BASIC 8K NASCOM

MANUEL DE PROGRAMMATION

TRADUCTION JCS COMPOSANTS

S O M M A I R E

1. GENERALITES

1.1 INTRODUCTION

Le BASIC de NASCOM est le Basic développé par Microsoft, auquel il a été ajouté quelques possibilités supplémentaires :

- il est compatible directement avec les moniteurs T2, T4 et Nas-sys,
- Nas-sys permet la correction de programmes et des données, grâce à ses fonctions puissantes d'édition,
- l'interface avec une imprimante est réalisée directement par l'E/S série, sans circuits supplémentaires,
- la lecture et l'écriture de cassette sont effectuées avec un contrôle de validité,
- des instructions graphiques sont incorporées.

Il permet des calculs en virgule flottante sur des nombres compris entre 1,70141 E38 à 2,9387 E-38, avec 7 chiffres significatifs. Il comprend les fonctions trigonométriques usuelles, les opérations sur les chaînes de caractères et le contrôle du PIO.

1.2 CONVENTIONS ADOPTEES

Dans ce qui suit, on a adopté les conventions suivantes :

- les ordres et les instructions en majuscules sont entrés en machine comme il est écrit,
- les données entre parenthèses < > doivent être fournies comme il est dit dans le texte. Ce qui est entre les parenthèses n'est pas obligatoire. Ce qui est suivi par des points ... peut être répété ou supprimé selon le besoin,
- les caractères devant être tapés en appuyant également sur la touche shift ou control sont précédés de Shift/ ou Control/,
- toute la ponctuation doit être entrée comme il est indiqué.

1.3 QUELQUES DEFINITIONS IMPORTANTES

- Caractère alphanumérique : lettre ou chiffre
- "Enter", "Newline", "Retour chariot" ou CR : concerne la touche du terminal qui provoque le déplacement de la tête d'impression ou du curseur au début de la ligne suivante.
- Ordres et Instructions :
  Les ordres sont des instructions qui sont en principe utilisées en mode direct (voir 1.4). Généralement, les ordres peuvent être utilisés comme des instructions à l'intérieur des programmes, à l'exception des ordres qui font sortir du programme pour rendre la main au Basic.
  Certains ordres, par exemple CONT, n'ont pas de sens dans un programme, alors que des expressions telles que DEF ne peuvent être utilisées que dans les programmes.
- Edition : processus de correction de lignes d'un programme ou de préparation d'une ligne de donnée, selon un certain format avant sa sortie sur un périphérique,
- Mot réservé : mots qu'il n'est pas permis d'utiliser dans la programmation, car ils ont une signification bien précise pour l'interpréteur Basic,
- Chaîne littérale : chaîne de caractères entre guillemets ", qui doit être entrée ou sortie comme il est indiqué entre les guillemets. Ceux-ci ne sont pas pris en compte.

1.4 MODES D'OPERATION

Le Basic permet deux modes d'opération. Dans le mode direct, les ordres et les instructions sont exécutés au moment de l'entrée, les résultats des opérations arithmétiques et logiques sont conservés en mémoire, mais les ordres ou instructions sont perdus après leur exécution.

Dans le mode indirect, les lignes d'instruction sont numérotées et conservées en mémoire. L'exécution commence après l'entrée de l'ordre RUN.

1.5 STRUCTURE DES LIGNES DE BASIC

Les lignes sont constituées d'un numéro de ligne, d'un espace, puis d'une ou de plusieurs instructions ou ordres. Dans le cas d'instructions multiples sur la même ligne, elles sont séparées par deux points :
Chaque numéro de ligne donne l'adresse de la ligne en mémoire; l'exécution des instruction sur chaque ligne se fait de la gauche vers la droite. Les numéros de ligne sont aussi les étiquettes de branchement. Ils peuvent être compris entre 0 et 65529. Il est recommandé de programmer en incrémentant les lignes de 5 en 5 ou de 10 en 10.

Les instructions sont identifiées par leur nom et quelquefois suivies d'arguments. Chaque ligne peut comprendre autant d'instructions séparées par : que l'on veut, mais ne doit pas dépasser 72 caractères.

Si Nas-sys est utilisé en mode normal, les lignes ne peuvent pas excéder 48 caractères. Cependant, on peut accepter 72 caractères en utilisant le mode Moniteur (ordre XO), le retour au Basic se faisant en tapant Z.

L'instruction REM permet d'insérer des remarques dans le programme :

```
REM  < remarque >
```

Cette instruction n'est pas exécutée, mais les branchements peuvent se faire sur l'étiquette correspondante.
Attention : la fin de l'instruction REM doit obligatoirement être une fin de ligne ou un retour chariot. Ainsi :

```
100 REM EFFECTUER CETTE BOUCLE : FOR I = 1 TO 10
                  Instruction FOR non exécutée
101 FOR I = 1 TO 10 ; REM EFFECTUER CETTE BOUCLE
                  Instruction FOR exécutée
```

Lorsque l'interpréteur Basic détecte une erreur qui arrête l'exécution du programme, il affiche un message d'erreur :

```
ordre direct              ?XX ERROR
instruction indirecte     ?XX ERROR IN nnnn
```

où XX est le code d'erreur (voir paragraphe 6.5) et nnnn est le numéro de ligne.

1.6 EDITION DES PROGRAMMES

Les possibilités suivantes sont offertes avec Nasbug T2, Nasbug T4 et Nas-sys en mode X0 (terminal externe) :

- correction de caractères : appuyer sur la touche Backspace jusqu'à supprimer l'erreur sur la ligne et retaper la fin de la ligne,
- correction de ligne : une ligne en cours de frappe peut être annulée en entrant le signe @ au lieu d'effectuer le retour chariot. Un retour chariot est imprimé automatiquement après la suppression de la ligne. L'effet est identique en entrant Control/U.
  Une ligne entrée avec un numéro de ligne déjà existant se substituera à la précédente.
- correction de programmes entrés : l'ordre NEW provoque l'effacement du contenu de toute la mémoire. On l'utilise généralement avant toute entrée de nouveau programme.

1.7 UTILISATION DU MONITEUR NAS-SYS

L'édition de programme peut se faire en utilisant toutes les possibilités de Nas-sys. Les lignes affichées sont passées à l'interpréteur Basic après appui sur Enter ou New-line. On peut donc lister un programme, l'éditer, et l'entrer à nouveau. Il en est de même des données.

2. EXPRESSIONS ET INSTRUCTIONS

2.1 LES EXPRESSIONS DU BASIC

2.1.1. Constantes

Les constantes peuvent être des entrées, des nombres en virgule flottante ou des chaînes de constantes (voir 4.1).
Exemple :

```
123
3.141
0.0436
1.25E+05
```

Quelle que soit la longueur des données numériques entrées, seuls les 7 premiers caractères seront pris en compte.
Ainsi

```
PRINT 1.2345678901     s'affiche
1.23457
OK
```

Le format d'un nombre imprimé est le suivant :

- si le nombre est négatif, le signe - est imprimé à gauche du chiffre. Le signe + est remplacé par un espace.
- si un entier est compris entre 0 et 999999, il est imprimé comme un entier,
- si la valeur absolue d'un nombre est plus grande ou égale à 0.01 et plus petite que 999999., il est imprimé avec une virgule (en fait un point) et sans exposant.
- si le nombre n'entre pas dans les deux cas précédents, il est imprimé en notation scientifique
  SX.XXXXXESTT
  S est le signe de la mantisse et de l'exposant, X les caractères de la mantisse et T ceux de l'exposant.
  L'exposant doit être compris entre -38 et +38, et le plus grand nombre qui puisse être utilisé est 1,70141E38, le plus petit 2.9387E-38.
  Exemple de représentation :

Nombre +1	Affichage 1
-1	-1
6523	6523
1E20	1E20
-12.3456E-10	-1.23456E-09
1.234567E-7	1.234567E-07
100000	1E+06
.1	.1
.01	.01
.000123	1.23E-04
-25.460	-25.46

Un espace est toujours imprimé après le nombre. Dans le cas où le nombre ne tient pas sur la ligne, un retour chariot est effectué et le nombre imprimé à la ligne suivante.

2.1.2. Variables

Une variable est la représentation symbolique d'un nombre qui lui est assigné. Sa valeur peut être explicitement donnée par le programmeur, soit calculée. En l'absence de toute spécification, la valeur 0 lui est attribuée.
Une variable peut avoir un nom aussi long que l'on veut, mais seuls les deux premiers caractères sont significatifs.
Le premier caractère du nom doit être une lettre.
Le nom ne doit pas comporter de mots réservés. Exemple :

Permis :		Non permis	
	A		%A
	Z1		
	TP		T0
	PSTG$		
	COUNT		RGOTO

2.1.3. Tableaux de variables

Il est souvent utile de représenter des variables par un même nom et de travailler sur des matrices.
Les tableaux de variables peuvent avoir plusieurs dimensions et ne sont limités que par l'espace mémoire disponible.
Forme VV (< indice > [, < indice > ...])
VV est le nom de la variable, et les indices des expressions entières. Les indices peuvent être entre parenthèses () ou [] . Le nombre de dimension est limité par la capacité d'une ligne et par le volume mémoire. Le plus petit indice possible est 0

A(5) est le sixième élément du tableau A, le premier étant 0

TABLEAU (I,2*J) Si la valeur de I vaut 3 et J vaut 2.4, la seconde expression sera transformée en entier par troncation. La variable sera TABLEAU (3,2)

L'instruction DIM alloue la mémoire des tableaux et les initialise à zéro.

DIM VV (< indice > [, < indice > ...])

VV est un nom permis de variable, et les indices les plus grands qui seront éventuellement rencontrés dans l'exécution du programme. Plusieurs tableaux peuvent être définis par la même instruction. Exemple :

```
113 DIM A(3), Q$(2,2,2)
114 DIM R2%(4),B(10)
115 DIM Q1(N),Z**(2+I)
```

Les expressions entre parenthèses sont évalués et tronqués au moment de l'exécution.
En l'absence de DIM avant une exécution de tableau, une valeur par défaut de 10 est donnée comme valeur de chaque dimension (soit 11 éléments).
Un message d'erreur BS ou SUBSCRIPT OUT OF RANGE sera affiché si l'indice évolue au-delà de la dimension prévue, ou si le nombre de dimensions ne correspond pas à celui annoncé par DIM.
Le message DD ou REDIMENSIONED ARRAY correspond à un re-dimensionnement de tableau, par exemple lorsque DIM est rencontré après un dimensionnement des tableaux par défaut à 10.

2.1.4. Opérations arithmétiques et logiques

L'ordre de prise en charge des opérateurs peut être imposé par l'usage de parenthèses (). En leur absence, les opérations d'une expression seront effectués dans l'ordre suivant :

()	parenthèses
↑	exponentiation
-	négation
*,/	multiplication, division
+,-	addition, soustraction

opérateurs relationnels

=	égal
< >	différent
<	plus petit que
>	plus grand que
<=, =<	plus petit que ou égal à
>=, =>	plus grand que ou égal à

NOT	non logique
AND	et logique
OR	ou logique

Les opérateurs relationnels peuvent être utilisés dans toutes les expressions. Ils ont la valeur -1 (vrai) ou 0 (faux).

2.1.5. Opérations logiques

Les opérateurs logiques peuvent être utilisés pour la manipulation de bit et les fonctions algébriques Booléenne.
Les opérateurs AND, OR, NOT transforment leurs arguments en entiers complémentés à 2 de 16 bits signés, compris entre 32768 et 32767.
Les résultats sont rendus dans ce même format.
Si les arguments ne sont pas dans ces limites, un message ILLEGAL FUNCTION CALL est affiché et l'exécution s'arrête.
Le tableau ci-dessous est la table de vérité des opérateurs logiques. Les opérations sont effectuées bit à bit.

AND

X	Y	X et Y
1	1	1
1	0	0
0	1	0
0	0	0

OR

X	Y	X ou Y
1	1	1
1	0	1
0	1	1
0	0	0

NOT

X	NOT X
1	0
0	1

Exemples :

63 AND 16 = 16	63 = 111111 binaire, 16 = 100000 binaire, et 63 AND 16 = 16
15 AND 14 = 14	15 = 1111 binaire, 14 = 1110 binaire, 15 AND 14 = 1110 binaire = 14
-1 AND 8 = 8	-1 = 1111111111111111, 8 = 1000 binaire, donc -1 AND 8 = 8
4 OR 2 = 6	4 = 100 binaire, 2 = 10 binaire, donc 4 OR 2 = 110 binaire = 6
10 OR 10 = 10	10 = 1010 binaire, donc 10 OR 10 = 1010 binaire = 10
-1 OR -2 = -1	-1 = 1111111111111111, -2 = 1111111111111110, donc -1 OR -2 = -1
NOT 0 = -1	Le complément de 16 zéros est 16 nn, qui est la représentation du complément à 2 de -1
NOT X = -(X+1)	Le complément à 2 d'un nombre quelconque est le complément bit à bit plus un.

Les opérateurs logiques sont en particulier utilisés pour tester si un nombre binaire correspond à une configuration de bit pré-déterminée. Ces nombres peuvent par exemple être lus sur les ports d'entrée et correspondre à un certain état d'une sonde extérieure.

2.1.6. Instruction LET

LET permet d'assigner une valeur à une variable
LET < VV > = < expression >

VV est un nom de variable et l'expression une quelconque expression autorisée.
Exemple :
```
1000 LET V = X
110  LET I = I + 1        Le signe "=" signifie donc "remplacé par"
```
Le mot LET est cependant optionnel et l'équation suivante est autorisée :
```
120 V= .5*(X+2)
```

2.1.7. Erreur de syntaxe

Les erreurs de syntaxe sont détectées : caractère non permis dans une ligne, ponctuation erronnée, parenthèses manquantes. Un message SN ou SYNTAX ERROR est alors détecté.
De la même façon, lorsque le résultat d'un calcul dépasse la dimension permise pour les nombres, le message OV ou OVERFLOW est affiché.
Un essai de division par zéro produit le message /0 ou DIVISION BY ZERO.

2.2 BRANCHEMENTS, BOUCLES ET SOUS-PROGRAMMES

2.2.1. Branchements

Les branchements permettent de changer l'ordre d'exécution des lignes de programme :

- GO TO, branchement inconditionnel, permet de continuer l'exécution à la ligne mmmm.
  GO TO < mmmm >
- IF ... THEN, branchement conditionnel, s'écrit
  IF < expression > THEN < mmmm >
  expression étant une expression arithmétique relationnelle ou logique permise, et mmmm un numéro de ligne.
  Si l'expression a une valeur différente de zéro logique, le programme continue à la ligne mmmm. Si l'expression est nulle, l'exécution se poursuit à la ligne qui suit l'instruction IF...THEN.

  Autre formule possible :

  IF < expression > THEN < instruction >

  Exemples :

  10 IF A=10 THEN 40 — Si l'expression A=10 est vraie, branchement à la ligne 40. Sinon, exécution à la ligne suivante.

  15 IF A<B+C OR X THEN 100 — L'expression après IF est évaluée. Si l'expression est non nulle (nul logique), le branchement est fait à la ligne 100.

  20 IF X THEN 25 — Si X n'est pas nul, le branchement se fait en ligne 25.

  30 IF X=Y THEN PRINT X — Si l'expression X=Y est vraie, sa valeur n'est pas un zéro logique et l'instruction PRINT est exécuté. Sinon, elle ne l'est pas. Suite de l'exécution à la ligne suivante.

  35 IF X=Y+3 GOTO 39 — Equivalent à l'instruction IF...THEN, mais GOTO doit être suivi par un numéro de ligne et non pas par une autre instruction.

- ON ... GOTO est un autre type de branchement conditionnel.
  ON < expression > GOTO < liste de numéro de ligne >
  La valeur de l'expression est tronquée pour la transformer en entier, que nous appellerons I. Le programme se branchera sur la ligne dont l'étiquette vaut I. Le nombre d'étiquettes (de numéro de ligne) que l'on peut mettre n'est limité que par la plaque que l'on dispose sur la ligne. Si I=0 ou s'il est plus grand que les étiquettes de la ligne, l'exécution se poursuivra à la ligne suivant l'instruction ON...GOTO.
  I doit être compris entre 0 et 255 inclus, sinon le message d'erreur FC ou ILLEGAL FUNCTION CALL sera affiché.

2.2.2. Boucles

L'instruction FOR ... NEXT permet d'effectuer des opérations répétitives.
FOR < variable > = < X > TO < Y > [STEP < Z >]

X, Y et Z sont des expressions. Elles sont évaluées la première fois que l'instruction FOR est rencontrée. La valeur X est assignée à la variable. Le programme effectue ensuite les instructions qui suivent. Lorsque l'instruction NEXT est rencontrée, le pas Z est ajouté à la variable qui est listée par rapport à la valeur finale Y. Si le pas Z est positif et la variable est plus petite ou égale à la valeur finale, ou si au contraire Z est négatif, et la variable plus grande ou égale à la valeur finale Y, le programme se branche sur l'instruction qui suit immédiatement l'instruction FOR, sinon l'exécution se poursuit à l'instruction qui suit le NEXT.
Si le pas n'est pas spécifié, il est mis à 1 par défaut.
Exemple :
```
10 FOR I = 2 TO 11                  La boucle est exécutée 10 fois avec la variable I prenant la
                                    valeur 2 à 11
20 FOR V = 1 TO 9.3                 La boucle est exécutée 9 fois jusqu'à ce que V soit plus grand
                                    que 9.3
30 FOR V = 10*N TO 3.4/Z STEP SQR(R)  Les valeurs initiales, finales et le pas seront calculés une
                                    fois seulement avant la première boucle.
40 FOR V = 9 TO 1 STEP - 1          Cette boucle sera exécutée 9 fois.
```
Les boucles FOR ... NEXT peuvent être imbriquées, c'est-à-dire que la boucle à l'intérieur d'une boucle sera exécutée. Exemple :
```
100 FOR I = 1 TO 10
120 FOR J = 1 TO I
130 PRINT A (I,J)
140 NEXT J
150 NEXT I
```

La ligne 130 imprimera 1 élément de A pour I = 1, 2 éléments pour I = 2, etc ... L'instruction NEXT de la boucle interne (J ici) doit apparaître avant le NEXT de la variable de la boucle externe (I ici).
Le nombre de boucles imbriquées n'est limité que par la taille mémoire.
L'instruction NEXT a la forme suivante :

NEXT [< variable > [, < variable > ...]]

où chaque variable est la variable de boucle FOR pour laquelle le NEXT est le point terminal. Un NEXT sans variable se référera au FOR le plus récent. Dans le cas de boucles imbriquées avec un point terminal commun, un seul NEXT peut être utilisé. La première variable dans la liste doit être celle de la boucle la plus récente, la seconde variable celle de la récente de second rang, etc. Si le Basic rencontre un NEXT avant que l'instruction FOR correspondante ait été exécutée, un message d'erreur NF ou NEXT WITHOUT FOR sera affiché et le programme arrêté.

2.2.3. Sous-programmes

Un sous-programme peut être utilisé chaque fois qu'une série d'opérations identique doit être exécutée en plusieurs endroits d'un programme. Un sous-programme est une suite d'instructions exécutée après qu'un branchement ait été fait grâce à l'instruction GOSUB. L'exécution du sous-programme se termine par RETURN qui effectue le branchement sur l'instruction qui suit immédiatement le dernier GOSUB. Forme de l'instruction :

GOSUB < numéro de ligne >

numéro de ligne étant celui de la première ligne du sous-programme. Un sous-programme peut être appelé de plusieurs endroits du programme et elle peut contenir elle-même un appel à un sous-programme. Cette cascade de sous-programmes imbriqués n'est pas limitée.
Le branchement sur les sous-programmes peut se faire sous une forme conditionnelle grâce à ON ... GOSUB

ON < expression > GOSUB < liste des numéros de ligne >

L'exécution est identique à ON ... GOTO, mais les numéros de ligne sont ceux des premières lignes des sous-programmes.
L'exécution se poursuit à l'instruction qui suit ON ... GOSUB après le RETURN d'un sous-programme.

2.2.4. Erreur de dépassement de mémoire

La taille mémoire peut limiter la complexité du programme. Si la limite de capacité de la mémoire est atteinte, le message OM ou OUT OF MEMORY est affiché.

2.3 ENTREES / SORTIES

2.3.1. Input

Cette instruction correspond à une demande d'entrée d'information. Son format est :

INPUT < liste de variables >

Les valeurs entrées au terminal sont assignées aux variables de la liste. Lorsque l'instruction est exécutée, un point d'interrogation ? est affiché sur le terminal, signifiant ainsi une demande d'information. L'opérateur tape les informations demandées, séparées par des virgules et appuie sur "enter". Si l'information n'est pas valide, par exemple si une tentative d'entrée de chaîne au lieu d'un nombre est faite, le BASIC affiche REDO FROM START ? et attend l'entrée correcte des données.
Si le nombre de données entrées est insuffisant, le BASIC affiche ?? et le système attend le complément de données.
Par contre, sil le nombre de données est supérieur au nombre de données attendu, la mise en garde EXTRA IGNORED est affichée, mais l'exécution se poursuit.
Après l'entrée de toutes les données, l'exécution se poursuit à l'instruction suivante.
On peut faire apparaître un commentaire en utilisant l'instruction de la manière suivante :

INPUT [" < commentaire > " ;] < liste de variable >

Le commentaire doit être entre guillemets, et séparé de la liste par ;
Exemple : 100 INPUT "QUELLES VALEURS" ; X,Y
provoquera l'affichage de QUELLES VALEURS ?
Entrer alors dans les valeurs de X et de Y. Si un retour chariot ou "enter" est effectué après ?, les variables de la liste ne seront pas changées et l'exécution continuera.

2.3.2. Print

L'instruction PRINT provoque l'affichage sur l'écran ou l'impression sur un terminal. Sa forme est la suivante :

PRINT liste d'expression

Les chaînes littérales peuvent être prises en charge si elles sont prises entre guillemets "".
Un saut de ligne résulte de l'entrée de PRINT non suivi d'une expression.

La position de l'impression dépend de la ponctuation utilisée pour séparer les éléments de la liste. En effet, la ligne est découpée en zones de 14 espaces chacune. Une virgule provoque l'impression de l'élément suivant au début de la zone suivante de 14 emplacements. Un point virgule ; conduit l'impression suivante à se faire immédiatement après la dernière valeur imprimée. Si une virgule et un point virgule terminent la liste, l'instruction PRINT provoquera l'impression sur la même ligne. Sinon, un retour chariot est imprimé.

2.3.3. Data, Read, Restore

- DATA

Les données numériques ou les chaînes peuvent être entrées grâce à l'instruction DATA, dont le format est :

DATA < liste >

où les éléments de la liste sont des valeurs numériques ou des constantes de chaîne séparées par des virgules. La liste est mise en mémoire et elle peut être relue ensuite grâce à l'instruction READ. Exemple :

```
10 DATA 1,2,-1E3,.04
20 DATA "ARR", MASCOM
```

Les espaces au début et en fin de liste sont supprimés, sauf s'ils sont à l'intérieur de guillemets ""

- READ

Cette instruction permet d'accéder aux valeurs DATA en mémoire. Format :

READ < liste de variables >

où les noms de variables sont séparés par des virgules.
Les valeurs de la liste DATA sont assignées aux valeurs correspondantes de la liste de READ. Ce processus va de la gauche vers la droite, jusqu'à la fin de la liste READ. S'il y a plus de variables dans la liste READ que dans la liste DATA, un message d'erreur OD ou OUT OF DATA est affiché.

S'il y a plus de variables dans la liste DATA que dans READ, l'instruction READ suivante qui sera exécutée prendra la dernière valeur de la liste DATA qui n'aura pas été lue.
Une instruction READ permet d'accéder à plusieurs instructions DATA, et plus d'un READ peut accéder aux données d'un DATA.
Si une liste de DATA n'est pas conforme au format prévu, une erreur SN ou SYNTAX ERROR est affichée. Le numéro de ligne du message d'erreur correspond au numéro de ligne du DATA qui comporte l'erreur.

- RESTORE

Après exécution de cette instruction, il sera possible de lire à nouveau les données de DATA grâce à un READ.

2.2.4. Csave, Cload

Les tableaux numériques peuvent être chargés sur cassette et relus grâce à CSAVE* et CLOAD*.

CSAVE* < nom du tableau >
CLOAD* < nom du tableau >

Le tableau est écrit en binaire avec une en-tête de 4 octets (210 octal) pour indiquer le début. Cette en-tête est recherchée au chargement.
Le nombre d'octets écrit est :

4 + 4 x nombre d'éléments du tableau.

Lorsqu'un tableau est écrit ou lu, l'indice du tableau se trouvant le plus à gauche est celui qui s'incrémente le plus rapidement. Ainsi :

```
DIM A(10)
CSAVE*A
      s'écrit  A(0), A(1), ... A(10)
DIM A(10,10)
CSAVE*A
      s'écrit A(0,0), A(1,0) ... A(10,0), A(10,1), ... A(10,10)
```

On peut alors écrire un tableau à deux dimensions sur cassette et la relire dans un tableau à une dimension.
Le BASIC de NASCOM effectue un contrôle de validité par sommation. En cas d'erreur, le message BAD est affiché et il faut recommencer l'exécution du programme et lire les données à nouveau.

2.2.5. Autres entrées-sorties

- WAIT

L'état des ports d'entrée peut être suivi par cette instruction :

WAIT < I,J > [, < K >]

I étant le numéro du port et J et K des expressions en entier. L'état du port est traité avec un OR exclusif avec K, et le résultat avec un AND avec J. L'exécution est arrêtée jusqu'à ce qu'une valeur logique non nulle soit obtenue. J prend la valeur des bits du port I à tester et l'exécution est suspendue tant que ces bits sont différents de ceux de K. L'exécution reprend à l'instruction qui suit le WAIT.
Si K est omis, sa valeur est supposée être zéro.
I, J et K doivent être entre 0 et 255. Exemples :

WAIT 20,6	L'exécution s'arrête jusqu'à ce que le bit 1 ou le bit 2 du port 20 soit égal à 1 (le bit 0 est le bit de poids faible). Poursuite de l'exécution à l'instruction suivante.
WAIT 10,255,7	L'exécution s'arrête jusqu'à ce que l'un des 5 bits de poids fort du port 10 vale 1, ou que l'un des 3 bits de poids faible vale zéro.

- POKE, PEEK

Des données binaires peuvent être entrées en mémoire grâce à l'instruction POKE :

POKE < I,J >

où I et J sont des entiers. L'octet J est placé en position mémoire I. I doit être inférieur à 32368 et J doit être compris entre 0 et 255. Les données peuvent être placées au-delà de l'emplacement 32768 en donnant à I une valeur négative. Dans ce cas, I est calculé en soustrayant 65536 de l'adresse désirée. Ainsi, pour placer une donnée à l'adresse 45000 par exemple, I sera égal à 45000 - 65536 = -20536.
Attention de ne pas utiliser la mémoire prise soit par le BASIC, soit par le système, sous peine d'avoir à rentrer le BASIC à nouveau.

PEEK est l'instruction complémentaire de POKE :

PEEK (I)

où I est un entier qui spécifie l'adresse de l'octet que l'on veut lire. I est choisi selon les mêmes règles que pour POKE. La valeur lue est un entier entre 0 et 255.
L'utilisation principale de ces deux instructions est de transmettre les arguments et les résultats entre le BASIC et les programmes en langage machine.

- DOKE, DEEK

Ces instructions sont identiques à POKE et PEEK, mais pour des données en deux octets. J peut donc être compris entre + 32767 et -32768, calculé selon les mêmes règles que pour I ci-dessus. L'octet de poids faible est placé à l'adresse I, et l'octet de poids fort à l'adresse I+1. Des échanges de données en 16 bits peuvent ainsi être faits avec des programmes en langage machine.

- OUT, INP

Ces instructions permettent de lire et d'écrire sur les ports :

OUT < I,J >

I et J étant des expressions entières, OUT envoie en sortie sur le port I l'octet de valeur J.
I et J doivent être compris entre 0 et 255.

INP < I >

INP lit un octet du port I, I étant une expression entière comprise entre 0 et 255. Exemple :

```
20 IF INP(J) = 16 THEN PRINT "0 "
```

3. FONCTIONS

Le format des fonctions est

< nom> (<argument>)

où le nom est une fonction déterminée auparavant et l'argument est une expression. Un seul argument est permis. Les fonctions peuvent être incluses dans des expressions.

Exemples :

```
10 LET T = (F*SIN(T))/P
20 C = SQR (A↑2 + B↑2 + 2 * A * COS (T) )
```

3.1 FONCTIONS INTRINSEQUES

Le BASIC de NASCOM définit des fonctions qui peuvent être appelées directement. Leur liste figure au paragraphe 6.3.

3.2 FONCTIONS UTILISATEUR

- DEF

Le programmeur peut définir ses propres fonctions grâce à DEF

DEF < nom de fonction > (< nom de variable >) = < expression >

où le nom de fonction doit commencer par FN suivi par un nom permis de variable et un nom de variable fictive. La variable fictive représente l'argument de la variable ou la valeur du CALL de la fonction. Un seul argument est permis par fonction utilisateur. Une quelconque expression peut apparaître à droite de l'équation, mais elle ne doit pas dépasser la ligne.
Les fonctions utilisateur sur les chaînes ne sont pas permises.
Exemple de fonctions permises :

```
10 DEF FNAVE (V,W) = (V+W)/2
12 DEF FNRAD (DEG) = 3.14159/180*DEG
```

qui calcule l'équivalent d'un angle de degré en radians.

Une fonction peut être définie à nouveau en exécutant un autre DEF avec le même nom. L'instruction DEF doit être exécutée avant que la fonction correspondante puisse être appelée.

- USR

Cette fonction appelle les sous-programmes en langage machine - voir annexe D -

3.3 MESSAGES D'ERREUR SUR FONCTION

Lorsqu'un appel non accepté d'une fonction est effectué, le message FC ou ILLEGAL FUNCTION CALL est affiché.
Les raisons de l'erreur peuvent être :

- indice de tableau négatif. Exemple : LET A(-1) = 0
- indice de tableau plus grand que 32767
- argument de LOG nul ou négatif
- argument de SQR négatif
- A négatif et B non entier dans A↑B
- appel de USR sans adresse de sous-programme
- arguments non valides pour MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, INSTR, STRING$, SPACE$, ON ... GOTO

L'appel d'une fonction utilisateur sans définition préalable par DEF fera apparaître l'erreur UF ou UNDEFINED USER FUNCTION.
Une fonction devant recevoir une chaîne par argument et qui reçoit une valeur numérique, ou inversement, provoquera l'affichage de l'erreur TM, ou TYPE MISMATCH.

4. CHAINES DE CARACTERES

Les expressions peuvent être soit des valeurs numériques, soit des chaînes de caractères.

4.1 CHAINES

Les chaînes sont des listes de caractères alphanumériques qui peuvent atteindre 255 caractères. Elles peuvent être soit des constantes, soit être représentées symboliquement par des variables.
Les chaînes sont limitées par des guillemets. Un nom de variable représentant une chaîne doit se terminer par le signe dollar $.
Exemples :

```
A$ = "ABCD"
B9$ = "14A/56"
FOOFOO$ = "ES"
```

Une chaîne entrée par une instruction INPUT n'a pas à être placée entre guillemets.
Un tableau de chaînes peut être dimensionné grâce à DIM. Chaque élément du tableau est une chaîne qui peut avoir 255 caractères. Le nombre total de caractères chaînés utilisé en tout point durant l'exécution d'un programme ne doit pas dépasser l'espace alloué pour les chaînes. Sinon, un message d'erreur OS ou OUT OF STRING SPACE sera affiché. L'espace réservé aux chaînes est alloué par l'instruction CLEAR (voir paragraphe 6.2).

4.2 OPERATIONS SUR LES CHAINES

4.2.1. Opérateurs de comparaison

Ils sont identiques à ceux utilisés pour les nombres :

=	égal	>	plus grand que
< >	différent	=<, <=	plus petit ou égal à
<	plus petit que	=>, >=	plus grand ou égal à

La comparaison est faite caractère par caractère sur la base des codes ASCII, jusqu'à ce qu'une différence soit trouvée. Si, durant la comparaison, la fin d'une chaîne est détectée avant la fin de l'autre, la première est la plus petite.
Les codes ASCII sont donnés en annexe A. Exemples :

```
A < Z          Valeur ASCII de A est 065, Z est 090
1 < A          Valeur ASCII de 1 est 049
"A  " > "A"    Les espaces en tête et en fin sont significatifs à l'intérieur des guillements.
```

4.2.2. Expressions contenant des chaînes

Les expressions peuvent contenir des chaînes littérales, des chaînes de variables, des chaînes de fonctions reliées par le signe + qui est ici un opérateur de concaténation. L'effet de l'opérateur de concaténation est d'ajouter la chaîne à

droite de l'opérateur à la fin de la chaîne située à sa gauche. Si le résultat de la concaténation est une chaîne de plus de 255 caractères de long, un message d'erreur LS ou STRING TOO LONG est affiché, et l'exécution est arrêtée.

4.2.3. Entrées/Sorties

Les mêmes instructions que pour les données numériques sont utilisables pour les chaînes.

- INPUT, PRINT

Ces instructions lisent et écrivent des chaînes sur le terminal. Les chaînes n'ont pas à être mises entre guillemets. Si elles ne le sont pas, les blancs en début et en fin sont ignorés, et la chaîne sera terminée par la première virgule ou le premier point virgule rencontré. Exemple :

```
10 INPUT ZOO$,FOO$          Lit deux chaînes
20 INPUT X$                 Lit une chaîne, et l'assigne à la variable X
30 PRINT X$,"RE-BONJOUR "   Ecrit deux chaînes, y compris les espaces et la ponctuation de la seconde.
```

- DATA, READ

Instructions identiques à celles des données numériques. Pour les conventions de format, voir les explications données pour INPUT et PRINT ci-dessus.

4.3 FONCTIONS DE CHAINES

Le format d'appel des fonctions intrinsèques est le même que pour les fonctions numériques. Voir le paragraphe 6.3 pour la liste des fonctions. Le nom des fonctions de chaîne doit se terminer par le signe dollar.

5. AUTRES ORDRES

Le BASIC de NASCOM possède des ordres qui ne sont généralement pas trouvés pour un BASIC 8K. En plus des fonctions DEEK et DOKE vues précédemment, il est possible d'appeler le moniteur, manipuler l'écran d'affichage, utiliser directement des terminaux, des imprimantes et les circuits graphiques.

- MONITOR

Cet ordre donne la main au moniteur. Le contrôle peut être rendu au BASIC à partir de Nas-sys en utilisant l'ordre moniteur J pour reprendre un nouveau traitement, soit l'ordre Z pour poursuivre le traitement BASIC déjà entamé en préservant les programmes. Si des points d'arrêt ont été placés dans le programme machine pour permettre leur mise au point, le retour au BASIC se fera par EFFFA ou EFFFD, car les ordres J et Z ne placent pas les points d'arrêt. L'ordre MONITOR est souvent utile pour émettre un ordre XO pour mettre une imprimante en fonctionnement.

- WIDTH N

Les lignes d'entrées et de sortie sont supposées être de 48 caractères, ce qui est un inconvénient si l'on utilise une imprimante. WIDTH (N) donne à la ligne la longueur de N caractères, N étant compris entre 1 et 255. La sortie sur le port série sera automatiquement effectuée après N caractères. Le passage à la ligne sera effectué par ailleurs chaque 48 caractères sur l'écran de NASCOM.
En entrée, le buffer de ligne contiendra soit 72 caractères, soit N si N est plus grand.
A l'entrée de données ou de programmes avec les moniteurs Nasbug T2, T4 ou Nas-sys en mode XO, un passage à la ligne sera effectué après 48 caractères pour les besoins internes. A l'entrée de données ou de programmes avec Nas-sys en mode normal, la largeur de ligne est de 48 caractères et n'est pas effectuée par l'ordre WIDTH.

- CLS

Cet ordre efface l'écran, quel que soit le moniteur.

- SCREEN X,Y

Le curseur est placé sur la ligne Y, position X.
X peut être compris entre 1 et 48, et Y entre 1 et 16. Remarquons que la ligne 16 est celle du haut de l'écran et qu'elle n'est pas décalée. La ligne 1 vient juste dessous et la ligne 15 est en bas de l'écran. Exemple :

```
SCREEN 24,8
PRINT "BONJOUR"    Le message BONJOUR sera placé sur la ligne 8, à partir du caractère 24.
```

Avec le moniteur Nas-sys, seulement 1 caractère à la fois peut être affiché sur la ligne 16. En effet, il se replace ensuite en bas de l'écran. Pour éviter cette situation, on peut utiliser le programme de l'annexe H.

- LINE N

L'ordre LIST affichera 5 lignes de programme, puis les 5 lignes suivantes si l'on appuie sur une touche quelconque. On peut changer le nombre de lignes et le faire passer de 5 au nombre N : par exemple, 14 peut occuper l'écran ou plus pour faire un listage sur une imprimante série.
N doit être situé entre +32767 et -32768, étant entendu que les nombres négatifs représentent 65536+N.

- SET, RESET, POINT

Ces trois ordres sont utilisables avec les options graphiques à définition réduite de NASCOM 1, ou avec NASCOM 2 avec les caractères graphiques. Les points peuvent être définis en noir ou blanc, sur une grille de 96 de large sur 48 de haut. Le point de coordonnée 0,0 est en haut à gauche.

```
SET (X,Y)     positionne le point X,Y à blanc
RESET (X,Y)   si le point X,Y est blanc, le remet à noir
POINT (X,Y)   est une fonction qui donne la valeur 1 si le point X,Y est blanc, et 0 s'il est noir.
```

X et Y doivent être respectivement entre 0-95 et 0-47. Les points dont les valeurs Y sont dans la plage 45-47 apparaissant sur la ligne du haut (ligne non décalée). Pour les autres valeurs, le point apparaît sur la ligne correspondante, en descendant alors que la valeur de Y augmente.
La ligne où se trouve un point est calculée par L = INT(Y/3)+1
La position du caractère où un point apparaît est C = INT(X/2)+1
Les caractères, autres que les caractères graphiques C0H à FFH sont écrasés par SET, et ignorés par RESET et POINT.

- Entrée de donnée avec Nas-sys.

Dans le mode normal, Nas-sys génère un passage à la ligne après affichage d'un point d'interrogation ? et l'utilisateur peut composer une ligne en utilisant les aides à l'édition. Pour certaines applications où un format complexe est requis pour l'écran, cela ne convient pas. Le mode d'opération peut alors être changé comme il suit :

```
DOKE 4175,-6670
```

omet le passage à la ligne et les données comportent toute la ligne.

DOKE 4175,-6649
ne permet l'entrée que d'un caractère à la fois, comme pour les opérations en mode XO. L'éditeur de Nas-sys n'est pas opérationnel. Le caractère de passage à la ligne ne sera pas sorti après le point d'interrogation.
DOKE 4175,-25
restaure les opérations dans les conditions normales.
Attention à l'utilisation de ces ordres, car des arguments incorrects provoqueront des erreurs irréparables dans le programme. Les valeurs ci-dessus pourront être modifiées avec les nouvelles versions de l'interpréteur BASIC.

- Utilisation d'une imprimante

En plus des imprimantes et terminaux reliés à l'interface série, les imprimantes peuvent être branchées au PIO de NASCOM. Les programmes nécessaires peuvent être entrés et appelés grâce aux ordres DOKE et POKE - voir les manuels de moniteur pour les informations détaillées.

- Fin provoquée de programme

"Escape" ($hift et Newline) arrête définitivement le programme en cours (voir paragraphe 6.4), mais seulement à partir du clavier de NASCOM. La génération d'un NMI (interruption non masquable) conduit au même résultat. Il est possible de terminer de cette façon les opérations de lecture ou d'écriture sur cassette, ou les entrées de ligne avec Nas-sys en mode normal. Mais un effet identique est obtenu en appuyant sur Reset et en entrant l'ordre Z.

6. REPERTOIRE DES INSTRUCTIONS

6.1 LES ORDRES

Le BASIC de NASCOM accepte les ordres après l'apparition de "OK" sur l'écran.

CLEAR	Donne la valeur zéro à toutes les variables.
CLEAR [< expression >]	Comme CLEAR, mais donne à la longueur de chaîne traitée la valeur de l'expression. En l'absence d'expression, la dimension de chaîne est mise à 50 octets.
CLOAD < expression chaîne >	Provoque le chargement en mémoire du programme en cassette désigné par le premier caractère de < expression chaîne > . Un ordre NEW est exécuté avant le chargement.
CLOAD ? < expression chaîne >	Vérifie que le programme spécifié peut être chargé et ne contient pas d'erreurs.
CLOAD * < nom de tableau >	Charge le tableau désigné à partir d'une cassette. Peut être utilisé comme une instruction de programme.
CONT	Continue l'exécution du programme après un appui sur la touche "Escape", ou après un STOP ou un END. L'exécution continue à l'instruction suivante, sauf si l'entrée venant du terminal était interrompu. Dans ce cas, l'exécution reprend avec la ré-impression du "prompt". L'utilisation de CONT est intéressante pendant la mise au point des programmes, par exemple pour rectifier des boucles fermées. En tapant sur "Escape", le contrôle est donné au BASIC qui peut alors utiliser les ordres directs. On peut imprimer les valeurs intermédiaires grâce aux ordres PRINT directs, changer la valeur des variables ... L'exécution peut être relancée par un CONT ou un GOTO direct à une ligne de programme. L'exécution ne peut pas être reprise si une erreur en mode direct a été faite durant la suspension du traitement ou si le programme a été modifié.
CSAVE < expression chaîne >	Provoque le chargement du programme sur cassette sous la dénomination spécifiée par le premier caractère de l'expression.
CSAVE * < nom de tableau >	Provoque le chargement sur cassette du tableau. Peut être employé dans un programme.
LIST	Fait le listage du programme en mémoire à partir du numéro de ligne le plus bas. Le listage se termine à la fin du programme ou en tapant Escape (Shift + Newline).
LIST [< numéro de ligne >]	Affiche le programme à partir de la ligne donnée. LIST affiche n lignes, n étant 5 si une valeur n'est pas spécifiée par l'ordre LINES, et attend l'entrée d'un caractère pour continuer. Le contrôle au BASIC est effectué en tapant Escape.
NEW	Efface le programme en mémoire et met toutes les variables à zéro. Est utilisé avant d'entrer un nouveau programme.
NULL < expression entière >	Défini le nombre de nuls à imprimer à la fin de chaque ligne, égal à <expression entière> -1 .NULL 0 est équivallent à NULL 256 .La valeur par défaut est 1, c'est-à-dire qu'aucun nul n'est affiché. Les nuls sont générés par l'interface série, et non par le moniteur. L'ordre NULL est utilisé pour: - produire un délai supplémentaire dans la commande d'une imprimante série pour permettre le retour du chariot, - produire un délai supplémentaire pendant un LISTING ou PRINTING sur cassette, de telle façon qu'à la relecture l'interpréteur aura le temps d'effectuer les opérations accessoires de "nettoyage" avant de lire la ligne suivante.
RUN [< numéro de ligne >]	Fait commencer l'exécution du programme en mémoire à la ligne donnée. Si le numéro est omis, l'exécution commence au numéro le plus bas.

6.2 LES INSTRUCTIONS

Dans ce qui suit, X et Y sont des expressions quelconques autorisées, I et J des expressions tronquées en entier, ou des entiers, V et W des noms de variables. Le format des lignes de BASIC est le suivant :

< nnnn > < instruction > [: < instruction > ...]

où nnnn est le numéro de ligne.

DATA < liste >	Spécifie des données à lire par l'instruction READ. Les éléments de la liste peuvent être des nombres ou des chaînes, et sont séparés par des virgules.
DEF FNV (<W>)= <X>	Définit une fonction utilisateur. Le nom de la fonction est FN suivi d'un nom de variabl autorisé. L'instruction doit tenir sur une ligne de 72 caractères.
DOKE <I> , <J>	Place J en mémoire à l'adresse I et IM. Si I ou J sont négatifs, ils sont transformés par la formule 65536 + I (ou J)
DIM <V> (<I> [, J ...]) [, ...]	Alloue la mémoire aux tableaux. Un seul DIM peut dimensionner plusieurs tableaux tenant sur une ligne. Le plus petit indice est 0. Sans instruction DIM, un tableau est supposé avec des dimensions égales à 10. Ainsi, A(I,J) aura 121 éléments par défaut, de A(0,0) à A(10,10), en l'absence de DIM le spécifiant.
END	Termine l'exécution d'un programme.

FOR <V> = <X> TO <Y> [STEP <Z>]	Permet de répétiter l'exécution d'instructions. La première exécution se fait avec V = X et se poursuit jusqu'à NEXT. Z est ajouté à V. Si Z < 0 et V >= Y, ou si Z > 0 et V <= Y, le Basic se branche sur l'instruction après FOR. Sinon, l'exécution continue à l'instruction qui suit NEXT.
GOTO <nnnn>	Branchement inconditionnel au numéro de ligne nnnn.
GOSUB <nnnn>	Branchement inconditionnel au sous-programme commençant à la ligne nnnn.
IF <X> GOTO <nnnn>	Comme IF ... THEN, mais GOTO doit être obligatoirement suivi d'un numéro de ligne.
IF <X> THEN <instruction> [: instruction ...] IF <X> THEN <Y>	Si la valeur de X est différente de 0, branchement sur le numéro de ligne donné ou sur l'instruction après THEN. Sinon, l'exécution se poursuit à la ligne qui suit le IF ... THEN.
INPUT <V> [, <W> ...]	Provoque une demande d'entrée du terminal. Les valeurs entrées sont assignées aux variables de la liste.
LET <V> = <X>	Assigne à la variable la valeur de l'expression. Le mot LET n'est pas obligatoire.
LINES <A>	Donne le nombre de lignes affichées par l'ordre LIST.
NEXT [<V> , <W> ...]	Dernière instruction d'une boucle FOR. V est la variable de la boucle la plus récente, W de celle qui vient juste avant, etc. Sans la présence de variables, NEXT termine la boucle la plus récente.
ON <I> GOTO <liste de numéros de ligne>	Fait le branchement sur le numéro de ligne de rang I de la liste. Les éléments de la liste sont séparés par des virgules. Si I égal à 0 ou est plus grand que le nombre d'éléments de la liste, l'exécution continue à l'instruction suivante. Une erreur est affichée si I < 0 ou I > 255.
ON <I> GOSUB <liste>	Comme ON ... GOTO, mais les éléments de la liste sont les numéros de lignes initiaux des sous-programmes.
OUT <I> , <J>	Envoie sur le port I l'octet J. Il faut que 0 <= I et J <= 255
POKE <I> , <J>	Place en I (ou à l'adresse dérivée de I), l'octet J. Avec 0 <= J <= 255 et -32768 < I < 65536. Si I est négatif, l'adresse dérivée est 65536 + I
PRINT <X> [, <Y> ...]	Provoque l'affichage sur le terminal de la valeur des expressions de la liste. Les espacements sont déterminés par la ponctuation. , impression au début de la zone de 14 colonnes suivantes ; impression immédiate. rien : impression à la ligne suivante. Les chaînes littérales peuvent être affichées si elles sont placées entre guillemets "".
READ <V> [, <W> ...]	Assigne aux variables les valeurs données par DATA. L'affectation se fait en séquence à partir de la première valeur de l'instruction DATA.
REM [<commentaire>]	Permet l'insertion de commentaires. N'est pas exécuté, mais permet le branchement. Dans les versions étendues, les commentaires peuvent être ajoutés à la fin d'une ligne s'il est précédé du signe ' (guillemet simple).
RESTORE	Permet de relire les données de l'instruction DATA. L'instruction READ après RESTORE commence avec la première donnée de la première instruction DATA.
RETURN	Termine un sous-programme et branche sur l'instruction qui suit le GOSUB le plus récent.
SCREEN <X> , <Y>	Place le curseur sur la ligne Y, position du caractère X.
STOP	Arrête l'exécution du programme. Le contrôle est rendu au BASIC qui affiche BREAK IN LINE nnnn.
WAIT <I> , <J>[, <K>]	Les opérations logiques suivantes sont effectuées avec l'octet du port I : XOR avec K et AND avec J. L'exécution se poursuit jusqu'à ce que le résultat soit un non zéro. La valeur par défaut de K est 0. On doit avoir 0 <= I, J, K <= 255.
WIDTH <n>	Donne aux buffers d'entrée et d'impression la valeur n - voir paragraphe 5.

6.3 FONCTIONS INTRINSEQUES

Les fonctions suivantes peuvent être appelées dans un programme sans qu'il soit nécessaire de les définir préalablement X et Y sont des expressions, I et J des expressions entières et X$ et Y$ des expressions chaînes.

ABS (X)	Donne la valeur absolue de l'expression X. Ainsi, ABS(X) = X si X >= 0 et -X si X <0
ASC (X$)	Donne le code ASCII du premier caractère de la chaîne X$.
ATN (X)	Donne l'arc-tangente de X, en radian, dans la fourchette de -π/2 à π/2
CHR$ (I)	Donne une chaîne dont un élément est le code ASCII "I". Voir les codes ASCII en annexe A.
COS (X)	Donne le cosinus de X, X étant en radians.
EXP (X)	Donne e à la puissance X. On doit avoir X <= 87.3365
FRE (0)	Donne le nombre d'octets de mémoire non utilisé par BASIC. Si l'argument est une chaîne donne le nombre d'octets libres dans l'emplacement chaîne.
INP (I)	Donne l'octet se trouvant au port I
INT (X)	Donne l'entier le plus grand contenu dans X
LEFT$ (X$,I)	Donne le I ème caractère à partir de la gauche de la chaîne X$.

LEN (X$)	Donne la longueur de la chaîne X$. Tous les caractères sont comptés, y compris les blancs.
LOG (X)	Donne le logarithme naturel de X. On doit avoir X > 0.
MID$ (X$,I [,J])	Sans la présence de J, donne les caractères de la chaîne X$ situés à droite du I ème caractère. Si I > LEN (X$), MID$ donne la chaîne avec des nuls. On doit avoir 0 < I < 255. Avec 3 arguments, donne une chaîne de caractères de longueur J à partir du I ème caractère. Si J est plus grand que le nombre de caractères de X$ à la droite du I ème caractère, MID$ donne le reste de la chaîne. On doit avoir 0 <= J <= 255.
POS (I)	Donne la position de la tête d'imprimante. La position la plus à gauche est 0.
RND (X)	Donne un nombre au hasard entre 0 et 1. X < 0 fait commencer une nouvelle séquence de nombres au hasard X > 0 donne le nombre au hasard suivant de la séquence X = 0 donne le dernier nombre. Les séquences commençant avec le même nombre négatif seront identiques entre elles.
RIGHT$ (X$,I)	Donne les I caractères de droite de la liste X$. Si I = LEN (X$), donne X$.
SGN (X)	Si X > 0, donne 1, Si X = 0, donne 0, si X < 0, donne -1. Exemple d'utilisation : ON SGN (X) + 2 GOTO 100,200,300 fait le branchement sur 100 si X < 0, sur 200 si X = 0, et sur 300 si X > 0.
SIN (X)	Donne le sinus de la valeur de X en radians. On a COS (X) = SIN (X + 3.14159/2)
SPC (I)	Affiche I blancs sur le terminal. On doit avoir 0 < = I < = 255
SQR (X)	Donne la racine carrée de X, avec X > 0
STR$ (X)	Donne la chaîne que représente X.
TAB (I)	Vient placer des espaces jusqu'à la position I sur le terminal, la position étant repérée par 0 à gauche jusqu'à 71 à droite. Si le chariot est déjà au-delà de I, TAB n'a pas d'effet. On doit avoir 0 < = I < = 255. Ne doit être utilisé que dans les instructions PRINT.
TAN (X)	Donne la tangente de X, X étant en radians.
USR (X)	Appelle le sous-programme utilisateur en langage machine, dont l'argument est X.
VAL (X$)	Donne la valeur numérique de la chaîne X$. Si le premier caractère de X$ n'est pas +,-, & ou un chiffre, VAL (X$) = 0

6.4 CARACTERES SPECIAUX

Le BASIC de NASCOM affecte des fonctions spéciales à plusieurs caractères ASCII. Les caractères tels que Control/C, Control/S, sont obtenus en appuyant sur la touche Control et tapant la lettre désignée. La liste suivante donne ces caractères spéciaux. Il est à remarquer que les caractères précédés d'un astérisque ne sont pas utilisés avec le moniteur Nas-sys en mode normal.

*	@	(Escape ou Shift + Newline pour Nas-sys) Efface la ligne en cours et exécute un retour chariot.
	Backspace	Efface le dernier caractère entré. S'il n'y a plus de caractère, taper un ratour chariot C
*	-	(Signe souligné) - Comme Backspace .
	Newline (ou Enter)	Effectue le retour de la tête d'impression ou du chariot à la ligne suivante.
	Escape	(Shift + Newline) - Interrompt l'exécution du programme ou de la liste. Prend effet à la fin de l'exécution de l'instruction en cours. BASIC reprend le contrôle et affiche OK. L'ordre CONT permet de reprendre l'exécution. Voir paragraphe 6.1. Attention : ceci n'est valable qu'à partir du clavier de NASCOM. La touche ESC sur un terminal série connecté n'aura aucun effet.
	:	Sépare les instructions sur une ligne.
	?	Equivaut à l'instruction PRINT .
	Rubout	(Control Z) Supprime le caractère précédent d'une ligne en entrée. Le premier Rubout imprime le dernier caractère à imprimer. Chaque Rubout successif imprime le caractère suivant à gauche, et l'efface en imprimant un caractère nouveau.
*	Control/R	Imprime Newline et écho la ligne en entrée.
	Control/U	Comme @
	Caractères miniscules	Ils sont toujours traités en tant que minuscules, mais LIST et PRINT les transformeront en majuscules s'ils ne font pas partie d'une chaîne littérale, d'un commentaire dans sa forme REM ou sa forme entre guillemets.

6.5 MESSAGES D'ERREUR

Après une erreur, BASIC reprend le contrôle et affiche OK. Les valeurs des variables et le texte du programme ne sont pas altérés, mais le programme ne peut pas être continué par CONT. Tout le contexte nécessaire aux GOSUB et FOR est perdu. Cependant, le programme peut être poursuivi par un GOTO en mode direct.

Ordre direct ?XX ERROR
Instruction indirecte ?XX ERROR IN YYYY
où XX est le code erreur et YYYY le numéro de ligne qui contient l'erreur. La liste des erreurs est donnée ci-dessous par son code, son numéro de message et le message complet d'erreur.

BS
SUBSCRIPT OUT OF RANGE — On a essayé de traiter un élément de tableau avec un indice dépassant la dimension définie. C'est le cas par exemple pour LET A(1;1,1) = Z alors que A a été défini par DIM A (10,10).

DD
REDIMENSIONED ARRAY — Après avoir dimensionné un tableau, une seconde instruction de dimension est rencontrée. Ceci arrive en particulier lorsque DIM est donné alors qu'une dimension de 10 par défaut a déjà été attribuée pour ce tableau.

FC
ILLEGAL FUNCTION CALL — Le paramètre donné à une fonction mathématique ou à une fonction de chaîne dépasse les limites permises. Ceci se produit dans les cas suivants :
- indice de tableau négatif
- indice trop grand, supérieur à 32767
- LOG avec un argument nul ou négatif
- SQR avec un argument négatif
- A↑B avec A négatif et B non entier
- appel à USR avant l'entrée de l'adresse du programme en langage machine
- un argument non permis dans MID$, LEFT$, RIGHT$, INP, OUT, WAIT, SCREEN, WIDTH, SET, RESET, POINT, DEEK, DOKE, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR ou ON ... GOTO.

ID
ILLEGAL DIRECT — INPUT et DEF ne sont pas permis en mode direct.

NF
NEXT WITHOUT FOR — La variable d'une instruction NEXT ne correspond pas à un FOR exécuté précédemment.

OD
OUT OF DATA — Une instruction READ a été exécutée, mais toutes les données de DATA ont déjà été lues. Trop de données lues, ou données en nombre insuffisant.

OM
OUT OF MEMORY — Le programme est trop important ou a trop de variables, trop de boucles FOR, trop de GOSUB ou des expressions trop complexes. - Voir annexe C -

OV
OVERFLOW — Le résultat du calcul dépasse le format admissible. Si le résultat est trop petit, la valeur zéro est donnée et le calcul se poursuit sans message d'erreur.

SN
SYNTAX ERROR — Parenthèses manquantes, caractère non permis, ponctuation incorrecte.

RG
RETURN WITHOUT GOSUB — RETURN a été rencontré sans qu'il y ait eu un GOSUB correspondant.

UL
UNDEFINED LINE — Le numéro de ligne de branchement indiqué par un GOTO, GOSUB, IF...THEN, n'existe pas.

/0
DIVISION BY ZERO — Division par zéro, ou zéro à une puissance négative.

CN
CAN'T CONTINUE — Essai de traitement d'un programme qui n'existe pas, qu'une erreur ait été rencontrée ou que le programme ait été modifié.

LS
STRING TOO LONG — La chaîne contient plus de 255 caractères.

OS
OUT OF STRING SPACE — Le nombre de caractères chaîne dépasse le volume alloué. Utiliser l'ordre CLEAR pour allouer plus d'espaces chaîne, ou utiliser des chaînes plus petites, ou moins de variables chaîne.

ST
STRING FORMULA TOO COMPLEX — Expression chaîne trop longue ou trop complexe. La fractionner en deux ou plus.

TM
TYPE MISMATCH — La variable à gauche d'une instruction ou d'une opération est numérique alors que la partie droite est une chaîne ou inversement. Une fonction devant avoir un argument sous forme de chaîne reçoit un argument numérique, ou inversement.

UF
UNDEFINED USER FUNCTION — Le programme fait appel à une fonction utilisateur qui n'a pas été définie.

6.6 MOTS RESERVES

Certains mots sont utilisés par l'interprétateur BASIC et ne peuvent pas être pris comme nom de variable. Il s'agit d'une part de la liste des mots ci-dessous, et d'autre part, des noms des fonctions intrinsèques.

CLEAR	NEW	AND	OUT
DATA	NEXT	CONT	POINT
DIM	PRINT	DEF	RESET
END	READ	DOKE	POKE
FOR	REM	FN	SCREEN
GOSUB	RETURN	LINES	SET
GOTO	RUN	NOT	SPC
IF	STOP	NULL	WAIT
INPUT	TO	ON	WIDTH
LET	TAB	OR	
LIST	THEN		
	USR		

7. MISE EN FONCTIONNEMENT DU BASIC

Le Basic se présente sous forme de cassette compatible avec NASBUG T4 et NAS-SYS. Il est précédé sur la cassette par un programme de chargement permettant l'utilisation avec NASBUG T2.

Pour effectuer le chargement avec Nasbug T2,entrer en premier lieu le programme de chargement grâce à l'ordre L .Effectuer son exécution à partir de E00, en tapant EE00. Le Basic est ensuite chargé par blocs de 256 octets.
Pour chaque bloc, une ligne s'affiche sur l'écran:

```
      SSSS RRRR
ou    SSSS      est l'adresse du bloc
      RRRR      le volume d'information à charger après le bloc.
```

Si une erreur est détectée, un message "ERROR" s'affiche. On doit alors rembobiner la cassette avant le bloc mal lu, et continuer la lecture. Il n'est pas necessaire de rembobiner complètement.

Pour effectuer le chargement avec Nasbug T4 ou Nas-sys, taper R et procéder de la manière habituelle.

Apres le chargement, exécuter le Basic à partir de 1000H. Le Basic répondra alors:

```
      Memory Size ?
```

Vous devez alors taper:
-soit NL,auquel cas toute la zone au dessus de 3000H sera utilisée,
-soit une adresse décimale qui sera l'adresse maximum utilisable par le programme.On peut ainsi réserver de la place pour des programmes en code machine sans risquer de les détruire.

Le Basic inscrit alors:

```
      NASCOM TAPE BASIC Ver 1
      Copyright (c) 1978 by Microsoft
        n    Bytes free
      OK
```

ou n est le nombre d'octets disponibles pour le programme et les données.
Le programme ou les ordres directs peuvent alors être entrés.

Après un ordre MONITOR ou un RESET, le Basic peut être appelé à nouveau en l'exécutant à partir de 100CH, et sans avoir à ré-initialiser le programme et l'espace mémoire (démarrage à chaud).

Le Basic cassette peut être entré sur des EPROM. Il est alors appelé de la même façon, puisque la zone de travail se trouve au delà de 3000H.

ANNEXE A

CODE ASCII

DECIMAL	CHAR.	DECIMAL	CHAR.	DECIMAL	CHAR.
000	NUL	043	+	086	V
001	SOH	044	,	087	W
002	STX	045	-	088	X
003	ETX	046	.	089	Y
004	EOT	047	/	090	Z
005	ENQ	048	0	091	[
006	ACK	049	1	092	\
007	BEL	050	2	093	]
008	BS	051	3	094	^
009	HT	052	4	095	_
010	LF	053	5	096	`
011	VT	054	6	097	a
012	FF	055	7	098	b
013	CR	056	8	099	c
014	SO	057	9	100	d
015	SI	058	:	101	e
016	DLE	059	;	102	f
017	DC1	060	<	103	g
018	DC2	061	=	104	h
019	DC3	062	>	105	i
020	DC4	063	?	106	j
021	NAK	064	@	107	k
022	SYN	065	A	108	l
023	ETB	066	B	109	m
024	CAN	067	C	110	n
025	EM	068	D	111	o
026	SUB	069	E	112	p
027	ESCAPE	070	F	113	q
028	FS	071	G	114	r
029	GS	072	H	115	s
030	RS	073	I	116	t
031	US	074	J	117	u
032	SPACE	075	K	118	v
033	!	076	L	119	w
034	"	077	M	120	x
035	#	078	N	121	y
036	$	079	O	122	z
037	%	080	P	123	{
037	&	081	Q	124	|
039	'	082	R	125	}
040	(	083	S	126	~
041	)	084	T	127	DEL
042	*	085	U		

LF = Line Feed FF = Form Feed Cr = Carriage Return Del = Rubout

UTILISATION DES CODES ASCII : FONCTION CHR$

La fonction CHR$ donne une chaîne dont le caractère a la valeur ASCII égale à X. La fonction ASC(X$) convertit le premier caractère d'une chaîne en sa valeur ASCII décimale.

L'une des utilisations fréquentes de CHR$ est l'envoi d'un caractère spécial sur le terminal utilisateur. Le caractère le plus utilisé est le BEL (ASCII 7), qui provoquera une sonnerie sur certains terminaux, et un "beep" sur d'autres. C'est ainsi que l'on peut souligner un message d'erreur, ou ... réveiller l'utilisateur un peu lent en faisant PRINT CHR$ (7) ;

Un autre emploi des autres caractères spéciaux est permis par les CRT qui possèdent le positionnement du curseur et d'autres fonctions spéciales (mise en marche d'une imprimante par exemple). Sur certain terminal de visualisation par exemple, le caractère "form feed" (CHR$(12)) provoquera l'effacement de l'écran et le positionnement du curseur en haut à gauche.

Certains CRT permettent de faire des courbes et des graphiques en utilisant la fonction CHR$ du Basic de NASCOM.

A N N E X E B

INFORMATIONS SUR LES REPRESENTATIONS EN MEMOIRE ET LA VITESSE D'EXECUTION

A. REPRESENTATION MEMOIRE

La taille prise par un programme dépend du nombre d'éléments qui forment le programme.

ELEMENT	ESPACE REQUIS
Variable numérique	6 octets
Tableau	(Nb d'éléments de chaîne ou numérique) * 6 * 5 * (nb de dimensions) * 2 octets
Fonction intrinsèque	1 octet pour le CALL
Fonction utilisateur	6 octets pour la définition
Mot réservé	1 octet chacun
Autre caractère	1 octet
Boucle FOR active	16 octets
GOSUB actif	5 octets
Parenthèses	6 octets la paire
Résultat intermédiaire	10 octets

B. REDUCTION DU VOLUME MEMOIRE

Quelques règles permettent de réduire le volume mémoire.

- Utiliser plusieurs instructions par ligne de programme puisque les numéros de ligne prennent 5 octets.
- Supprimer les espaces inutiles.
  Ainsi, au lieu de:
  10 PRINT X,Y,Z utiliser : 10 PRINTX,Y,Z
- Supprimer les commentaires inutiles, ce qui économise 1 octet pour REM et 1 octet par caractère ensuite.
- Utiliser des variables au lieu de constantes lorsque la valeur est utilisée plusieurs fois.
  Ainsi la constante 3.14159 utilisée 10 fois dans un programme demandera 40 octets de plus que si on assigne à la variable P la valeur correspondante :
  10 P = 3.14159
  P étant utilisé 10 fois.
- Ne pas mettre END à la fin d'un programme économise 1 octet.
- Ré-utiliser à nouveau les noms de variables ayant déjà servi.
- Utiliser des sous-programmes lorsque des portions de programme doivent être traitées plusieurs fois.
- Les éléments de tableau ayant l'indice 0 doivent être utilisés.
  Ainsi, le tableau dimensionné par:
  100 DIM A(10)
  possède 11 éléments, de A(0) à A(10).

C. POUR AUGMENTER LA VITESSE D'EXECUTION

- La suppression des espaces et des REMarques réduit légèrement le temps d'exécution.
- Les variables sont placées dans un tableau dans l'ordre d'apparition dans le programme. Chaque variable est ensuite recherchée dans le tableau. Il faut donc ré-utiliser les mêmes noms de variables afin que le tableau soit le plus court possible.
- Utiliser NEXT sans variable.
- Utiliser des variables au lieu de constantes, particulièrement dans les boucles FOR.
- Les variables chaîne sont définies par le BASIC par la longueur de la chaîne, et un pointeur de la première position de la chaîne. Lorsque les chaînes sont manipulées, l'espace chaîne se remplit de résultats intermédiaires. Ceux-ci sont éliminés grâce à un programme de "nettoyage". Ce programme sera d'autant moins utilisé qu'il y a moins de variables chaînes et que les chaînes sont plus longues.

ANNEXE C

FONCTIONS TRIGONOMETRIQUES COMPLEMENTAIRES

Les fonctions suivantes peuvent être obtenues grâce aux fonctions intrinsèques fournies par le BASIC de NASCOM.

FONCTION	FORMULATION EN BASIC
SECANTE	SEC(X) = 1/COS(X)
COSECANTE	CSC(X) = 1/SIN(X)
COTANGENTE	COT(X) = 1/TAN(X)
SINUS INVERSE	ARCSIN(X) = ATN(X/SQR(-X*X+1))
COSINUS INVERSE	ARCCOS(X) = -ATN X(X/SQR(-X*X+1)) +1.5708
SECANTE INVERSE	ARCSEC(X) = ATN(XSQR(X*X-1)) +SGN(SGN(X)-1)*1.5708
COSECANTE INVERSE	ARCCSC(X) = ATN(I/SQR(X*X-1)) +(SGN(X)-1)*1.5708
COTANGENTE INVERSE	ARCCOT(X) = ATN(X)+1.5708
SINUS HYPERBOLIQUE	SINH(X) = (EXP(X)-EXP(-X))/2
COSINUS HYPERBOLIQUE	COSH(X) = (EXP(X)+EXP(-X))/2
TANGENTE HYPERBOLIQUE	TANH(X) = EXP(-X)/EXP(X)+EXP(-X)) *2+1
SECANTE HYPERBOLIQUE	SECH(X) = 2/(EXP(X)+EXP(=X))
COSECANTE	CSCH(X) = 2/(EXP(X)-EXP(-X)
COTANGENTE	COTH(X) = EXP(-X)/(EXP(X)-EXP(-X)) *2+1
SINUS HYPERBOLIQUE INVERSE	ARCSINH(X) = LOG(X+SQR(X*X+1))
COSINUS HYPERBOLIQUE INVERSE	ARCCOSH(X) = LOG(X+SQR5X*X+-1))
TANGENTE HYPERBOLIQUE INVERSE	ARCTANH(X) = LOG((1+X)/(1-X))/2
SECANTE HYPERBOLIQUE INVERSE	ARCSECH(X) = LOG((SQR(-X*X+1)+1)/X)
COSECANTE HYPERBOLIQUE INVERSE	ARCCSCH(X) = LOG((SGN(X)*SQR(X*X+1)+1)/X
COTANGENTE HYPERBOLIQUE INVERSE	ARCCOTH(X) = LOG((X+1)/(X-1))/2

ANNEXE D

BASIC ET LANGAGE MACHINE

Le BASIC de NASCOM prévoit une interface avec les sous-programmes en langage machine. La fonction USR permet d'appeler les programmes en assembleur .

La première opération à effectuer est de réserver du volume mémoire. Ainsi, lorsque BASIC demande "MEMORY SIZE" au moment de l'initialisation, l'adresse à introduire est l'adresse maximum allouée, le complément disponible étant réservé aux programmes en assembleur. En effet, le BASIC utilisera les emplacements dont il a besoin à partir de l'adresse 4096. Les emplacements de 0C00H à 1000H qui ne sont pas utilisés par le moniteur peuvent également être utilisés pour les sous-programmes de l'utilisateur.

Si la réponse à la question MEMORY SIZE ? est trop petite, BASIC posera la question à nouveau jusqu'à ce que la mémoire soit suffisante. Voir l'annexe C.

Les programmes en assembleur peuvent être entrés en mémoire par la manière habituelle ou à partir d'un programme BASIC grâce à l'instruction POKE ou DOKE.

L'adresse de départ du programme en code assembleur est en USRLOC, une zone de deux octets placée en 3004H et 3005H. La fonction USR appelle le programme dont l'adresse est en USRLOC. Au départ, USRLOC contient l'adresse de ILLFUN, qui donne le message d'erreur ILLEGAL FUNCTION CALL. Ce message sera donc affiché si une adresse de programme n'a pas été préalablement chargée.

Lorsque USR est appelé, le pointeur de pile est positionné pour 8 niveaux (16 octets) de stockage d'adresse. Si cela est insuffisant, la pile du BASIC peut être sauvegardée et une autre pile constituée pour être utilisée par le programme en assembleur. Au retour du sous-programme, il est nécessaire de restituer la pile du BASIC.

USR est appelé avec un argument unique. Le programme en assembleur peut connaître cet argument en appelant le programme dont l'adresse est en 1008H et 1009H, l'octet bas étant en 1008H. Ce programme (DEINT) conserve l'argument dans la paire de registres [D,E].

L'argument est transformé en entier par troncature et une erreur FC est affichée s'il n'est pas compris entre -32768 et 32767.

Pour faire passer un résultat d'un programme en langage machine vers le BASIC, placer la valeur dans la paire de registres [A,B] . Cette valeur doit être un entier signé en 16 bits compris entre -32768 et +32767. Appeler ensuite le programme dont l'adresse est en 100AH et 100BH. S'il n'est pas appelé, USR(X) donnera X. Pour revenir au BASIC, le sous-programme en langage machine exécutera une instruction RET.

Tous les programmes de manipulation d'interruption doivent sauvegarder la pile, les registres A-L et le PSW. Ils doivent aussi ré-activer la permission d'interruption avant le branchement sur le BASIC puisqu'une interruption rend impossible toute interruption ultérieure.

Plusieurs sous-programmes en assembleur peuvent être utilisés, l'argument de USR désignant le programme appelé. Des arguments supplémentaires peuvent être passés en utilisant POKE ou DOKE, et d'autres valeurs peuvent être renvoyées grâce à PEEK ou DEEK.

ANNEXE E

CHARGEMENT ET LECTURE DE CASSETTE

Les programmes peuvent être chargés sur cassette grâce à l'instruction CSAVE. CSAVE est utilisable en mode direct ou indirect, dans le format suivant :

CSAVE < expression chaîne >

Le programme en mémoire est chargé en cassette sous le nom spécifié par le premier caractère de l'expression chaîne. Le programme appelé A est donc enregistré par CSAVE"A". Après la fin de l'instruction CSAVE, le programme rend toujours le control au moniteur. Les programmes sont écrits sur cassette dans le code interne utilisé par BASIC. Les valeurs des variables ne sont pas chargées, bien que l'instruction indirecte CSAVE ne modifie pas les valeurs des variables en mémoire. Les nombres de nuls (voir ordre NULL) n'a pas d'effet sur le CSAVE.

Avant de lancer CSAVE, mettre en marche la cassette. Vérifier que la cassette est en bonne position et mettre l'appareil en position d'enregistrement.

CSAVE écrit tout d'abord un bloc en-tête qui contient le premier caractère de l'expression chaîne et appelle ensuite le programme du moniteur chargé de l'écriture sur cassette.

Les programmes peuvent être chargés en mémoire grâce à l'ordre CLOAD, de même format que CSAVE. CLOAD a pour effet d'exécuter un ordre NEW et de charger en mémoire le fichier spécifié. Le contrôle est ensuite rendu au BASIC. La lecture commence par la recherche de 3 zéros consécutifs et du nom du fichier. Si celui-ci n'est pas trouvé, ou est mal lu, l'ordinateur poursuivra sa recherche jusqu'à ce qu'il soit arrêté et relancé.

Lorsque le programme est trouvé, le message suivant est affiché :

FICHIER x FOUND

Des données peuvent être écrites et lues grâce aux ordres CSAVE* et CLOAD* dont les formats sont :

CSAVE* < nom de tableau de variable >
CLOAD* < nom de tableau de variable >

- Voir le paragraphe 2.4.4.

Un contrôle par sommation est prévu à l'écriture et à la lecture. Si le contrôle n'est pas positif à la lecture, le message BAD est affiché et le contrôle est repris par BASIC. Les données incorrectes auront été chargées.

En cas de mauvaise lecture, un GOTO direct ou un ordre CLOAD peut être utilisé pour relire les données.

Avec le moniteur Nas-sys, CLOAD ? < expression chaîne > lit le programme sans le mettre en mémoire, ce qui permet d'effectuer un contrôle de ce qui est chargé en mémoire.

On peut aussi utiliser l'ordre XO avec Nasbug T4 et Nas-sys et effectuer des LIST et des PRINT sur cassette. Ceci permet de conserver des bibliothèques de sous-programmes qui peuvent ensuite être incorporés dans différents programmes.

A N N E X E F

CONVERSION DE PROGRAMMES BASIC EN BASIC NASCOM

Certaines incompatibilités existant entre le BASIC NASCOM et des BASIC utilisés par d'autres systèmes.

1. CHAINES

Certains BASIC demandent que la longueur des chaines soit préalablement déclarée. Les instructions correspondantes devront être supprimées du programme. Dans certains de ces BASIC, une déclaration du type A$ (I,J) déclare un tableau de chaines de J éléments, chacun ayant une longueur I. Il faudra transformer ces instructions en instructions DIM A$ (J).

Le BASIC de NASCOM utilise "+" pour la concaténation de chaines et non pas "," ou "&". Il utilise également LEFT$, RIGHT$ et MID$ pour prélever des portions de chaine. D'autres BASIC utilisent A$ (I) pour accéder au I ème caractère de la chaine A$ et A$ (I,J) pour isoler une portion du I ème ou J ème caractère. Convertir de la façon suivante :

A$ (I) à transformer en MID$ (A$,I,1)
A$ (I,J) " MID$ (A$,I,J-I+1)

Ceci suppose que la référence à un indice de A$ est soit dans une expression, soit à droite d'une assignation. Si la référence de A$ est à gauche d'une expression d'assignation et X$ est l'expression chaine utilisée pour remplacer des caractères dans A$, la conversion se fera ainsi :

A$ (I) = X$ à transformer en A$ = LEFT$ (A$,I-1)+X$+MID$(A$,I+1)

A$ (I,J) = X$ " " A$ = LEFT$ (A$,I-1)+X$+MID$(A$,J+1)

2. ASSIGNATIONS MULTIPLES

Certains BASIC permettant d'écrire des expressions telles que :

500 LET B=C=0

ce qui assigne à B et à C la valeur 0. Dans le BASIC de NASCOM, l'effet est tout différent. Les signes "=" situés à droite du premier en partant de la gauche seront interprétés comme des opérateurs logiques. Dans l'exemple précédent, B serait mis à -1 si C égalait zéro. Si C n'était pas égal à zéro, B serait mis à 0.

Les instructions précédentes seront à transformer ainsi :

500 C=0:B=C

3. DIVERS

Certains BASIC utilisent le signe "\" au lieu de ":" comme signe de séparation entre instructions d'une même ligne. Des programmes qui utilisent la fonction MAT devront être ré-écrits afin d'utiliser FOR...NEXT pour effectuer des opérations identiques.

A N N E X E G

ZONES MEMOIRE UTILISEES

Le BASIC réside de 1000H (4096 décimal) à 2FFF (12287) et utilise la zone de 3000H (12288) à 3E11H (12506) comme zone de travail.

La zone mémoire de 0C80H (3200) à 1000H (4096) est libre pour les programmes utilisateur en langage machine.

La taille mémoire minimum nécessaire à l'entrée du BASIC est de 12670 octets.

ANNEXE H

SOUS-PROGRAMMES UTILES

1. POUR ECRIRE SUR LA LIGNE 16 AVEC LE MONITEUR NAS-SYS

La ligne 16 est la ligne qui n'est pas décalée automatiquement.

```
1    REM              CE PROGRAMME PERMET D'ECRIRE SUR
2    REM              LA LIGNE 16 AVEC NAS-SYS
10   CLS
20   SCREEN 1,15
25   REM              MISE SUR LA DERNIERE LIGNE
30   PRINT            "EN-TETE";
35   REM              COPIE SUR LA PREMIERE LIGNE
40   FOR C=2954 to 3000 STEP 2
50   DOKE C+64,DEEK(C)
60   NEXT C
65   REM CHR$(27)     GENERE ESC=EFFACEMENT LIGNE
70   PRINT CHR$(27);
80   REM              LA SUITE PEUT CONTINUER ICI
```

2. TRANSFORMATION D'UN NOMBRE HEXADECIMAL EN DECIMAL

```
4    CLS
5    PRINT
10   INPUT            "ENTRER NB HEX";H$
20   T=0:D=1
30   FOR P=LEN(H$)-1 TO 0 STEP-1
40   C=ASC(MID$(H$,D,1))
50   D=D+1
60   IF (C >=48)*(C <=57) THEN C=C-48:GOTO 100
70   IF (C >=65)*(C <=70) THEN C=C-55: GOTO 100
80   PRINT            "NB HEX POSITIF ENTER":GOTO 5
100  T=T+C*16 ^ P
110  NEXT
120  PRINT            "HEX  ";H$;"  EN DECIMAL VAUT";T
130  GOTO 5
```

3. POUR PASSER EN MODE X AVEC NAS-SYS

```
10  DOKE 3189 , 1922
20  DOKE 3187 , 1917
20  POKE 3112 , <option>
```

La valeur option peut être 0,1,16,17,32,33,48,ou 49.Ainsi,pour XD,l'option est 0.
Ce programme peut servir à mettre en marche une imprimante série sous contrôle programme.

4. POUR PASSER EN MODE N AVEC NAS-SYS

```
10  DOKE 3189 , 1922
20  DOKE 3187 , 1919
```

Ce programme peut servir à couper une imprimante série.

5. POUR PASSER EN MODE U AVEC NAS-SYS

```
10  DOKE 3189 , 1921
20  DOKE 3187 , 1918
```

Ceci permet de passer en mode U pour les drivers I/O écrits par les utilisateurs.

6. POUR PASSER EN MODE CLAVIER K AVEC NAS-SYS

```
10  POKE 3111,<option>
```

Les options 0,1,4,5 correspondent aux modes normal,machine à écrire,machine à écrire ou graphique,graphique.

7. AUSCULTATION DU CLAVIER POUR LES ENTREES AVEC NAS-SYS

Le programme suivant fait un appel à USR, qui ausculte le clavier et renvoie la valeur ASCII du caractère tapé sur le clavier.(Ou 0 en l'absence de caractère)

```
10  DOKE 3200 , 25311
20  DOKE 3202 , 312
30  DOKE 3204 , 18351
40  DOKE 3206 , 10927
50  DOKE 3208 , -8179
60  POKE 3210 , 233
70  DOKE 4100 , 3200
```

L'appel de USR peut par exemple être fait de la manière suivante:

```
30  IF USR(0)<>THEN GOTO $00
```

Le programme USR est le suivant:

```
                RST $CAL
0C80 DF 62      DEFB ZIN        ;auscultation des entrées
0C82 38 01      JR C, CHAR      ;passer si caractère
0C84 AF         XOR A           ;vidage de A
0C85 47         CHAR LD B, A    ;mettre car. dans B
0C86 AF         XOR A           ;vidage de A
0C87 2A 0A 10   LD HL(100A)     ;prendre adresse dans HL
0C8A E9         JP (HL)         ;saut et retour
```

A N N E X E I

PROGRAMME DE CHARGEMENT A UTILISER AVEC NASBUG T2

```
0E00   CD   51   00   CD   3E   00   FE   FF
0E08   20   0D   06   03   CD   3E   00   FE
0E10   FF   20   04   10   F7   18   12   FE
0E18   1E   20   E8   06   03   CD   3E   00
0E20   FE   1E   20   E2   10   F7   C3   51
0E28   00   CD   3E   00   6F   CD   3E   00
0E30   67   CD   3E   00   5F   CD   3E   00
0E38   57   0E   00   CD   70   0E   CD   3E
0E40   00   39   C2   62   0E   43   0E   00
0E48   CD   3E   00   77   81   4F   23   10
0E50   F7   CD   3E   00   39   28   08   EF
0E58   45   72   72   6F   72   1F   00   C3
0E60   03   0E   CD   79   0E   AF   3A   C2
0E68   03   0E   CD   51   00   C3   86   02
0E70   CD   7E   0E   E3   CD   7E   0E   E3
0E78   C9   3E   1F   C3   4A   0C   E5   D5
0E80   CD   32   02   01   E1   C9
```

MANUEL DE PROGRAMMATION BASIC 8K NASCOM 1

ADDITIF

- P.2 1.6 Edition de programme

Le paragraphe suivant est à ajouter:

Lorsque RUBOUT (Control Z) est tapé, le signe "\" apparait, suivi du caractère à effacer. Chaque appui sur RUBOUT imprime le caractère suivant devant être enlevé. L'entrée d'un nouveau caractère imprime "\", suivi du nouveau caractère entré.

Ex: 100 X=\=X\Y=10
L'entrée de 2 RUBOUTS supprime "=" et "X" qui sont remplacés par "Y=".

- P.4 Description de l'instruction DIM

Remplacer "R2%" par "R2", et "Z≠" par "Z".

- P.11 6.4 Caractères spéciaux

.a- Lire pour le premier des caractères spéciaux:

*@ (Escape ou Shift + NL pour Nas-Sys)
Efface la ligne en cours et exécute un retour chariot.

.b- L' astérisque * manque devant RUBOUT, pour indiquer que cette instruction n'est pas utilisable en mode normal avec Nas-Sys.

- P.12 Message d'erreur FC

Supprimer les mentions "STRING$, SPACE$, INSTR".

- P.13 Mise en fonctionnement du BASIC

Avec le moniteur T4 et après "... de la manière habituelle.", ajouter:
"Le moniteur ignorera automatiquement le programme loader. Bien entendu, on peut aussi dérouler la cassette pour éviter la prise en compte du loader".

- P.17 Annexe D, BASIC et language machine

. ligne 12 3004H et 3005H (et non 1004H et 1005H)
. ligne 20 1008H et 1009H (et non E008H et E009H), octet bas en 100BH
. ligne 26 100AH et 100BH (et non E00AH et E00BH)

- P.20 Annexe H, programme N°2

Il vaut mieux écrire les lignes 60 et 70 ainsi:

```
60 IF C>=48 AND C<=57 THEN ...
70 IF C>=65 AND C<=70 THEN ...
```

- P.20 Annexe H, programme N°3

Lire " 10 DOKE 3189,1925 " et non " 10 DOKE 3189,1922 ".

- P.21 Remplacer "80 IF USR(0)<> THEN GOTO 500"
par: "80 IF USR(0)<>0 THEN GOTO 500"

JCS COMPOSANTS
25 RUE DES MATHURINS
75008 - PARIS

CE DOCUMENT DOIT NOUS ETRE RETOURNE DATE ET ACCEPTE. MERCI.

à concevoir par Public - Jaton

CONTRAT D'ACHAT ET D'UTILISATION DE LOGICIEL

JCS Composants (ou tout Agent sous-nommé agréé par JCS Composants) agissant en son nom ou pour le compte de son fournisseur, vend à l'Acheteur le droit d'utiliser le logiciel désigné ci-dessous, sous réserve de l'accord de celui-ci sur les conditions de ce contrat.

En cas de désaccord sur ces conditions, et avant toute utilisation, l'Acheteur devra retourner à JCS Composants ou à son représentant le logiciel et la documentation qui l'accompagne, qui lui seront remboursés.

1 - L'Acheteur s'engage à employer le logiciel concerné sur un seul système micro-informatique pour lequel il a été conçu par le fournisseur.

Autant de contrats d'achat et d'utilisation de logiciel devront être établis que l'Acheteur possède de systèmes.

2 - Le logiciel objet de ce contrat ne peut être ni prêté ni cedé à un tiers sans accord préalable écrit de JCS Composants ou du fournisseur.

3 - L'Acheteur ne peut ni copier ni dupliquer le logiciel quelque soit sa forme (listing, cassettes, disques, mémoire morte), si ce n'est pour son usage propre, afin de le modifier ou de l'intégrer dans un ensemble de programmes.

Cependant, le nombre de telles copies sera limité à 5. Dans tous les cas, la mention de Copyright devra subsister, même si le logiciel est inclu dans un ensemble plus vaste, ou modifié pour satisfaire les besoins particuliers de l'Acheteur. Dans ce cas, il reste toujours la propriété du fournisseur, et le présent contrat s'applique à l'ensemble du programme comprenant le logiciel.

4 - L'Acheteur s'engage à ne pas dupliquer et diffuser tout ou partie de la documentation accompagnant le logiciel. Des copies supplémentaires peuvent être acquises auprès de JCS Composants.

Cependant, pour les besoins propres de l'Acheteur, celui-ci peut dupliquer partiellement les textes ou listings fournis dans la limite de 5 exemplaires par logiciel achetés.

5 - Le présent contrat peut prendre fin à l'initiative de l'Acheteur, ou à l'initiative de JCS Composants ou du fournisseur dans le cas où toutes les clauses de cet accord ne seraient pas respectées.

Dans le cas de cessation, l'Acheteur accepte de retourner à JCS Composants tous le logiciel et la documentation qui l'accompagne, (originaux et toutes les copies faites). L'Acheteur établira alors une déclaration de rénonciation à l'usage du logiciel, soit dans sa forme originale, soit modifié ou intégré à d'autres programmes, et affirmant,qu'à sa connaissance,toutes les copies partielles ou totales ont été détruites.

6 - Le logiciel est garantie pendant une durée de trois mois quant à sa conformité par rapport aux spécifications du fournisseur. Cette garantie ne porte que sur le logiciel livré dans sa forme originale, à la condition que celle-ci n'ait pas été changée ou altérée par l'utilisateur.

7 - En aucun cas JCS Composants, le fournisseur, ou l'Agent de distribution agréé ne pourront être tenus pour responsable de dommages directs ou indirects survenus à l'occasion de l'utilisation du logiciel et du matériel qui l'utilise.

8 - Le logiciel faisant l'objet de ce contrat est dénommé : Basic 8K

POUR L'ACHETEUR :

NOM : GIRAUD

SOCIETE :

ADRESSE : 8 cité Charmois B 435
54500 VANDOEUVRE

"LU ET APPROUVE",DATE,SIGNATURE :

lu et approuvé

POUR JCS COMPOSANTS : 15/01/80

DATE ET SIGNATURE DE L'AGENT AGREE :

Lu et approuvé,