0,2,4,11,2,4,10,2,5,1,3,4,11,1,4,8,2,4,11,2,5,1,2,5,3,2,5,6,2,5
,5,1,5,3,4,5,2,2,4,12,1,5,3,16,999
190 '** Second channel
200 DATA 3,10,4,3,6,2,3,3,2,3,6,2,3,8,2,3,5,4,3,3,3,0,0,1,3,3,1,0,0,1,3,3,2,3,6,
4,3,8,4,3,3,8,3,6,6,3,5,2,3,3,4,3,1,4,3,10,4,3,12,2,4,1,2,3,10,2,3,6,2,3,8,4,3,6
,4,0,0,4,3,1,2,3,5,2,3,3,4,3,1,2,4,1,4,3,11,2,3,10,2,3,6,2,3,10,2,3,8,2,3,6,2,3,
5,2,3,3,2,3,8,4
210 DATA 3,6,2,3,3,4,3,1,2,3,6,2,3,5,4,3,3,16,999
220 '**Play tone
230 IF a(p1,1,1)=999 THEN SOUND #1,0,0,0:RETURN
240 IF a(p1,1,1)=0 THEN SOUND #1,0,0,0 ELSE SOUND #1,A(P1,1,1)+ok,A(P1,2,1),12
250 AFTER A(P1,3,1)*11 GOSUB 230
260 P1=P1+1
270 RETURN
280 IF a(p2,1,2)=999 THEN SOUND #3,0,0,0:RETURN
290 IF a(p2,1,2)=0 THEN SOUND #3,0,0,0 ELSE SOUND #3,A(P2,1,2)+ok,A(P2,2,2),12
300 AFTER A(P2,3,2)*11,3 GOSUB 280
310 P2=p2+1
320 RETURN
```

Certainement, vous utiliserez le Typhon aussi pour des jeux. En partant du programme suivant, vous pouvez p.ex. développer facilement un jeu D3.

```
10 '** Define colours
20 DIM F%(15)
30 FOR N=0 TO 15
40 a=(54*N)/17:IF n<9 THEN F%(N)=a ELSE f%(n)=51-a
50 INK N,F%(n):BORDER 26-((27*N)/16)
60 NEXT N
70 '** DRAW picture
80 MODE 0
90 FOR N=0 TO 266 STEP 2
100 MOVE (n*6)/5,400-n,(n/6)AND 15:DRAW 640-(n*6)/5,400-n:DRAW 640-(n*6)/5,n/2:D
RAW (n*6)/5,n/2:DRAW (n*6)/5,400-n
110 NEXT N
120 '** Change colours
130 OFF%=0
140 FOR N%=0 TO 15
150 INK N%,F%((N%+OFF%)AND 15)
160 NEXT N%:ON BREAK STOP:DELETE 2:OFF%=OFF%+1:GOTO 140
```

Elargissez un peu le programme suivant, et vous obtiendrez un programme de jeu connu , qui se sert du traitement multitâche du Typhon (choisissez des évènements synchrones , s.v.p.)

```
10 '## Multitasking-demonstration
20 '## Choose synchronous events !
30 '** Prepare screen
40 MODE 1
50 BORDER 26
60 x=RND(312)*2
70 y=RND(190)*2+14
80 sx=2
90 sy=2
100 c=0
110 TAG
120 '** Initialize events
130 GOSUB 420
140 EVERY 1,0 GOSUB 290
150 EVERY 4,1 GOSUB 360
160 EVERY 5,2 GOSUB 400
170 '** Main loop
180 CONT:ON BREAK GOTO 260
190 'Move bat
200 d=0:IF INKEY(1)<>-1 THEN d=2
210 IF INKEY(8)<>-1 THEN d=d-2
220 IF (xc+d)>0 AND (xc+d)<590 THEN xc=xc+d
230 PLOT xc,16,0:MOVE xc+2,16:DRAWR 46,0,2:PLOT xc+50,16,0
240 GOTO 180
250 '** Clear buffers and return to BASIC
260 CLEAR INPUT:a=REMAIN(0)+REMAIN(1)+REMAIN(2)+SQ(4)
270 END
280 '** Move ball
290 IF TEST(x+6,y-16)=2 THEN sy=0-sy
300 MOVE x,y,1:PRINT"o";
310 IF (y+sy>400) OR (y+sy)<14 THEN sy=0-sy
320 IF (x+sx>624) OR (x+sx)<0 THEN sx=0-sx
330 x=x+sx:y=y+sy
340 RETURN
350 '** Change ink
360 INK 1,c+11
370 c=(c+1)AND 15
380 RETURN
390 '** Set pixel
400 PLOT RND(640),RND(400),3:RETURN
410 '** Play tone
420 IF s=1 THEN a=239
430 IF s=2 THEN a=190 ELSE a=159
440 IF s=0 THEN a=119
450 SOUND 4,a
460 s=(s+1)AND 3
470 ON SQ(4) GOSUB 420
480 RETURN
```

Le programme suivant fournit la base pour un jeu de caverne connu (sur la base de l'instruction POS).

```
10 '## Demonstration for sideways-scrolling
20 BORDER 0:INK 0,0:INK 1,26:INK 2,18:INK 3,9
30 RESTORE
40 '** Loop
50 READ a:IF a=999 THEN 30
60 READ b
70 '** Scroll
80 POS 1
90 TAN
100 '** Fill tree areas
110 ORIGIN 0,0,624,640,0,a:CLG 2
120 ORIGIN 0,0,624,640,b,400:CLG 3
130 TAN
140 ON BREAK STOP
150 GOTO 50
160 '** Datas for caves
170 DATA 100,300,106,306,112,306,118,306,130,300,142,294,154,290,186,286,226,290
,208,280,236,310,220,326,230,310,210,320,190,330,170,340,150,350,130,360,110,370
,90,380,70,390,50,394,34,396,20,392,22,398,30,380,34,382,52,370,50,358,68,348,70
,330,80,320
180 DATA 90,310,92,308,94,306,96,304,98,302,100,300,98,302,96,304,98,302,100,300
,90,290,80,270,60,240,40,228,50,220,36,240,34,256,30,250,40,266,36,280,26,290,16
,300,20,310,30,330,40,320,50,330,46,310,60,300,70,316,80,314,86,312,90,310,94,30
6,96,304,98,302
190 DATA 999,999
```

Au chapitre VII, nous avions dit que le Typhon peut très bien être utilisé pour la production de RSX. Par le programme suivant, nous vous en donnons la preuve.

Ce programme définit p.ex. l'instruction !TEXTCOPY, à l'aide de laquelle vous pouvez produire une copie sur papier en caractères:

```
10 '## Hardcopy with chars
20 DEF DATA textcopy=30
30 END
40 LF$=CHR$(13)+CHR$(10)'Code for linefeed
50 XP=POS(#0)
60 YP=VPOS(#0)
70 '** Copy-loop
80 FOR N=1 TO CLS(#0)
90 FOR M=1 TO WIDTH(#0)
100 LOCATE #0,M,N
110 r$=READ CHR$(#0)
120 ON BREAK GOTO 160:IF PRINT THEN PRINT#8,r$; ELSE 120
130 NEXT M
140 ON BREAK GOTO 160:IF PRINT THEN PRINT#8,LF$; ELSE 140
150 NEXT N
160 LOCATE #0,XP,YP
170 '** Restore screen-position and return to main program
180 RETURN
```

Egalement le programme suivant peut être utilisé pour des copies sur papier. Il travaille dans le mode de graphique et remplace de différentes couleurs par de différentes trames.

```
10 '** Hardcopy with different patterns
20 DIM c(15,2)
30 FOR n=0 TO 15
40 READ c(n,1),c(n,2)
50 NEXT n
60 sp$=CHR$(27)+"A"+CHR$(7)'Linefeed of 7 pixels
70 gr$=CHR$(27)+"L"+CHR$(127)+CHR$(2)'639 pixels graphics-mode
80 lf$=CHR$(13)+CHR$(10)'Linefeed
90 DEF DATA shadedcopy=100
100 END
110 '** Copy
120 ORIGIN 0,0
130 ON BREAK STOP:IF PRINT THEN PRINT#8,sp$; ELSE 130
140 FOR y=399 TO 0 STEP -14
150 ON BREAK STOP:IF PRINT THEN PRINT#8,gr$; ELSE 150
160 FOR x=0 TO 638
170 GOSUB 230
180 PRINT#8,CHR$(z);
190 NEXT x
200 PRINT#8,lf$;
210 NEXT y
220 RETURN
230 '** Read 7 pixels from screen and decode them
240 z=0:h=64:FOR n=0 TO 7
250 p=TEST(x,y-n*2)
260 i=c(p,((y/2-n)AND 1)+1)
270 b=x AND 7
280 k=1:FOR m=1 TO b
290 k=k+k
300 NEXT m
310 IF (i AND k) THEN z=Z+h
320 h=h/2
330 NEXT n
340 RETURN
350 '** Patterns - binary encoded
360 DATA 0,0,255,255,255,&x1010101,&x10101010,&x1010101,0,&x1010101,255,127,255,
119,247,127,255,51,119,&x1101110,204,51,136,34,136,68,0,&x10101010,128,8,136,0,0
,128
```

Beaucoup de clients désirent se servir d'un compilateur pour la production d'une routine de tri dans un programme de traitement de fichier. Nous avons programmé une routine !SORT, qui réalise très vite des processus de tri alphabétique pour une matrice de chaînes de caractères du BASIC. Le programme compilé est appelé par !SORT,@variable$(0)

```
10 DEF DATA sort=30
20 END
30 'SHELL-METZNER - sort
40 DEF READ AD'Get address of array
50 N=PEEK W(AD-2)'Read number of elements
60 M=N
70 M=M/2
80 IF M=0 THEN RETURN
90 J=1
100 K=N-M
110 I=J
120 L=I+M
130 S1=I+I+1+AD-3:S2=L+L+L+AD-3'Get addresses of string-descriptors
```

```
140 a=PEEK(s1):c=PEEK(s2):a1=PEEK w(s1+1)-1:a2=PEEK w(s2+1)-1:IF a>c THEN cn=a E
LSE cn=c'Length and addresses of strings
150 '** Test, if first string less than second
160 FOR mm=1 TO cn
170 y=PEEK(a1+mm):z=PEEK(a2+mm):IF y<z THEN 250
180 IF y=z THEN NEXT mm:IF a<=c THEN 250
190 '** Exchange string-descriptors
200 POKE S1,PEEK(S2):POKE W(S1+1),PEEK W(S2+1)
210 POKE S2,A:POKE W(S2+1),a1+1
220 '** Next element
230 I=I-M
240 IF I<1 THEN 250 ELSE 120
250 J=J+1
260 IF J>K THEN 70 ELSE 110
```

Le programme que vous voyez ci-dessus, peut être expliqué à l'aide du programme suivant; mais ne le compilez pas!

```
10 '## Sort-demonstration
20 '## Do not compile this program
30 '** Prepare array
40 DIM a$(200),b$(200)
50 FOR n%=0 TO 200
60 FOR m%=1 TO 3
70 a$(n%)=a$(n%)+CHR$(65+RND*60)
80 NEXT m%
90 b$(n%)=a$(n%)
100 NEXT n%
110 '** Sort
120 PRINT"realisation de tri";CHR$(13);
130 a=TIME
140 !SORT,@a$(0)
150 B=TIME-a
160 PRINT "fin de tri apres secondes:";b/300
170 '** Print new and old order
180 FOR n%=0 TO 200
190 PRINT b$(n%),a$(n%)
200 NEXT n%
```

Enfin, nous aimerions offrir aux programmateurs professionnels quelques exemples d'intégration de code machine dans des programmes compilés.

Le programme suivant produit des instructions RSX à l'aide desquelles vous pouvez fixer séparément les couleurs dans 4 parties différentes de l'écran et, de cette manière, produire toutes les 27 couleurs en MODE 0 simultanément sur l'écran. Malheureusement, cela ne fonctionne pas sur le CPC 464, car celui-ci refuse de traiter des routines rapides d'interruption avec des évènements synchrones.

De toute façon, ce programme définit les instructions !INK et !BORDER. Celles-là changent les couleurs de la seconde et de la quatrième partie de l'écran. Les parties 2,3,4 sont d'ailleurs désignées par 1,2,3. Par !INK,Pen,partie de l'écran,couleur, vous déterminez la couleur d'un PEN dans une partie de l'écran. Par !BORDER, partie de l'écran, couleur on définit la couleur de marge d'une partie de l'écran.

```
10 '## Multicolour - screen
20 a$="":SPEED INK 1,1
30 cx=1
40 '** Initialize fast ticker
50 AFTER 100 GOSUB 200
60 a=REMAIN(0)
70 IF PEEK(&BB01)=&E0 THEN ad=&AC69 ELSE ad=&AC4F
80 a2=PEEK w(ad)
90 a1=@a$+2:MERGE &2A,a1,6,193,&ED,&5B,a2,&CD,&BCEF
100 a=CREAL(&BCE3,0,@a$)
110 '** Initialize PENs
120 DIM f(26),c(26)
130 FOR n=0 TO 26:READ c(n):NEXT n
140 FOR n=0 TO 16
150 READ p:GOSUB 280:POKE @f(0)+n,co:POKE @f(0)+n+17,co:POKE @f(0)+n+34,co
160 NEXT n
170 DEF DATA ink=320,border=340,init=10
180 END
190 '** Called on interrupts
200 IF (INP(&F500)AND 1)=1 THEN cx=1 ELSE cx=cx+1
210 ON cx GOTO 220,220,240,250,260,220
220 RETURN
230 '** Show colours
240 f=@f(0)+16:GOTO 270
250 f=@f(0)+33:GOTO 270
260 f=@f(0)+50
270 MERGE &2A,f,&1E,&10,1,&7F00,&ED,&59,&7E,&ED,&79,&2B,&1D,&F2,FRE-10,&C9'MC to
r colour-palette
280 '** Encode colour
290 co=c(p)
300 RETURN
310 '** Table for encoding colours
320 DATA 84,68,85,92,88,93,76,69,77,86,70,87,94,64,95,78,71,79,82,66,83,90,89,91
,74,67,75
330 '** Table for colours after initialisation
340 DATA 1,24,20,6,26,0,2,8,10,12,14,16,18,22,23,24,1
350 '** Set ink
360 DEF READ pe,ab,p
370 GOTO 400
380 '** Set border
390 DEF READ ab,p:pe=16
400 GOSUB 280
410 POKE @f(0)+pe+ab$17-17,co
420 RETURN
```

Le programme suivant se sert de ces instructions supplémentaires

```
10 '## Demostration for multicolour-screen
20 '## Do not compile this program !
30 '** Prepare ten windows
40 INK 15,26:|INK,15,1,26:|INK,15,2,26:|INK,15,3,26
50 MODE 0
60 FOR n=0 TO 9
70 WINDOW #7,n*2+1,n*2+3,1,25
80 PAPER #7,n:CLS #7
90 PEN #7,15:LOCATE #7,1,1:PRINT #7,USING"##";n:LOCATE #7,1,7:PRINT #7,USING"##"
;n+10:IF n<7 THEN LOCATE #7,1,13:PRINT #7,USING"##";n+20
100 NEXT n
110 '** Show all 27 colours
120 FOR n=0 TO 9
130 INK n,n
140 |INK,n,1,n+10
150 IF n<7 THEN |INK,n,2,n+20 ELSE |INK,n,2,0
160 |INK,n,3,0
170 NEXT n
180 GOTO 180
```

Le programme suivant représente un programme utilitaire pour disquette.

```
10 DEF DATA formatd=30,formatv=50,messageoff=180,messageon=210,testdrive=240,ret
rynumber=310,readsector=350,writesector=380
20 END
30 a=&C1'|FORMATD   format data only
40 GOTO 60
50 a=&41'|FORMATV   format vendor
60 c=a
70 a$=""
80 FOR n=0 TO 32 STEP 4
90 POKE @a$+n+1,0:POKE @a$+n+2,c:POKE @a$+n+3,2
100 c=c+2:IF(c AND &F)>9 THEN c=c-9
110 NEXT n
120 f$=CHR$(&52)+CHR$(&C6)+CHR$(7)
130 FOR n=0 TO 39
140 FOR m=0 TO 32 STEP 4:POKE @a$+m,n:NEXT m
150 MERGE &3A,n,&57,&1E,0,&3A,a,&4F,&21,a$,&DF,f$
160 NEXT n
170 RETURN
180 '|MESSAGEOFF ; messages off
190 POKE &BE78,255
200 RETURN
210 '|MESSAGEON ; messages on
220 POKE &BE78,0
230 RETURN
240 'Test for disc mounted ; form :|TESTDRIVE,drive,@variable%
250 DEF READ lw,ad
260 f$=CHR$(&30)+CHR$(&C6)+CHR$(7)
270 MERGE &3A,lw,&DF,f$,&32,f$
280 IF (ASC(f$)AND 223)>1 THEN x=0 ELSE x=-1
290 POKE w(ad),x
300 RETURN
310 'Change number  of reads before error-message (using|RETRYNUMBER,n )
320 DEF READ n
330 POKE &BE66,n
340 RETURN
350 CLEAR 'Read sector (form:|READSECTOR,drive,512-byte-buffer,sector,track)
360 f$=CHR$(&66)+CHR$(&C6)+CHR$(7)
370 GOTO 400
380 CLEAR 'Write sector (form:|WRITESECTOR,drive,512-byte-buffer,sector,track)
390 f$=CHR$(&4E)+CHR$(&C6)+CHR$(7)
400 DEF READ lw,bf,sc,tr
410 IF lw=0 THEN |A:st=sc-1+PEEK(&A89F) ELSE |B:st=sc-1+PEEK(&A8DF)
420 MERGE &3A,lw,&5F,&3A,st,&4F,&3A,tr,&57,&2A,bf,&DF,f$
430 RETURN
```