

# **Programmation BASIC SANYO PHC-25**



Version du 31/10/2025



Dédicace à tous ceux qui préserve le patrimoine informatique.

Un grand merci à Olipix qui a lancé les GAME JAM.

À tous les contributeurs de la GAME JAM et aux lecteurs du futur.



# Table des matières

Table des matières.....	5
Introduction .....	11
Les variables .....	13
Le BASIC .....	17
Graphismes .....	19
ABS.....	23
AND.....	24
ASC .....	25
CHR\$.....	26
CLEAR .....	27
CLOAD.....	28
CLOAD? .....	29
CLS.....	30
COLOR .....	31
CONSOLE .....	40
CONT.....	42
COS.....	43
CSAVE .....	44
CSRLIN .....	45
CTOFF.....	46
CTON.....	47
DATA .....	48
DEF FN.....	49
DIM.....	51

ELSE .....	52
END .....	53
EXEC.....	54
EXP .....	56
FOR .....	57
FRE.....	58
GOSUB .....	60
GOTO.....	61
IF .....	62
INKEY\$.....	63
INP .....	64
INPUT .....	65
INPUT# .....	66
INT.....	68
KEY .....	69
LCOPY .....	70
LEFT\$.....	71
LEN.....	72
LET .....	73
LINE.....	74
LIST.....	76
LLIST.....	78
LOCATE.....	80
LOG .....	81
LPOS .....	82
LPRINT.....	83

MID\$.....	84
NEW .....	85
NEXT.....	86
NOT .....	87
ON GOSUB .....	88
ON GOTO.....	90
OR .....	91
OUT.....	92
PAINT.....	93
PEEK.....	94
PLAY .....	95
POINT .....	97
POKE .....	98
POS .....	99
PRESET .....	100
PRINT.....	101
PRINT#.....	103
PSET .....	104
READ .....	105
REM .....	106
RESTORE.....	107
RETURN .....	108
RIGHT\$ .....	109
RND.....	110
RUN.....	112
SCREEN.....	113

SCREEN 1 .....	115
SCREEN 2 .....	116
SCREEN 3 .....	117
SCREEN 4 .....	119
SCRIN .....	121
SGN .....	122
SIN .....	123
SLOAD .....	124
SOUND .....	125
SPC .....	126
SQR .....	127
SSAVE .....	128
STEP .....	129
STICK .....	130
STOP .....	131
STRIG .....	132
STR\$ .....	133
TAB .....	134
TAN .....	135
TIME .....	136
THEN .....	137
TO .....	138
USR .....	139
VAL .....	141
Mémoire du PHC-25 .....	145
L'écran du SANYO PCH-25 .....	150

Jeu de caractères .....	163
Trucs et Astuces .....	166
Jeux Graphique.....	180
INP et OUT .....	183
Dossier technique .....	187
Messages d'erreur .....	193



# Introduction

Reprise des documentations sur le BASIC du SANYO PCH-25.

- Notice d'utilisation
- Classeur bleu

La documentation étant peu loquace, le but est d'avoir un document le plus juste et le plus complet possible.

Les tests de syntaxes et autres ont été effectuées pour la plupart sur émulateur.

D'autres ont pu être fait sur des machines réelles. Ceci dans le but de clarifier toutes les instructions BASIC.

NDR : à date aucune référence littéraire trouvé pour cet ordinateur.

Le second but de ce document c'est la **GAME JAM 2025** sur **SANYO PHC-25**.

De façon à fournir une bonne base de documentation pour pouvoir créer un logiciel.

Il est toujours possible de participer aux GAME JAM après coup.

C'est d'ailleurs très utile pour compléter les logithèques de ces machines obscures.

La programmation du **Z80** n'est pas abordée ici.

Toutefois, cette documentation essaye d'expliquer le How To pour un programme binaire au travers les instructions pour le langage machine.

La documentation étant quasi nulle sur le sujet de l'assembleur dans un PHC-25, pas du Z80 dont on trouve pléthore documentation.

Pour les **INP** et **OUT** du **SANYO PHC-25** nous aimerais bien avoir plus d'indication.



## **Les variables**

Dans la documentation, elles sont nommées constantes.

Considérant le peu de place en mémoire leur étant réservée, ce n'est pas trop étonnant qu'elles s'appellent constante.

C'est un peu alambiqué, on ne parlera que de variables.

## **Chaîne de caractères**

Elle est constituée par un ou plusieurs caractères enfermés entre guillemets. La forme générale étant "caractère ..."

Exemple :

"**PHC 25**"

NB : Les guillemets ("") ne sont pas compris dans la chaîne de caractères.

Utilisée pour mémoriser des chaînes de caractères jusqu'à 255 caractères. Les chaînes de caractères utilisent le signe \$ après le nom variable.

Exemple : P\$, DX\$, XI\$, CD\$

Une restriction du BASIC fait qu'il n'est possible d'avoir que 2 caractères pour le nom de la chaîne de caractère.

Exemple :

**AB\$**  
**ABC\$**

Sont une même variable.

Il en va de même avec la fonction [DEF FN](#), 2 caractères uniquement.

## **Valeurs**

Il y a 3 notations possible, voir ci-après. Il est possible de stocker des nombres de  $10^{-39}$  à  $10^{+38}$ .

Il n'y a pas de possibilité de notation binaire.

## **Notation point (.) fixe**

On obtient un nombre négatif ou positif qui inclut le point décimal avec neuf chiffres significatifs.

Exemples :

33.169  
-45.765

## **Notation décimal flottant**

On obtient un nombre positif ou négatif exprimé en notation exponentielle. L'exposant va de -38 à +38.

La mantisse peut avoir jusqu'à 9 chiffres significatifs.

Exemple :

123456789 E-38

## **Notation hexadécimale**

La constante hexadécimale est exprimée par une combinaison de caractères de 0 à F précédés par &H.

Exemple :

&HC000

Comme il s'agit de la notation du PHC-25, toutes les valeurs hexadécimales dans ce livre l'utilise.

La notation générale utilise souvent le signe « \$ ».

## **Tableau**

Le groupage de données peut être fait sous le terme TABLEAU. La dimension du tableau est définie par la valeur entre parenthèses. L'indice commence à 0.

Exemple :

**A(3)**  
**K\$(4)**

A contient 3 éléments [0,2].

K contient 4 éléments de type chaîne de caractère.

Il est possible d'avoir des tableaux à n dimensions.

Exemple :

**X(3,3,3)**

## **Nom des variables**

Tous les mots clef du BASIC ne peuvent pas être des noms de variables.

- Un nom de variable doit se coder sur 2 caractères.
- Seul les 2 premiers caractères sont pris en considération.
- Un nom de variable ne peut pas débuter par un chiffre : 0 à 9
- Un nom de variable débute par une lettre [A-Z].  
Suivi d'une lettre ou d'un chiffre [A-Z | 0-9].
- Pour les chaines de caractères, le nom est suivi du signe « \$ ».

Utiliser les recommandations et conventions ci-dessous.

Pour les boucles, la recommandation et de se réserver les lettres I, J et K.

Pour les Sous routines, on doublera ces lettres pour avoir II, JJ et KK. Une sous-routine pouvant être appelée par une boucle.

Éviter la lettre L, qui peut se confondre avec 1 et I.

Pour les coordonnées, la convention est d'utiliser X, Y et Z. Pouvant être suivi d'un chiffre ou d'une lettre pour définir n coordonnées.

Pour l'INPUT clavier, il est recommandé d'utiliser K\$ (Key, Keyboard)

# Le BASIC

## **Principe**

Les instructions sont mises dans des lignes de code. Ces lignes sont numérotées.

Ce numérotage va de 0 à 65535.

Une ligne numéro 0 est possible, c'est une particularité du BASIC du PHC-25.

## **Instructions par ligne**

Pour mettre Plusieurs instructions sur une ligne de code, on sépare les instructions par « : ».

Tous les espaces peuvent être supprimés dans une ligne de code.

## **Les instructions**

L'ensemble des instructions du PHC-25 sont décrites ci-après.

Il s'agit d'une reprise du document officiel, avec des mises à jours et explications plus précises.  
Elles sont rangées par ordre alphabétique.

## **Les entiers**

Il n'y a pas de possibilité d'avoir des entiers comme dans la plupart des autres BASIC.

La notation :

```
10 I%=10
```

N'est pas possible.

Pour adapter un programme qui utilise ce type de variable au PHC-25, il faudra recoder et utiliser l'instruction [INT](#).

**NDR : la documentation indique le contraire, c'est à vérifier sur une machine physique.**

## **Trigonométrie**

Le PHC-25 travail en radian. De plus la variable PI n'est pas défini, mais son caractère oui. Si vous désirez utiliser les fonctions trigonométriques dans votre programme, il faut donc prévoir des choses.

Définir la variable PI :

```
PI=3.141592653
```

Nous utilisons 9 décimales, ce qui correspond aux possibilités du PHC-25.  
Cela implique que vous devez réserver ce nom de variable. Il est donc judicieux de le considérer comme une instruction.

Conversion en degré :

```
DEF FN DG(R)=R*(180/PI)
```

Nous utilisons la fonctionnalité DEF FN pour généraliser.  
Ce n'est pas obligatoire et vous pouvez simplement appliquer la formule.

Conversion en radian :

```
DEF FN RD(D)=D*(PI/180)
```

Nous utilisons la fonctionnalité DEF FN pour généraliser.  
Ce n'est pas obligatoire et vous pouvez simplement appliquer la formule.

# Graphismes

Les SCREEN 1 et 2 sont dédiés à des interactions de type texte. Même s'il est possible de faire quelques trucs graphiques.

Avec une bonne dose de compréhension et de courage il est possible d'arriver à des choses bien avec le mode 2.

Les SCREEN 3 et 4 sont dédiés aux graphismes.

Cela implique que l'instruction INPUT ne fonctionnera pas sur ce type d'écran.

INPUT forcera un mode texte (1 ou 2), plus généralement le mode 1.

Ce pourquoi au démarrage il est demandé 1 ou 2 écrans.

Une sous-routine simulant l'INPUT est possible au sacrifice de la ligne sur laquelle faire la saisie.

La recommandation dans ce cas là est d'utiliser la ligne 15.

Considérer alors une résolution de 256x180 pixel. Une ligne texte occupant 12 lignes graphique.

L'instruction PRINT fera des équivalents de PSET() vis-à-vis des points visibles du caractère.

C'est une forme de surimpression.



# **Les Instructions du BASIC**



# **ABS**

## ***Objet***

Renvoie la valeur absolue de l'argument fourni.

## **Syntaxe**

**ABS (x)**

x : Nombre. Peut aussi être une instruction qui renvoie un nombre.

## ***Exemple***

```
10 PRINT ABS(9*(-2))
```

Aura comme résultat 18.

TO DO : exemple avec fonction.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

DEF FN

# **AND**

## ***Objet***

Cette instruction n'est pas documentée.

Provoque un ET logique avec les 2 variables fournies.

Table AND :

<b>A</b>	<b>B</b>	<b>R</b>
0	0	0
0	1	0
1	0	0
1	1	1

## ***Syntaxe***

**a AND b**

a et b doivent être des valeurs numériques.

## ***Exemple***

**10 A = B AND C**

**100 IF A=1 AND B=0 THEN 1000**

## ***Adresse mémoire de l'instruction***

**&H**

## ***Mots clés associés***

**IF, NOT, OR**

# **ASC**

## ***Objet***

Renvoie la valeur du code ASCII du premier caractère d'une chaîne de caractères.

## **Syntaxe**

**ASC (a\$)**

a\$ : Chaîne de caractère.

## ***Exemple***

```
10 A$="TEST"  
20 PRINT ASC(A$)
```

Affiche 84 car la valeur ASCII de la lettre T est 84.

Voir le tableau ASCII.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

CHR\$, LEFT\$, MID\$, RIGHT\$, STR\$, VAL

# **CHR\$**

## ***Objet***

Génère un caractère de code ASCII égal à la valeur fournie en argument.

Selon le mode écran, il y a un comportement différent.

Voir le chapitre sur le [jeu de caractères](#) du PHC-25.

## ***Syntaxe***

**CHR\$ (x)**

x : nombre entier de 0 (&h00) à 255 (&hFF).

## ***Exemple***

```
PRINT CHR$(42)
PRINT CHR$(20)+CHR$(49)
```

NDR : tous les caractères ne sont pas affichables avec CHR\$() et le type de mode écran influence cela aussi.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[ASC](#), [LEFT\\$](#), [MID\\$](#), [RIGHT\\$](#), [SCREEN](#), [STR\\$](#), [VAL](#)

# **CLEAR**

## ***Objet***

Remettre à zéro toutes les variables numériques et à valeur nulle toutes les valeurs alphanumériques.

## **Syntaxe**

```
CLEAR I, [J]
```

I : Taille de la mémoire pour y stocker des données. Obligatoire.

J : Optionnel. Limite supérieure de la mémoire pour le programme BASIC.

## ***Exemple***

```
CLEAR 400, &hE050
```

400 caractères réservé.

Limite de mémoire supérieur &hE050, adresse au-delà de laquelle on met un programme en langage machine.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

**FRE**

# **CLOAD**

## ***Objet***

Charger en mémoire un programme ou un fichier existant sur une cassette.  
CLOAD efface le programme précédent avant de charger celui qui est sur la cassette.

## **Syntaxe**

**CLOAD"nom"**

nom : Le nom est une chaîne dont les six premiers caractères sont significatifs et doivent être identiques au nom sous lequel le programme a été sauvegardé par CSAVE.

Le guillemet de fin n'est pas obligatoire.

CLOAD seul chargera le premier programme de la cassette.

## ***Exemple***

**CLOAD**

Charge le programme depuis le magnétophone.

**CLOAD"othelo"**

Charge le programme « Othelo » depuis le magnétophone. Si le programme n'est pas Othelo continu avec le programme suivant sur la cassette.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[CLOAD?](#), [CSAVE](#), [SLOAD](#), [SSAVE](#)

# **CLOAD?**

## ***Objet***

Une comparaison est faite entre le programme en mémoire et celui qui est sur la cassette.

NDR : Ce type d'instruction était nécessaire car lié à la qualité des médiums et du magnétophone.

## **Syntaxe**

`CLOAD?"nom"`

Nom : nom du programme à tester. Seul les 6 premier caractères comptent.

## ***Exemple***

`CLOAD?"prog"`

Va comparer le programme en mémoire avec le programme « prog » qui est sur la cassette.

## ***Adresse mémoire de l'instruction***

&H

## **Mots clés associés**

[CLOAD](#), [CSAVE](#), [SLOAD](#), [SSAVE](#)

# **CLS**

## ***Objet***

Tous les caractères affichés sur l'écran sont effacés.

Tous les caractères affichés sur l'écran définie par CONSOLE sont effacés.

## **Syntaxe**

**CLS**

## ***Exemple***

**10 CLS**

Efface l'écran.

**10 CONSOLE 3,5  
20 CLS**

Seules les lignes 3 à 8 seront effacées.

Les lignes sont numérotées de 0 à 15.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

COLOR, CONSOLE, SCREEN

# COLOR

## ***Objet***

Désignation de la couleur des caractères, ou graphiques affichés sur l'écran.

Même si on désigne les mêmes chiffres, la couleur affichée est différente selon le mode écran choisi (1,2,3,4).

## **Syntaxe**

```
COLOR a,b,c
```

a : défini selon les modes la couleur du texte,

b : défini selon les modes 2, 3 et 4, la couleur de fond,

c : défini la palette de couleurs utilisées dans les modes écran 1,2,3 et 4.

Ci-après pour chaque mode les couleurs associées et leurs fonctionnements.

## Mode 1

Dans ce mode, la couleur de l'écran est désignée par les valeurs a et c sans tenir compte de b.

La couleur de l'écran entier est changée en exécutant l'instruction COLOR et ensuite CLS.

De même, un changement de palette affecte tout l'écran.

Il n'y a pas de graphique possible en mode 1. Ce mode est textuel, de type console.

Les paramètres, a et c peuvent être utilisés seul ou ensemble.

## Palettes

Il y a 4 couleurs disponibles dans 2 palettes selon le tableau ci-dessous :

La valeur de c définit la palette utilisée.

Palette 1 :

a	c	Caractère	Fond	Affichage
1	1	Vert Clair	Vert	189 ; 255 ; 181
2	1	Vert	Vert Clair	16 ; 132 ; 8
3	1	Blanc	Orange	255 ; 198 ; 173
4	1	Orange	Blanc	255 ; 66 ; 107

La colonne affichage donne une valeur approchée RGB du texte (ou fond si inversion).

Palette 2 :

a	c	Caractère	Fond	Affichage
1	2	Blanc	Orange	255 ; 198 ; 173
2	2	Orange	Blanc	255 ; 66 ; 107
3	2	Vert Clair	Vert	189 ; 255 ; 181
4	2	Vert	Vert Clair	16 ; 132 ; 8

La colonne affichage donne une valeur approchée RGB du texte (ou fond si inversion).

NDR : Semble assez éloigné de ce que dit la documentation. Se fier à une machine physique.

Exemples :

```
COLOR , , 1
```

Passe la console sur la palette 1, Vert et Vert clair. Le changement de palette affecte tout l'écran.

```
COLOR 1
```

Passe le texte en mode Vert Clair sur fond vert. Affecte toutes les instructions qui suivent jusqu'au prochain COLOR.

**COLOR 2**

Passe le texte en mode inverse vidéo Vert sur fond Vert Clair. Affecte toutes les instructions qui suivent jusqu'au prochain COLOR.

```
COLOR 1,1,1  
COLOR 1,2,1
```

Ces 2 instruction on le même résultat, le paramètre b est ignoré.

## Mode 2

La valeur de a définit la couleur des caractères. Il y a 4 couleurs disponibles. Ce sont les mêmes que le mode 1 ci-dessus.

La valeur de b définit la couleur de fond de l'écran. Il y a 9 couleurs disponibles.

La valeur de c définit la palette de couleur utilisée. Il y a 2 palettes disponibles.

Préalable :

```
SCREEN 2,1,1
```

Ci-dessus, le mode 2 est appliqué à l'écran 1 et est affiché.

NB : Le PHC-25 dispose de la possibilité d'avoir 2 écrans au détriment de l'espace mémoire.

## Palettes

Le choix de la palette se fait via la valeur de c.

La valeur de b va peindre le fond, par exemple dans le cas d'un CLS.

Dans le cas des graphiques, fait une inversion entre le noir et la couleur vis-à-vis des fonctions graphiques.

En aucun cas b n'influence l'affichage des caractères.

Palette no 1 :

a	b	c	Caractères	Affichage	Fond	Graphiques
0	0	1	Vert	189 ; 255 ; 181	Noir	Noir
1	1	1	Vert	189 ; 255 ; 181	Vert	Vert
2	2	1	Vert (inversé)	16 ; 132 ; 8	Jaune	Jaune
3	3	1	Orange	255 ; 198 ; 173	Violet	Violet
4	4	1	Orange inversé)	255 ; 66 ; 107	Rouge	Rouge
5	5	1	Orange inversé)		Blanc	Blanc
6	6	1	Orange inversé)		Bleu clair	Bleu clair
7	7	1	Orange inversé)		Rose	Rose
8	8	1	Orange inversé)		Orange	Orange

Exemple :

```
COLOR , ,1
```

Passe sur la palette 1, ceci affecte tout l'écran.

```
10 COLOR ,2  
20 CLS
```

L'écran sera jaune après le CLS.

Palette no 2 :

a	b	c	Caractères	Affichage	Fond	Graphiques
0	0	2	Orange	255 ;198 ;173	Noir	Noir
1	1	2	Orange	255 ;198 ;173	Blanc	Blanc
2	2	2	Orange (inversé)	255 ;66 ;107	Bleu clair	Bleu clair
3	3	2	Vert	189 ; 255 ; 181	Rose	Rose
4	4	2	Vert (inversé)	16 ; 132 ; 8	Orange	Orange
5	5	2	Vert (inversé)		Vert	Vert
6	6	2	Vert (inversé)		Jaune	Jaune
7	7	2	Vert (inversé)		Violet	Violet
8	8	2	Vert (inversé)		Rouge	Rouge

Exemple :

```
COLOR , ,2
```

Passe sur la palette 2, ceci affecte tout l'écran.

Remarque :

Les couleur Rouge et Vert, correspondent au couleur de fond de base des caractères. Il peut être opportun de les utiliser quand il y a besoin de texte à afficher.

Exemple :

```
1000 SCREEN 2,1,1
1010 COLOR , ,1
1020 CLS
1030 PSET (120,120) ,3
1040 LINE(12,12)-(48,48) ,2,BF
1050 LOCATE 5,12:PRINT"Hello World!"
9000 GOSUB 9980
9910 END
9980 K$="" :K$=INKEY$ :IF K$="" THEN 9980
9990 RETURN
```

NDR : Il est techniquement possible d'avoir 11 couleurs à l'écran en jouant sur la couleur de fond des textes. En utilisant l'espace pour peindre dans la couleur. Le bloc est entier et il ne faut rien appliquer dessus.

## **Mode 3**

La valeur de a, définit la couleur des caractères. Il y a 4 couleurs disponibles.

La valeur de b définit la couleur de fond de l'écran. Il y a 5 couleurs disponibles.

La valeur de c définit la palette de couleur utilisée. Il y a 2 palettes disponibles.

## **Palettes**

Palette 1 :

TO DO

Palette 2 :

TO DO

NDR : La documentation originale est insuffisante.

## Mode 4

Ce mode est assez particulier pour définir les couleurs.

Il s'agit d'un mode bicolor avec 4 variantes.

Noter qu'il n'est pas possible de faire de l'affichage texte en inverse.

### Palettes

Pour choisir la variante, il faut que les valeurs de a et b soient identiques puis fixer c à 1 ou 2 avec l'instruction COLOR. Puis enchaîner avec l'instruction CLS.

Variantes et couleurs :

a	b	c	Caractères	Fond	Graphiques
0	0	1	Vert Clair	Vert	Vert clair
1	1	1	Vert	Vert clair	Vert
0	0	2	Blanc	Noir	Blanc
1	1	2	Noir	Blanc	Noir

Exécuter l'instruction COLOR a,b,c suivi de CLS pour activer la variante.

L'instruction COLOR „c permet de changer de variante. Mais uniquement sur 2 possibilités (a et b = 0 ou a et b = 1).

## Exemples

Variante 1.

```
10 SCREEN 4,1,1
20 COLOR 0,0,1:CLS
30 COLOR 1
40 PRINT "Hello World!"
50 LINE (100,100)-(150,150),1,BF
1000 K$=INKEY$:IF K$="" THEN 1000
```

Passe sur la palette 1. Les valeurs a et b étant identique (à 0), met le fond et le texte en Vert lorsque CLS s'exécute.

L'instruction COLOR 1 passe le texte en Vert clair.

De même le paramètre de couleur pour les instructions graphiques (0 Vert ; 1 Vert Clair).

Variante 2.

```
10 SCREEN 4,1,1
20 COLOR 1,1,1:CLS
30 COLOR 0
40 PRINT "Hello World!"
50 LINE (100,100)-(150,150),0,BF
1000 K$=INKEY$:IF K$="" THEN 1000
```

Passe sur la palette 1. Les valeurs de a et b étant identique (à 1), met le fond et le texte en Vert clair lorsque CLS s'exécute.

L'instruction COLOR 0 passe le texte en Vert.

De même le paramètre de couleur pour les instructions graphiques (0 Vert ; 1 Vert Clair).

Variante 3.

```
10 SCREEN 4,1,1
20 COLOR 0,0,2:CLS
30 COLOR 1
40 PRINT "Hello World!"
50 LINE (100,100)-(150,150),1,BF
1000 K$="" :K$=INKEY$:IF K$="" THEN 1000
```

Passe sur la palette 2. Les valeurs a et b étant identique (à 0), met le fond et le texte en Noir lorsque CLS s'exécute.

L'instruction COLOR 1 passe le texte en Blanc.

De même le paramètre de couleur pour les instructions graphiques (0 Noir ; 1 Blanc).

Variante 4.

```
10 SCREEN 4,1,1
20 COLOR 1,1,2:CLS
30 COLOR 0
40 PRINT "Hello World!"
50 LINE (100,100)-(150,150),0,BF
1000 K$="" :K$=INKEY$:IF K$="" THEN 1000
```

Passe sur la palette 2. Les valeurs de a et b étant identique (à 0), met le fond et le texte en Blanc lorsque CLS s'exécute.

L'instruction COLOR 0 passe le texte en Noir.

De même le paramètre de couleur pour les instructions graphiques (0 Noir ; 1 Blanc).

## **Addendum**

Les valeurs a, b, c, peuvent varier de 0 à 255 avec des effets de bord selon le mode choisi.  
Il est déconseillé de sortir des valeurs définies dans les modes décrit ci-avant.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[CLS](#), [LOCATE](#), [PRINT](#), [SCREEN](#)

# **CONSOLE**

## ***Objet***

Désigne les limites de déplacement vers le bas et le haut de l'écran.  
Seules les lignes désignées vont scroller ou être effacées.

Faire un [PRINT](#) sur la dernière ligne définie par CONSOLE entraînera un scroll si le PRINT n'est pas suivi d'un « ; ».

Sur les modes écran 1 et 2, la CONSOLE reste active après la fin de programme.  
Il peut s'avérer utile de faire un CONSOLE 0,16 avant le [END](#) du programme.

Définir une 2<sup>ème</sup> instruction CONSOLE provoque un effet de bord. Cette nouvelle instruction s'appliquant à la zone précédente définie par CONSOLE.

Il est donc nécessaire de faire CONSOLE 0,16 avant de faire une nouvelle sélection par CONSOLE.

Pour les modes 3 et 4, CONSOLE fonctionne aussi. Cela permet d'effacer un bloc de lignes si nécessaire. Consulter [SCREEN](#) pour comprendre mieux ce qui se passe.

## **Syntaxe**

**CONSOLE a,b**

a : ligne de départ

b : nombre de ligne (>0) – [1,254]

## ***Exemple***

```
10 CLS
20 PRINT "TITRE"
30 CONSOLE 1,15
40 CLS
```

Le titre n'est pas effacé par le CLS de la ligne 40.

```
CONSOLE 0,16
```

Tout l'écran. Depuis la ligne 0 et sur 16 lignes.

## ***Adresse mémoire de l'instruction***

&H

### ***Mots clés associés***

**CLS, LOCATE**

# **CONT**

## ***Objet***

CONT veut dire continuer.

Utilisé pour reprendre l'exécution d'un programme après une suspension par la commande CTRL/C ou après l'exécution d'une instruction STOP.

L'exécution reprend à l'endroit où elle a été interrompue.

CONT n'est plus utilisable si une modification quelconque a été faite dans le programme, ni après une instruction END.

L'écran affiche « Can't continue ».

## ***Syntaxe***

CONT

## ***Exemple***

CONT

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

END, STOP

# **COS**

## ***Objet***

Le cosinus de X exprimé en radians.  
Le résultat est donné en simple précision.

## **Syntaxe**

**COS (x)**

X : angle en radians.

## ***Exemple***

**PRINT COS (0.4)**

Affichera 0.921060995.

## ***Adresse mémoire de l'instruction***

&H

## **Mots clés associés**

[DEF FN](#), [SIN](#), [TAN](#)

# **CSAVE**

## ***Objet***

Sauvegarde sur cassette d'un programme en mémoire.

Remarques : Chaque programme sauvegardé est identifié par un nom.

Le BASIC copie le programme existant en mémoire sur la cassette lorsque la commande CSAVE est exécutée. Elle se sert des six premiers caractères pour le « nom ».

Avant l'utilisation d'un CSAVE, il est nécessaire de vérifier que le magnétophone est branché correctement et que les touches RECORD et PLAY sont bien enfoncées.

## ***Syntaxe***

```
CSAVE Programme
```

## ***Exemple***

```
CSAVE "MonProgramme"
```

La référence sur la cassette du programme sera « MonPro ». Les 6 premier caractères.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[CLOAD](#), [CLOAD ?](#), [SSAVE](#)

# **CSRLIN**

## ***Objet***

Obtient la ligne de la position du curseur.

## **Syntaxe**

CSRLIN

## ***Exemple***

```
10 LOCATE 12,7  
20 PRINT CSRLIN
```

Le curseur est en position colonne 12, ligne 7, le programme affichera 7.

## **Adresse mémoire de l'instruction**

&H

## **Mots clés associés**

LPOS, POS

# **CTOFF**

## ***Objet***

Fermer l'interrupteur de commande à distance du magnétophone (REMrte Control) arrêtant ainsi le défilement de la cassette.

Sous condition que les connecteurs de prise soient correctement câblés.

## **Syntaxe**

CTOFF

## ***Exemple***

```
1000 PRINT "APPUYER SUR LA TOUCHE PLAY DU MAGNÉTOPHONE"  
1010 PRINT "APPUYER ENSUITE SUR UNE TOUCHE"  
1020 K$=INKEY$:IF K$="" THEN 1020  
1030 CTON  
1040 ...  
1050 CTOFF
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[CLOAD](#), [CSAVE](#), [CTON](#), [SSAVE](#), [SLOAD](#)

# **CTON**

## ***Objet***

Mettre le magnétophone en marche dans le cas où la touche PLAY est enfoncée.  
La commande à distance du magnétophone est enclenchée par cette commande, démarrant ainsi la cassette.

Vérifier que la touche PLAY du magnétophone est bien enclenchée.  
Sous condition que les connecteurs de prise soient correctement câblés.

## **Syntaxe**

CTON

## ***Exemple***

```
1000 PRINT "APPUYER SUR LA TOUCHE PLAY DU MAGNÉTOPHONE"
1010 PRINT "APPUYER ENSUITE SUR UNE TOUCHE"
1020 K$=INKEY$:IF K$="" THEN 1020
1030 CTON
1040 ...
1050 CTOFF
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[CLOAD](#), [CSAVE](#), [CTOFF](#), [SSAVE](#), [SLOAD](#)

# **DATA**

## ***Objet***

Utilisé pour enregistrer dans un programme des valeurs numériques et des caractères constants qui seront appelés par des instructions [READ](#).

Les chaînes de caractères doivent être encadrées de guillemets

Les lignes DATA peuvent se situer n'importe où dans le programme.

## ***Syntaxe***

```
DATA valeur[, valeur, ...]
```

## ***Exemple***

```
10 FOR I=1 TO 3: READA$(I): NEXT I
20 FOR I=1 TO 3: READB(I): NEXT I
30 FOR I=1 TO 3: PRINT A$(I); "="; B(I); "F": NEXT I
100 DATA "PAIN", "VIN", "JAMBON", 2, 6.50, 10
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[READ](#), [RESTORE](#)

# DEF FN

## Objet

Définir une fonction écrite par l'utilisateur et lui donner un nom.

On ne peut pas passer de chaîne de caractères en paramètre (**Type mismatched**).

Le nom ne peut pas être de type chaîne de caractère (**Type mismatched**).

Il est possible de se passer des espaces séparateurs.

NDR : La documentation indique qu'il serait possible d'avoir N paramètre séparé par une virgule.  
Apparemment ce n'est pas fonctionnel, il faudrait approfondir le sujet.

## Syntaxe

```
DEF FN <nom>(x)=<fonction>
```

nom : le nom donné à la fonction. **Le nom doit avoir 2 caractères maximum.**

x : le paramètre.

fonction : la formule de la fonction.

## Exemples

```
1000 REM
1010 DEF FN SEC(X)=1/COS(X) : REM secante
1020 DEF FN CSC(X)=1/SIN(X) : REM cosecante
1030 DEF FN COT(X)=1/TAN(X) : REM cotangente
1040 DEF FN SINH(X)=(EXP(X)-EXP(-X))/2: REM sinus hyperbolique
1050 DEF FN COSH(X)=(EXP(X)-EXP(-X))/2: REM cosinus hyperbolique
1060 DEF FN TANH(X)=EXP(-X)/EXP(X)+EXP(-X))*2+1: REM tangente hyperbolique
1070 DEF FN SECH(X)=2/(EXP(X)+EXP(-X)): REM secante hyperbolique
1080 DEF FN CSCH(X)=2/(EXP(X)-EXP(-X)): REM cosecante hyperbolique
1090 DEF FN COTH(X)=EXP(-X)/(EXP(X)-EXP(-X))*2+1: REM cotangente hyperbolique
1100 DEF FN ARCSINH(X)=LOG(X+SQR(X*X+1)): REM inverse sinus hyperbolique
1110 DEF FN ARCCOSH(X)=LOG(X+SQR(X*X-1)): REM inverse cosinus hyperbolique
1120 DEF FN ARCTANH(X)=LOG((1+X)/(1-X))/2: REM inverse tangente hyperbolique
1130 DEF FN ARCSECH(X)=LOG((SQR(-X*X+1)+1)/X): REM inverse secante hyperbolique
1140 DEF FN ARCCSCH(X)=LOG((SGN(X)*SQR(X*X+1))/X): REM inverse cosecante hyperbolique
1150 DEF FN ARCCOTH(X)=LOG((X+1)/(X-1))/2: REM inverse cotangente hyperbolique
```

Défini des fonctions supplémentaires de trigonométrie. Dans un BASIC Standard.

Revoir le code pour tout définir en 2 caractères.

```

1000 REM
1010 DEF FN SE(X)=1/COS(X) : REM secante
1020 DEF FN CS(X)=1/SIN(X) : REM cosecante
1030 DEF FN CT(X)=1/TAN(X) : REM cotangente
1040 DEF FN SH(X)=(EXP(X)-EXP(-X))/2: REM sinus hyperbolique
1050 DEF FN CH(X)=(EXP(X)-EXP(-X))/2: REM cosinus hyperbolique
1060 DEF FN TH(X)=EXP(-X)/(EXP(X)+EXP(-X))*2+1: REM tangente hyperbolique

1070 DEF FN HS(X)=2/(EXP(X)+EXP(-X)): REM secante hyperbolique
1080 DEF FN HC(X)=2/(EXP(X)-EXP(-X)): REM cosecante hyperbolique
1090 DEF FN HT(X)=EXP(-X)/(EXP(X)-EXP(-X)*2+1): REM cotangente hyperbolique

1100 DEF FN ARCSINH(X)=LOG(X+SQR(X*X+1)): REM inverse sinus hyperbolique
1110 DEF FN ARCCOSH(X)=LOG(X+SQR(X*X-1)): REM inverse cosinus hyperbolique
1120 DEF FN ARCTANH(X)=LOG((1+X)/(1-X))/2: REM inverse tangente hyperbolique
1130 DEF FN ARCSECH(X)=LOG((SQR(-X*X+1)+1/X)): REM inverse secante hyperbolique
1140 DEF FN ARCCSCH(X)=LOG((SGN(X)*SQR(X*X+1)/X): REM inverse cosecante hyperbolique
1150 DEF FN ARCCOTH(X)=LOG((X+1)/(X-1))/2: REM inverse cotangente hyperbolique

```

NDR : \$à\$\$ !!!!

```

1010 DEF FN SE(X)=1/COS(X)

1010 DEFFNSE(X)=1/COS(X)

```

Les 2 lignes sont équivalentes.

Exemple :

```

10 DEFFNF(X)=3*X+6
20 PRINT FN F(5)

```

Affichera 21

## ***Adresse mémoire de l'instruction***

&H

## **Mots clés associés**

Toutes fonctions.

# **DIM**

## ***Objet***

Préciser les valeurs maximales des indices pour les tableaux et réserver la place utile en mémoire.  
En même temps on assure la place utile pour ranger la mémoire.

Lorsqu'un tableau de variables est utilisé sans instruction DIM, la valeur maximum est considérée comme étant 10 lignes en 3 dimensions.

**NDR :** Valeur à calculer/tester.

Une erreur apparaît si l'on fait appel à une valeur plus grande.

La valeur minimale est zéro.

## **Syntaxe**

```
DIM variable[,variable, ...]
```

## ***Exemple***

```
10 DIM B$(20,10), C(30), E(10,3,5,7)
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[FRE](#),

## **ELSE**

Suivant les résultats d'un test, l'on prendra une décision ou l'on choisira une suite pour l'exécution du programme.

Si le résultat du test est vrai, c'est-à-dire différent de zéro, alors l'instruction [THEN](#) suivra jusqu'à ELSE ou la fin de la ligne et sera exécutée.

Si l'expression est égale à zéro (valeur fausse), seules les instructions suivant ELSE, si elles existent, seront exécutées.

## **Syntaxe**

```
IF <expression> THEN <instruction> ELSE <instruction>
```

Expression : Expression à tester

Instruction : toutes instructions valides du BASIC, s'il y en a plusieurs, les séparer par « : ».

L'exception étant l'instruction [REM](#).

## **Exemple**

```
5 A=-5: B=-10
10 IF A<>0 THEN B=10 ELSE B=0: GOTO 100
30 PRINT A,B
40 END
100 PRINT A,B
110 A=A+1
120 GOTO 10
```

## **Adresse mémoire de l'instruction**

&H

## **Mots clés associés**

[AND](#), [IF](#), [NOT](#), [THEN](#), [OR](#)

# **END**

## ***Objet***

Terminer l'exécution d'un programme.

L'instruction END peut être placée n'importe où dans le programme pour terminer l'exécution.

Le message BREAK n'est pas imprimé lorsque l'instruction END est exécutée.

La commande [CONT](#) ne peut être utilisée après un END.

## ***Syntaxe***

**END**

## ***Exemple***

**9990 END**

Le programme s'arrête lorsqu'il atteint la ligne 9990.

**NDR :** Le PHC-25 revient sur un SCREEN 1 (Texte). Cela implique la perte de tous les tracés graphiques. Sauf si utilisation du second écran.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[CONT](#)

# EXEC

## *Objet*

Le programme en basic ira à l'adresse indiquée par la variable X pour exécuter un programme écrit en langage machine (hexadécimal). Le programme aura été écrit grâce à l'instruction POKE et devra se finir automatiquement par la valeur (&HC9) (RET) qui permettra un retour au langage Basic.

## *Syntaxe*

**EXEC &Hx**

X est l'adresse hexadécimale du programme à exécuter.

## *Exemple*

```
10 CLEAR 100,&HF000
20 LET J=00
30 INPUT "valeur Hex";A$
40 N=VAL("&H"+A$)
50 IF A$="ZZ" OR A$="zz" THEN 90
60 POKE &HF000+(J),N
70 J=J+1
80 GOTO 30
90 END
100 CLS
110 EXEC &HF000
120 PRINT "SORTIE"
130 END
```

Valeur à passer : 3E, 41, 32, 8F, 60, C9

Le programme met le caractère A à la position 16,5 de l'écran ([LOCATE](#)). Pour les modes SCREEN 1 et SCREEN 2.

Son adresse mémoire étant &h608F.

Voir chapitre sur [l'écran du SANYO PHC-25](#) pour plus d'information.

## *Adresse mémoire de l'instruction*

&H

## **Mots clés associés**

PEEK, POKE, USR

# **EXP**

## ***Objet***

Renvoie la valeur de e à la puissance x.

## **Syntaxe**

**EXP (x)**

x : Puissance pour e.

## ***Exemple***

```
10 PRINT EXP(6)
```

Retourne : 403.428794.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[DEF FN](#)

# **FOR**

## ***Objet***

FOR NEXT répète et exécute une chaîne de commandes le nombre de fois désigné. La première expression X, est la valeur initiale, Y est une valeur limite à ne pas dépasser. Les instructions entre FOR et NEXT sont exécutées autant de fois que nécessaire avec chaque fois incrémentation du compteur variable de la quantité indiquée après l'instruction STEP pour que le compteur dépasse la limite Y. Dans un tel cas, l'exécution passe à l'instruction suivante NEXT.

## **Syntaxe**

```
FOR variable=X TO Y (STEP Z)
...
NEXT
```

NEXT est obligatoire.

## ***Exemple***

### ***Adresse mémoire de l'instruction***

&H

### **Mots clés associés**

NEXT, STEP, TO

# FRE

## *Objet*

Retourne la valeur de la mémoire disponible.

## *Syntaxe*

**FRE (x)**

x :

**FRE (x\$)**

x\$ :

## *Exemple*

```
10 PRINT FRE(x)
```

Nombre d'octets restant dans la mémoire pour le BASIC.

```
10 PRINT FRE(x$)
```

Nombre d'octets restant dans la mémoire pour les chaînes de caractères.

La valeur de la variable n'a pas d'impact. Elle est présente pour spécifier le type de donnée.

## *Adresse mémoire de l'instruction*

&H

## ***Mots clés associés***

**CLEAR**

# **GOSUB**

## ***Objet***

Aller à un sous-programme et revenir.

## **Syntaxe**

**GOSUB adresse**

adresse : numéro de ligne du sous-programme.

## ***Exemple***

```
1010 PRINT"BCD"
1020 GOSUB 1050
1030 PRINT"DEF"
1040 END
1050 PRINT"123"
1060 RETURN
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

**GOTO**, **ON**, **RETURN**

# **GOTO**

## ***Objet***

Saut inconditionnel à une ligne qui a été désignée.  
Une erreur est signalée si la ligne désignée n'existe pas.

## **Syntaxe**

**GOTO adresse**

adresse : ligne cible du saut inconditionnel.

## ***Exemple***

TO DO : Appliquer pour conversion d'un While Wend et do until.

Le PHC-25 n'a pas d'instruction WHILE WEND, ni d'instruction DO UNTIL.  
Voir le [Chapitre Trucs et Astuces](#) pour effectuer une conversion de programme.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[GOSUB](#), ON

# **IF**

## ***Objet***

Suivant les résultats d'un test, l'on prendra une décision ou l'on choisira une suite pour l'exécution du programme.

Si le résultat du test est vrai, c'est-à-dire différent de zéro, alors l'instruction THEN suivra jusqu'à ELSE ou la fin de la ligne et sera exécutée.

Si l'expression est égale à zéro (valeur fausse), seules les instructions suivant ELSE, si elles existent, seront exécutées.

## ***Syntaxe***

```
IF <expression> THEN <instruction> ELSE <instruction>
```

THEN est obligatoire. Si expression est VRAI alors les instructions sont exécutées.

ELSE est optionnel. Si expression est FAUSSE alors les instructions sont exécutées.

THEN et ELSE doivent être sur la même ligne de code.

## ***Exemple***

```
5 A=-5: B=-10
10 IF A<>0 THEN B=10 ELSE B=0: GOTO 100
30 PRINT A,B
40 END
100 PRINT A,B
110 A=A+1
120 GOTO 10
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

AND, ELSE, NOT, OR, THEN

# **INKEY\$**

## ***Objet***

Obtient le caractère de la touche lorsque l'on appuie sur le clavier.  
Permet de tester quelle touche a été appuyée.

## **Syntaxe**

**INKEY\$**

## ***Exemple***

```
100 IF INKEY$="" THEN GOTO 100
110 IF INKEY$<>"A" THEN GOTO 100
```

Ligne 100, tant qu'aucune touche n'est appuyée retourne à la ligne 100  
Ligne 110, test si la touche A majuscule a été appuyée.

NDR : INKEY\$ mémorise l'appui multiple sur les touches. Le buffer est donc rempli.  
Dans certains cas cela peut provoquer un enchaînement indésirable.

SOLUTION ?

Voir aussi [Trucs et Astuces](#).

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[STICK](#), [STRIG](#)

# **INP**

## ***Objet***

Lecture d'un octet sur le port I (adresse de &H00 à &HFF).  
Voir aussi le chapitre [INP et OUT](#).

## **Syntaxe**

**INP (x)**

x : adresse du port de 0 à 255.

## ***Exemple***

**PRINT INP(&H8F)**

Renvoie la valeur du port &H8F.

## **Adresse mémoire de l'instruction**

&H

## **Mots clés associés**

[OUT](#)

# INPUT

## ***Objet***

Lorsqu'une instruction INPUT est exécutée, le point d'interrogation est affiché, indiquant une demande de l'entrée des données. S'il y a une « chaîne de caractères », elle est affichée devant le point d'interrogation(?).

NDR : La fonction INPUT provoque un retour en SCREEN 1 lors de son utilisation en SCREEN 3 et 4.

## ***Syntaxe***

```
INPUT [chaine,]variable[,variable]
```

Variable peut être une chaîne de caractère ou une variable numérique.

## ***Exemple***

```
INPUT A
INPUT A$
INPUT »Donner votre valeur »,A
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[INPUT#](#)

# **INPUT#**

## **Objet**

Lecture de données dans un fichier séquentiel et affectation des valeurs aux variables de la liste.  
Le numéro du fichier est le numéro associé au fichier au moment de son ouverture.

La liste contient le nom des variables où l'on doit affecter les données existantes sur le fichier.  
Il faut que les types concordent et il n'y a pas de point d'interrogation sur l'écran comme dans le « INPUT ».

Lorsque l'on entre des données par l'intermédiaire du magnétophone, il est nécessaire d'enregistrer ces données par une instruction [PRINT#](#).

## **Syntaxe**

```
INPUT#-n, val
```

N	
0	Clavier
1	Magnétophone

Val : élément à lire, la donnée enregistrée doit correspondre au type renseigné.

## **Exemple**

```
10 INPUT#0-,A
```

NDR: Ne fonctionne pas comme décrit, le « ? » s'affiche.

## **Adresse mémoire de l'instruction**

&H

## **Mots clés associés**

INPUT, PRINT, PRINT#

# **INT**

## ***Objet***

Retourne le plus grand entier inférieur ou égal à X.

## **Syntaxe**

`INT(x)`

x : nombre dont on veut l'entier.

## ***Exemple***

`PRINT INT(99.99)`

Renvoie 99.

`PRINT INT(-32,12)`

Renvoie -33, c'est bien la valeur inférieur du nombre passé en argument.

## ***Adresse mémoire de l'instruction***

`&H`

## ***Mots clés associés***

[DEF FN](#)

# **KEY**

## ***Objet***

Les touches F1, F2, F3 et F4 sont reprogrammables en mode normal et en mode SHIFT. Ce qui donne 8 touches au total.

Une touche peut contenir 8 caractères exécutables au maximum.

## **Syntaxe**

```
KEYn, char
```

## ***Exemple***

```
KEY1, "CONSOLE"
```

La touche F1 écrira « CONSOLE ».

```
KEY7, "CTON"+CHR$(13)
```

La touche F3 (avec SHIFT) mettra le moteur du magnétophone en marche.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

# **LCOPY**

## ***Objet***

Envoi de l'information contenue sur l'écran vers l'imprimante. Aussi appelée « HARD COPY ». Ne marche que sur imprimante graphique.

NDR : fonction à tester dans un programme.

## **Syntaxe**

**LCOPY**

## ***Exemple***

### ***Adresse mémoire de l'instruction***

&H

## **Mots clés associés**

[LPRINT](#), [PRINT#](#)

# **LEFT\$**

## ***Objet***

Sélectionne le nombre spécifié de caractère à gauche de la chaîne fourni en argument.

## ***Syntaxe***

**LEFT\$ (A\$, x)**

A\$: Chaîne de caractère sur laquelle on veut faire l'extraction.

x : nombre de caractère à extraire.

Si x vaut 0, renvoie une chaîne vide.

La valeur de x ne doit pas dépasser 255.

Si la chaîne est plus courte que le x spécifié, LEFT\$ renvoie toute la chaîne.

## ***Exemple***

```
10 T$="HELLO WORLD"
20 PRINT LEFT$(T$,5)
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

ASC, CHR\$, MID\$, RIGHT\$, STR\$, VAL

# **LEN**

## ***Objet***

Renvoie la longueur de la chaîne fournie en argument.

## **Syntaxe**

**LEN (A\$)**

A\$ : Chaîne dont on veut connaître la longueur

## ***Exemple***

```
10 D$ = "LES QUATRE SAISONS"  
20 PRINT LEN(D$)
```

Affichera 18.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

# **LET**

## ***Objet***

La valeur de "expression" est substituée et devient une variable.

En fait, le signe égal (=) suffit pour constituer l'affectation de la variable.

L'instruction LET a le même but.

**NDR : Vous devez considérer cette instruction comme obsolète.**

Dans le cadre d'une adaptation de programme, supprimer l'instruction LET pour gagner de la mémoire.

## ***Syntaxe***

```
LET NomVariable = Expression
```

## ***Exemple***

```
10 LET A=B+1
```

```
10 A=B+1
```

Les 2 lignes sont identiques sur le PHC-25.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

# **LINE**

## ***Objet***

Relier les points par une ligne ou les insérer dans un rectangle.

Deux points désignés par des références données sont reliés par une ligne qui a une couleur désignée.

Lorsque "B" est désigné, un rectangle est dessiné utilisant les deux points comme angles opposés.  
De plus, si F est désigné, le rectangle est coloré par la couleur spécifiée ou celle de la dernière instruction [COLOR](#).

Les définitions en ligne et en colonne sont assujetties à la définition du [SCREEN](#) (1,2, 3 ,4).

## ***Syntaxe***

```
LINE (x1,y1) - (x2,y2) , (couleur) , (BF)
```

X1 : coordonnées sur l'axe horizontale

Y1 : coordonnées sur l'axe verticale

X2 : coordonnées sur l'axe horizontale

Y2 : coordonnées sur l'axe verticale

Couleur : couleur selon le mode écran

B : Bordure, il faut utiliser la lettre « B »

F : Fill (remplissage), il faut utiliser la lettre « F »

## ***Exemple***

```
LINE (5,5) - (20,20) ,3
```

Les points de coordonnées 5,5 et 20,20 sont reliés par une ligne de la couleur 3

```
LINE (5,5) - (20,20) ,3,B
```

Les points 5,5 et 20 ,20 appartiennent à un rectangle. Ils sont les deux points opposés.

**LINE (10,8)-(190,130),2,BF**

Un rectangle colorié de couleur 2 passe par les points de coordonnées 10-8 et 190-130

Pour colorier l'intérieur d'une autre couleur, utiliser l'instruction [PAINT](#).

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[COLOR](#), [PAINT](#), [PRESET](#), [PSET](#), [SCREEN](#)

# **LIST**

## ***Objet***

Cette commande affiche sur l'écran le programme en mémoire.

## **Syntaxe**

**LIST [n-m]**

Si n n'est pas spécifié, affiche tout le programme.

Si n est spécifié, affiche la ligne n.

Si n-m sont spécifiés, affiche les lignes n à m.

Si -m est spécifié, affiche du début du programme jusqu'à la ligne m.

## ***Exemple***

**LIST**

Affiche tout le programme.

**LIST 1000**

Affiche la ligne 1000.

**LIST 100-150**

Affiche toutes les lignes de 100 à 150.

**LIST 100-**

Affiche toutes les lignes depuis la ligne 100 jusqu'à la fin du programme.

NDR : à tester à l'intérieur d'un programme.

## ***Adresse mémoire de l'instruction***

&H

## **Mots clés associés**

LLIST

# **LLIST**

## ***Objet***

Lister sur l'imprimante tout ou partie du programme existant en mémoire.  
Assume le fait que l'imprimante fait 80 caractères de large.

## **Syntaxe**

**LLIST [n-m]**

Si n n'est pas spécifié, affiche tout le programme.  
Si n est spécifié, affiche la ligne n.  
Si n-m sont spécifiés, affiche les ligne n à m.  
Si -m est spécifié, affiche du début du programme jusqu'à la ligne m.

## ***Exemple***

**LLIST**

Imprime tout le programme.

**LLIST 1000**

Imprime la ligne 1000.

**LLIST 100-150**

Imprime toutes les lignes de 100 à 150.

**LLIST 100-**

Imprime toutes les lignes depuis la ligne 100 jusqu'à la fin du programme.

NDR : à tester à l'intérieur d'un programme.

## ***Adresse mémoire de l'instruction***

&H

## **Mots clés associés**

[LIST](#)

# **LOCATE**

## ***Objet***

Le curseur est déplacé vers une position sur l'écran qui est exprimée par une position horizontale et verticale.

Le coin gauche en haut de l'écran est 0,0.

La position horizontale (colonne) va de 0 à 31 pour les modes 1,2,4 d'écran.

Toutefois, lorsque l'écran 3 est utilisé, la position horizontale s'arrête à 15.

La position verticale (ligne) va de 0 à 15 pour tous les modes d'écran.

LOCATE n'est pas affecté par [CONSOLE](#).

Cependant l'inverse est vrai. Si le LOCATE est hors champ de [CONSOLE](#) alors la prochaine instruction d'affichage se produira à l'endroit du LOCATE (selon si il y a un « ; » ou non).

## **Syntaxe**

```
LOCATE X,Y
```

X : Colonne, [0,31] ou [0,16] en SCREEN 3.

Y : Ligne, [0,16].

## ***Exemple***

```
10 FOR I=1 TO 100
20 LOCATE 10,10
30 PRINT I
40 NEXT
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[CLS](#), [CONSOLE](#)

# **LOG**

## ***Objet***

Renvoie le logarithme népérien de l'argument. Cet argument doit être positif.

## ***Syntaxe***

**LOG (x)**

X doit être positif.

## ***Exemple***

**PRINT LOG(5.4)**

Affichera 1.68639895.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[DEF FN](#)

# **LPOS**

## ***Objet***

Renvoie la position de tête du pointeur de ligne dans le tampon de sortie de l'imprimante.

## **Syntaxe**

**LPOS (X)**

## ***Exemple***

```
10 LPRINT "DIMANCHE"  
20 PRINT LPOS (X)
```

Aura pour résultat 8.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[CSRLIN](#), [POS](#)

# **LPRINT**

## ***Objet***

Écriture de données sur l'imprimante.

Cette instruction est identique à PRINT, la sortie se fait sur l'imprimante connectée au PHC 25.

## **Syntaxe**

```
LPRINT val
```

Val est une valeur soit alphanumérique soit numérique.

## ***Exemple***

```
LPRINT "Bonjour. "
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[LCOPY](#), [LLIST](#), [LPOS](#), [LPRINT](#), [PRINT#](#)

# **MID\$**

## ***Objet***

Renvoie une sous chaîne de la longueur spécifiée depuis un rang spécifié.

## ***Syntaxe***

**MID\$ (A\$, R, L)**

R : Rang de début d'extraction. [1,254]

L : Nombre de caractère à extraire. [0,255]

Si L vaut 0, renvoie tous les caractères à partir de R.

NDR zone de R à tester.

## ***Exemple***

```
10 A$= "BONJOUR"  
20 PRINT MID$(A$,2,4)
```

Affichera ONJO.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[ASC](#), [CHR\\$](#), [DEF FN](#), [LEFT\\$](#), [RIGHT\\$](#), [STR\\$](#), [VAL](#)

# **NEW**

## ***Objet***

Effacer le programme présent en mémoire et remet à zéro toutes les variables.

## **Syntaxe**

**NEW**

## ***Exemple***

**NEW**

## ***Adresse mémoire de l'instruction***

**&H**

## ***Mots clés associés***

[CONT](#), [END](#), [STOP](#)

## **NEXT**

***Objet***

**Syntaxe**

***Exemple***

***Adresse mémoire de l'instruction***

&H

**Mots clés associés**

[FOR](#), [STEP](#)

# **NOT**

## ***Objet***

Cette instruction n'est pas documentée.

Provoque un NOT logique avec la variable fournie.

Table :

A	R
0	1
1	0

## **Syntaxe**

`NOT a`

a doit être une valeur numérique.

## **Exemple**

`A = NOT B`

## **Adresse mémoire de l'instruction**

&H

## **Mots clés associés**

[AND](#), [IF](#), [OR](#)

# **ON GOSUB**

## ***Objet***

Branchement multiple suivant la valeur de l'expression. C'est cette valeur qui détermine sur quelle ligne le programme sera branché.

Si la valeur est I, le programme sera branché sur I.

Si l'expression n'est pas intégrale (entière) les décimales seront arrondies.

Si la valeur de l'expression est zéro ou négative ou si elle est plus grande que les numéros de ligne, le programme passera à la ligne suivante.

## ***Syntaxe***

ON Valeur GOSUB L1,L2,Ln...

Valeur est la valeur sur laquelle doit s'effectué le test.

En fonction du résultat, branchement sur la ligne n.

## ***Exemple***

```
ON K GOSUB 100,200,300,400
```

Si K<=0, ligne suivante

Si K>=5, ligne suivante

Si K=1, ligne 100

Si K=2, ligne 200

Si K=3, ligne 300

Si K=5, ligne 400

Après le RETURN, le programme enchaîne avec ligne qui suit le ON GOSUB.

## ***Adresse mémoire de l'instruction***

&H

## **Mots clés associés**

GOSUB, GOTO, RETURN

# **ON GOTO**

## ***Objet***

Branchement multiple suivant la valeur de l'expression. C'est cette valeur qui détermine sur quelle ligne le programme sera branché.

Si la valeur est I, le programme sera branché sur I.

Si l'expression n'est pas intégrale (entière) les décimales seront arrondies.

Si la valeur de l'expression est zéro ou négative ou si elle est plus grande que les numéros de ligne, le programme passera à la ligne suivante.

## **Syntaxe**

ON Valeur GOTO L1,L2,Ln...

Valeur est la valeur sur laquelle doit s'effectué le test.

En fonction du résultat, branchement sur la ligne n.

## ***Exemple***

ON K GOTO 100,200,300,400

Si K<=0, ligne suivante

Si K>=5, ligne suivante

Si K=1, ligne 100

Si K=2, ligne 200

Si K=3, ligne 300

Si K=5, ligne 400

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[GOSUB](#), [GOTO](#), [RETURN](#)

# **OR**

## ***Objet***

Cette instruction n'est pas documentée.

Provoque un OU logique avec les 2 variables fournies.

<b>A</b>	<b>B</b>	<b>R</b>
0	0	0
0	1	1
1	0	1
1	1	1

## ***Syntaxe***

**a OR b**

a et b doivent être des valeurs numériques.

## ***Exemple***

```
TO DO  
A = &H82 OR B
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[AND](#), [IF](#), [NOT](#)

# **OUT**

## ***Objet***

Cette instruction permet d'envoyer sur le port I l'octet J.

NDR : nécessite plus d'investigation pour connaître tous les INP et OUT du PHC-25.

Voir chapitre [INP et OUT](#).

## ***Syntaxe***

`OUT I,J`

I : No de port,

J : Octet à envoyer.

## ***Exemple***

`OUT &h20,&h64`

La valeur &h64 sera envoyée sur le port no &h20.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[INP](#)

# **PAINT**

## ***Objet***

Les coordonnées x et y définissent 1 point se situant dans une zone de l'écran. Cette zone est peinte avec la couleur C sauf le périmètre de couleur P.

## **Syntaxe**

```
PAINT (X,Y),C,P
```

X : coordonnée horizontale.

Y : coordonnée verticale.

c : couleur de remplissage selon le SCREEN.

p : couleur de périmètre selon le SCREEN.

Si p est omis ou si p est égale à c alors le périmètre aura la couleur définie par c.

## ***Exemple***

```
10 LINE(0,0)-(255,0),2,BF  
20 LINE(255,0)-(255,191),2,BF  
30 LINE (255,191)-(5,186),2,BF  
40 LINE(5,186)-(0,0),2,BF  
50 PAINT (80,80),4,2
```

NDR : exemple à revoir car nécessite SCREEN.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

COLOR, LINE, PRESET, PSET, SCREEN

# **PEEK**

## ***Objet***

Renvoie la valeur de l'octet de l'adresse spécifiée. Les valeurs retournées vont de &H00 à &HFF.

## **Syntaxe**

PEEK(x)

X : adresse mémoire dont on veut lire l'octet.

## ***Exemple***

```
A=PEEK(&H6000)
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[EXEC](#), INP, OUT, [POKE](#), [USR](#)

# PLAY

## **Objet**

La musique est produite par le générateur de son (vendu en option).

## **Syntaxe**

PLAY"ox [sx] [mx] [vx] [lx] [tx] [rx] [x+] [x-] n"

Les fonctions entre crochet sont facultatives

Avec n: (cdefgab), notation Anglo-saxonne de la musique.

c	d	e	f	g	a	b
do	ré	mi	fa	sol	la	si

Dans l'instruction PLAY, pour la chaîne de caractères, les caractères suivants sont utilisés pour donner des significations spéciales :

ox	Désigne l'octave (l'octave plus haute est o4) initial 4
sx	Désigne la forme de l'enveloppe (se référer à l'annexe D.)
mx	Désigne la période de l'enveloppe (1<=x<=65535)
vx	Désigne le volume (0<=x<=15) initial 8
lx	Désigne la durée du son (0<=x<=64)
tx	Désigne la vitesse (tempo) du son initial 170
rx	Désigne la durée de la période de silence (aucun son) (1<=x<=64)
x+	Le son est remonté d'un demi-ton
x-	Le son est descendu d'un demi-ton

Tous les caractères de PLAY doivent être en minuscule.

## **Exemple**

PLAY"o4 18 v2 s5 a"

octave 4 ; longueur 8 ; volume 2 ; enveloppe 5 voix ; note LA

```
10 FOR I=0 TO 10
20 PLAY"o6132c"
30 NEXT I

10 FOR I=0 TO 10
20 PLAY"o41cdfgab05c"
30 NEXT I

10 PLAY
"o7fardl34fafdbcfa05farecferafbcl56gbfddeaedaeaddcffdfbfebacdfeadaeo4129ggabgag
afdfdfaffaddeeaddbccffcfdo7ffacca"
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

SOUND

# **POINT**

## ***Objet***

Renvoie le numéro de la couleur sélectionnée par les coordonnées (x,y).

## **Syntaxe**

**POINT (x,y)**

x,y coordonnée du point dont on veut la couleur.

## ***Exemple***

```
10 PSET(7,30),3  
20 PRINT POINT(7,30)
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[PRESET](#), [PSET](#)

# **POKE**

## ***Objet***

Insert un octet à une adresse désignée de la mémoire.

## **Syntaxe**

```
POKE Expression 1, Expression 2
```

Expression 1 : Adresse mémoire

Expression 2 : Octet à placer (&H00 à &HFF)

Les valeurs peuvent être sous forme hexadécimale ou décimale.

## ***Exemple***

```
10 POKE &H6010,&H81  
20 POKE 4053,24
```

Ligne 10 : Affiche le caractère pour les mode SCREEN 1 et SCREEN 2.

Ligne 20 : place à l'adresse &h0FD5 l'octet de valeur &H81.

```
POKE &h6080,42
```

Affiche le caractère « \* » première colonne 5<sup>ème</sup> ligne pour les modes SCREEN 1 et SCREEN 2.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[DATA](#), [EXEC](#), [INP](#), [OUT](#), [PEEK](#), [READ](#), [RESTORE](#), [USR](#)

# **POS**

## ***Objet***

Renvoie la position du curseur sur la colonne.

## **Syntaxe**

`POS (X)`

## ***Exemple***

```
10 LOCATE 12,7  
20 PRINT POS(X)
```

Afficher 12.

## ***Adresse mémoire de l'instruction***

`&H`

## ***Mots clés associés***

[CSRLIN](#), [LPOS](#)

# **PRESET**

## ***Objet***

Effacer le point de coordonnées (X,Y) sur l'écran.

Forte dépendance vis-à-vis de l'instruction [SCREEN](#).

## **Syntaxe**

**PRESET (X,Y)**

X : coordonnée horizontale.

Y : coordonnée vertical.

## ***Exemple***

```
5 SCREEN 4,1,1 TO 20
6 CLS
9 FOR X=1 TO 100
10 PSET (X,100),3
11 IF X> 20 THEN PRESET (X,100)
12 IF X> 50 THEN PSET(X,100),3
15 NEXT
20 GOTO 20
```

## ***Adresse mémoire de l'instruction***

&H

## **Mots clés associés**

[COLOR](#), [LINE](#), [PAINT](#), [PRESET](#), [PSET](#), [SCREEN](#)

# **PRINT**

## ***Objet***

Écrire des données sur l'écran.

Si la liste d'expressions est absente, une ligne blanche s'en suivra.

La position de chaque élément de la ligne est déterminée par la ponctuation.

## **Syntaxe**

```
PRINT élément[,|;[élément]]...
```

Les caractères doivent obligatoirement être encadrés entre guillemets.

Lorsque l'on écrit un nombre multiple d'expressions, chacune doit être séparée par une virgule (,) ou un point-virgule (;).

Une virgule entre deux expressions de la liste fait que l'expression suivant la virgule est imprimée au début de la zone suivante.

Si c'est un point-virgule, l'expression se fait immédiatement après la dernière expression écrite.

Un ou plusieurs espaces entre deux expressions ont le même effet que le point-virgule.

## ***Exemple***

```
10 A$="PROGRAMME"
20 B$="No.":C=15
30 D$="exemple de programme"
40 PRINT D$
50 PRINT
60 PRINT A$;B$;C
70 PRINT "suivant"
```

```
PRINT"Hello World !
```

Cette instruction fonctionne, malgré l'absence du guillemet en fin de ligne.

Va afficher tous les caractères présents jusqu'à la fin de la ligne.

## ***Adresse mémoire de l'instruction***

&H

## **Mots clés associés**

INPUT, INPUT #, LOCATE, LPRINT, PRINT#, SPC, TAB

# **PRINT#**

## ***Objet***

Écriture des données dans un appareil désigné.

La désignation de l'appareil se fait par le numéro suivant PRINT #

## **Syntaxe**

**PRINT#-n**

N=0 : écran

N=1 : magnétophone

N=3 : imprimante.

## ***Exemple***

### ***Adresse mémoire de l'instruction***

&H

## **Mots clés associés**

[INPUT](#), [INPUT #](#), [LOCATE](#), [LPRINT](#), [PRINT](#), [SPC](#), [TAB](#)

# **PSET**

## ***Objet***

Un point ayant une couleur désignée est affiché à l'endroit spécifié par les coordonnées X, Y.  
La couleur c dépend du [SCREEN](#) 1,2,3,4 et de la couleur choisie ([COLOR](#)).

## **Syntaxe**

**PSET (X, Y) , c**

X : coordonnée horizontale

Y : coordonnée verticale

C : couleur

## ***Exemple***

### ***Adresse mémoire de l'instruction***

&H

## **Mots clés associés**

[COLOR](#), [LINE](#), [PRESET](#), [PSET](#), [SCREEN](#)

# **READ**

## ***Objet***

Lecture des valeurs dans une instruction DATA et affectation aux variables citées. Une instruction READ doit toujours être utilisée en relation avec une ou plusieurs instructions DATA.

L'instruction READ distribue les valeurs données dans l'instruction DATA aux variables mentionnées.

Ces variables sont numériques ou de type chaîne et doivent s'accorder, sinon il peut y avoir "erreur de syntaxe".

S'il reste des DATA inutilisés, le READ suivant les utilise.

S'il n'en reste pas, ils seront ignorés.

On peut relire les DATA grâce à l'instruction [RESTORE](#).

## **Syntaxe**

```
READ valeur[,valeur]
```

Valeur est de type numérique ou alphanumérique.

## ***Exemple***

```
READ A$
READ A
READ A,A$
```

## ***Adresse mémoire de l'instruction***

&H

## **Mots clés associés**

[DATA](#), [PEEK](#), [POKE](#), [RESTORE](#)

# **REM**

## ***Objet***

Insérer des remarques et des notes dans le programme.

Bien que l'instruction REM ne soit pas exécutable, elle figure dans la liste avec le contenu du texte qui l'accompagne.

Les instructions qui suivraient REM après un deux-points « : » ne sont pas exécutées.

## **Syntaxe**

```
REM [texte]
```

Texte est optionnel.

## ***Exemple***

```
10 REM Menu
```

```
10 REM Menu: PRINT
```

L'instruction PRINT ne sera pas exécutée.

## ***Adresse mémoire de l'instruction***

&H

## **Mots clés associés**

# **RESTORE**

## ***Objet***

Permettre de relire des données à partir d'une certaine ligne.

Après l'instruction RESTORE, le premier [READ](#) prend ses valeurs dans la première instruction [DATA](#).

Lorsque le numéro de ligne est spécifié, la lecture se fera à partir de cette ligne.

## **Syntaxe**

```
RESTORE [n]
```

N : numéro de ligne pour relire les données.

## ***Exemple***

```
RESTORE
```

Restaure la lecture au premier DATA du programme.

```
RESTORE 100
```

Restaure la lecture à la ligne 100, ou au premier DATA après la ligne 100.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[DATA](#), [PEEK](#), [POKE](#), [READ](#)

# **RETURN**

## ***Objet***

Les instructions qui suivraient RETURN après un deux-points « : » ne sont pas exécutées.

## **Syntaxe**

**RETURN**

## ***Exemple***

```
...  
1050 RETURN
```

```
2000 RETURN: PRINT"HELLO WORLD"
```

Dans l'exemple ci-dessus, le PRINT ne sera jamais exécuté.

## ***Adresse mémoire de l'instruction***

**&H**

## **Mots clés associés**

[GOSUB](#), [ON GOSUB](#)

# **RIGHT\$**

## ***Objet***

Sélectionne le nombre spécifié de caractère à droite de la chaîne fourni en argument.

## ***Syntaxe***

**RIGHT\$ (A\$, n)**

A\$: Chaîne de caractère sur laquelle on veut faire l'extraction.

n : nombre de caractère à extraire.

Si x vaut 0, renvoie une chaîne vide.

La valeur de x ne doit pas dépasser 255.

Si la chaîne est plus courte que le x spécifié, RIGHT\$ renvoie toute la chaîne.

## ***Exemple***

```
10 A$="ABCDEFG"  
20 PRINT RIGHT$(A$,4)
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

ASC, CHR\$, DEF FN, LEFT\$, MID\$, STR\$, VAL

# RND

## ***Objet***

Initialiser le générateur de nombres aléatoires.

Si l'expression facultative manque, l'exécution est suspendue.

Si l'on utilise RND sans initialiser le générateur à chaque exécution du programme, la même suite de nombres aléatoires sera produite.

## ***Syntaxe***

**RND (x)**

x>0 commence une nouvelle séquence

x=0 donne le dernier nombre généré

x<0 génère un nouveau nombre aléatoire

## ***Exemple***

```
10 FOR I=1 TO 5
20 PRINT INT(RND(1)*100)
30 NEXT I

10 CLS:PRINT"RND TEST"
20 A=1:B=13
30 I=RND(-TIME):PRINT"SEED:";I: REM NEW SEED
40 FOR I=1 TO 10
50 N=A+INT((B-A+1)*RND(1)): REM BETWEEN [A,B]
60 PRINT "E";I;" :";A;" -"; B; " ="; N
70 NEXT I
80 END
```

Crée une nouvelle série et génère un nombre entre A et B.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

DEF FN

# **RUN**

## ***Objet***

Lancement de l'exécution du programme stocké en mémoire.  
La touche F1 contient par défaut l'instruction RUN.

## **Syntaxe**

**RUN [n]**

N : numéro de ligne, n'est pas obligatoire.

## ***Exemple***

```
RUN  
RUN 1000
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[END](#), [STOP](#)

# **SCREEN**

## ***Objet***

Le PHC 25 a deux pages d'écran qui sont définies à la mise sous tension.

- mode avec 1 page d'écran, la mémoire disponible sera de 14265 octets.
- mode avec 2 pages d'écran, la mémoire disponible sera de 8121 octets.

Ce qui correspond à la question posée à l'allumage « SCREEN ? ».

Il ne faut pas confondre le choix du nombre de pages utilisées et la façon d'utiliser ces pages.

L'instruction SCREEN provoque l'équivalent d'un CLS.

Voir aussi l'instruction COLOR.

## **Syntaxe**

**SCREEN a,b,c**

La valeur de a va fixer le choix de la résolution d'écran.

Il y a 4 possibilités d'utilisation d'une page suivant le tableau ci-dessous.

a	1	2	3	4
<b>Texte</b>	16L 32C	16L 32C	16L 16C	16L 32C
<b>Graphique</b>	(16x32)*	64x48	128x192	256x192

(\*) Les instructions graphiques fonctionnent.

La valeur b désigne le numéro de la page affectée par les instructions qui suivent.

La valeur c désigne le numéro de la page affichée à l'écran. (NDR : selon le choix fait au démarrage de la machine).

Les variables b et c ne sont utilisables qu'avec 2 pages écran (ce qui correspond au 2 que l'on a choisi à la mise sous tension).

## ***Exemple***

Voir ci-après pour les 4 mode d'écran.

## ***Adresse mémoire de l'instruction***

&H

## **Mots clés associés**

CLS, COLOR, CONSOLE

# **SCREEN 1**

## ***Objet***

Mode caractères.

Voir information avec [SCREEN](#).

Voir aussi l'instruction [COLOR](#).

## **Syntaxe**

```
SCREEN 1,b,c
```

La valeur b désigne le numéro de la page affectée par les instructions qui suivent.

La valeur c désigne le numéro de la page affichée à l'écran. (NDR : selon le choix fait au démarrage de la machine).

Les valeurs b et c n'ont pas de signification si 1 a été choisi au démarrage de la machine.

## ***Exemple***

Mode 1 écran :

```
SCREEN 1
```

Il n'y a pas besoin de spécifier les valeur b et c.

Mode 2 écrans :

```
SCREEN 1,1,1
```

Écran de type texte pour l'écran 1 et il est affiché à l'écran.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[CLS](#), [COLOR](#), [CONSOLE](#)

# **SCREEN 2**

## ***Objet***

Mode caractère et semi-graphique.

Voir information avec [SCREEN](#).

Voir aussi l'instruction [COLOR](#).

## **Syntaxe**

```
SCREEN 2,b,c
```

La valeur b désigne le numéro de la page affectée par les instructions qui suivent.

La valeur c désigne le numéro de la page affichée à l'écran. (NDR : selon le choix fait au démarrage de la machine).

## ***Exemple***

Mode 1 écran :

```
SCREEN 2
```

Il n'y a pas besoin de spécifier les valeur b et c.

Mode 2 écrans :

```
SCREEN 2,1,1
```

Passe sur l'écran 1, en mode 2 et l'affiche.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[CLS](#), [COLOR](#), [CONSOLE](#)

# **SCREEN 3**

## ***Objet***

Mode graphique.

Voir information avec [SCREEN](#).

Voir aussi l'instruction [COLOR](#).

Comme il s'agit d'un mode graphique, l'instruction [PRINT](#) doit être considérée comme générant du graphisme. De façon implicite, [PRINT](#) fait des équivalents de [PSET](#), cela implique un affichage de type transparent, ou si vous préférez, une surimpression du caractère sur la zone. [PRINT " "](#) (un espace), n'effacera rien à l'écran, de même qu'un [PRINT SPC\(10\)](#).

## ***Syntaxe***

```
SCREEN 3,b,c
```

La valeur b désigne le numéro de la page affectée par les instructions qui suivent.

La valeur c désigne le numéro de la page affichée à l'écran. (NDR : selon le choix fait au démarrage de la machine).

## ***Exemple***

Mode 1 écran :

```
SCREEN 3
```

Il n'y a pas besoin de spécifier les valeur b et c.

Mode 2 écrans :

```
SCREEN 3,1,1
```

Passe sur l'écran 1, en mode 3 et l'affiche.

## ***Adresse mémoire de l'instruction***

&H

## **Mots clés associés**

CLS, COLOR, CONSOLE

# **SCREEN 4**

## ***Objet***

Mode graphique.

Voir information avec [SCREEN](#).

Voir aussi l'instruction [COLOR](#).

Comme il s'agit d'un mode graphique, l'instruction [PRINT](#) doit être considérée comme générant du graphisme. De façon implicite, [PRINT](#) fait des équivalents de [PSET](#), cela implique un affichage de type transparent, ou si vous préférez, une surimpression du caractère sur la zone. [PRINT " "](#) (un espace), n'effacera rien à l'écran, de même qu'un [PRINT SPC\(10\)](#).

## ***Syntaxe***

```
SCREEN 4,b,c
```

La valeur b désigne le numéro de la page affectée par les instructions qui suivent.

Selon le cas, la plage d'adresses n'est pas la même.

La valeur c désigne le numéro de la page affichée à l'écran. (NDR : selon le choix fait au démarrage de la machine).

## ***Exemple***

Mode 1 écran :

```
SCREEN 4
```

Il n'y a pas besoin de spécifier les valeur b et c.

Mode 2 écrans :

```
SCREEN 4,1,1
```

## ***Adresse mémoire de l'instruction***

&H

## **Mots clés associés**

CLS, COLOR, CONSOLE

# **SCRIN**

## ***Objet***

Renvoie la valeur ASCII du caractère de coordonnées d'écran (c,l).

## **Syntaxe**

**SCRIN(c,l)**

c : numéro de la colonne

L : numéro de la ligne

## ***Exemple***

```
5 CLS
10 PRINT"ABCDEF"
20 Z=SCRIN(5,0)
30 PRINT Z
```

Affichera 70.

La valeur renvoyée est bien 70 qui est le code ASCII de F en décimal.

Les coordonnées texte commence en haut à gauche (0,0).

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[CHR\\$](#), [LOCATE](#)

# **SGN**

## ***Objet***

Connaître le signe d'un nombre.

## **Syntaxe**

**SGN (x)**

x : Nombre dont on veut connaître le signe.

Renvoie 1 si  $x \geq 0$

Renvoie -1 si  $x < 0$

## ***Exemple***

```
PRINT SGN(912)
```

Affichera 1.

SGN peut être utiliser en mode calcul pour inverser une valeur.

TO DO : Exemple.

## ***Adresse mémoire de l'instruction***

&H

## **Mots clés associés**

[DEF FN](#)

# **SIN**

## ***Objet***

Calcule le sinus de l'angle en radians.  
Le résultat est obtenu en précision simple.

## **Syntaxe**

**SIN(x)**

x : angle dont on veut le sinus.

## ***Exemple***

```
PRINT SIN(0.32)
```

Résultat : .314566561

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[COS](#), [DEF FN](#), [SIN](#), [TAN](#)

# **SLOAD**

## ***Objet***

L'information de l'écran enregistrée sur la cassette est chargée dans la mémoire de l'écran.  
Si la page de l'écran utilisée n'est pas celle qui a été mémorisée, la commande SLOAD est arrêtée et une ERREUR est affichée.

NDR : Technique à étudier pour éventuellement charger un programme binaire en mémoire.  
Donc usage de 2 écrans.

## ***Syntaxe***

```
SLOAD "nom"
```

Le nom est sensible à la casse.  
8 caractères maximum.

## ***Exemple***

```
SLOAD "data"
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[CLOAD](#), [CLOAD?](#), [SSAVE](#)

# **SOUND**

## ***Objet***

Une sortie de son est produite directement en envoyant les commandes au générateur de son. La sortie du son désiré peut être faite en désignant l'information (DATA) qui est dans le registre et les registres de 0 à 13 listés ci-dessous.

## **Syntaxe**

```
SOUND registre,data
```

Registre :

Data :

## ***Exemple***

### ***Adresse mémoire de l'instruction***

&H

### **Mots clés associés**

[PLAY](#)

# **SPC**

## ***Objet***

Écrit le nombre spécifié d'espace.

Ne fonctionne que pour les modes écran 1 et 2. En mode écran 3 et 4, n'a aucun effet, voir SCREEN.

Ne s'applique qu'avec PRINT et LPRINT.

## **Syntaxe**

**SPC (x)**

x : nombre d'espace. [0-255].

## ***Exemple***

```
10 PRINT"GAME";
20 PRINT SPC(3);
30 PRINT"OVER"
```

Place 3 espaces entre les 2 mots.

Noter le point-virgule en fin des lignes 10 et 20 pour continuer l'affichage sur la même ligne.

Les 3 lignes peuvent se fusionner.

```
10 PRINT"GAME";SPC(3);"OVER"
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[LOCATE](#), [LPRINT](#), [PRINT](#), [PRINT#](#), [TAB](#)

# **SQR**

## ***Objet***

Renvoie la racine carrée de x.

## ***Syntaxe***

**SQR(x)**

x doit être zéro ou positif.

## ***Exemple***

```
PRINT SQR(10)
```

Résultat : 3.16227766.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[DEF FN](#)

# **SSAVE**

## ***Objet***

L'information affichée sur l'écran est sauvegardée sur la cassette du magnétophone.

SSAVE peut servir de commande ou d'instruction.

Avant l'exécution d'un SAVE, s'assurer que le magnétophone est correctement branché et que les touches RECORD et PLAY sont bien enfoncées.

Le chargement de la sauvegarde ne pourra se faire que sur le même écran. Si vous sauvegarder l'écran 2, vous ne pourrez charger que sur l'écran 2 (choix au démarrage du PHC-25).

## **Syntaxe**

```
SSAVE "fichier"
```

Fichier : nom du fichier qui contiendra les informations de l'écran. Le nom est sensible à la casse.

## ***Exemple***

```
SSAVE "data"
```

C'est la méthode qui a été utiliser pour faire le premier dump de la ROM Basic.

TO DO : le programme en question.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[CLOAD](#), [CLOAD?](#), [CSAVE](#), [SLOAD](#)

# **STEP**

## ***Objet***

STEP sert à définir l'incrément (ou le décrément) appliqué à la variable de boucle à chaque itération.

## **Syntaxe**

```
1000 FOR variable = début TO fin STEP pas
      ' instructions
1100 NEXT
```

## ***Exemple***

```
10 FOR I=1 TO 10 STEP 2
20 NEXT I
```

Compte de 1 à 10 par pas de 2.

## ***Adresse mémoire de l'instruction***

&H

## **Mots clés associés**

[FOR](#), [NEXT](#)

# **STICK**

## ***Objet***

Permet de tester les manettes et le clavier (touches de direction)  
Nécessite d'avoir le synthétiseur pour les joysticks.

## **Syntaxe**

**STICK(x)**

x=0 : Touches du clavier

x=1 : Joystick 1

x=2 : Joystick 2.

## ***Exemple***

```
10 IF STICK(1)=1 then print "nord"
20 IF STICK(1)=2 then print "nord-est"
30 IF STICK(1)=3 then print "est"
40 IF STICK(1)=4 then print "sud-est"
50 IF STICK(1)=5 then print "sud"
60 IF STICK(1)=6 then print "sud-ouest"
70 IF STICK(1)=7 then print "ouest"
80 IF STICK(1)=8 then print "nord-ouest"
90 IF STICK(1)=0 then print "manette inactive"
```

8 directions sont détectables via le Joystick 1.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[STRIG](#)

# **STOP**

## ***Objet***

Arrête le programme en exécution.

Le STOP peut être mis à n'importe quel endroit du programme pour arrêter son exécution. Toutefois, contrairement à l'instruction [END](#), lorsque STOP est exécuté, le message suivant apparaît :

BREAK in nnnnn

Nnnnn étant le numéro de ligne où se situe le STOP.

La reprise du programme peut se faire via [CONT](#) si et seulement si aucune modification n'est faite.

## ***Syntaxe***

STOP

## ***Exemple***

9990 STOP

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[END](#), [CONT](#)

# **STRIG**

## ***Objet***

Test du bouton de Joystick ou barre espace.

## ***Syntaxe***

**STRIG (x)**

x=0 : Barre espace.

x=1 : Bouton Joystick 1.

x=2 : Bouton Joystick 2.

Renvoie 1 si l'appuie est détecté, sinon 0.

## ***Exemple***

```
10 IF STRIG(0)=0 THEN GOTO 10
```

Attend l'appuie sur la barre d'espace.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[STICK](#)

# **STR\$**

## ***Objet***

Transforme un nombre en chaîne de caractères.

## **Syntaxe**

**STR\$ (x)**

## ***Exemple***

```
10 X=10
20 PRINT STR$(X)
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

ASC, CHR\$, DEF FN, LEFT\$, MID\$, RIGHT\$, VAL

# **TAB**

## ***Objet***

Met des espaces jusqu'à la colonne spécifiée.

## **Syntaxe**

**TAB (x)**

x : numéro de colonne.

## ***Exemple***

```
10 PRINT"XXX";TAB(10);"XXX"  
20 PRINT"PHC-25";TAB(10);"PROGRAMME"
```

Affichera la 2<sup>ème</sup> partie du PRINT à partir de la colonne 10.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

LPRINT, PRINT, PRINT#, [SPC](#)

# **TAN**

## ***Objet***

Renvoie en simple précision la tangente de l'angle, exprimée en radians

## **Syntaxe**

**TAN (x)**

x : angle dont on veut la tangente.

## ***Exemple***

```
PRINT TAN(1.34)
```

Renvoie : 4.67344123.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

COS, [DEF FN](#), SIN

# TIME

## ***Objet***

Compteur de temps qui ajoute 1 chaque 1/360 secondes.  
Le compteur à une précision faible.

## **Syntaxe**

TIME

## ***Exemple***

```
10 A=TIME  
20 B=TIME-A  
30 PRINT INT(B/360)  
40 GOTO 20
```

En conjonction avec [RND](#), fait un RANDOMIZE (fonction qui n'existe pas).

I=RND (-TIME)

La syntaxe « I= » est obligatoire.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[RND](#)

# **THEN**

## ***Objet***

Suivant les résultats d'un test, l'on prendra une décision ou l'on choisira une suite pour l'exécution du programme.

Si le résultat du test est vrai, c'est-à-dire différent de zéro, alors l'instruction THEN suivra jusqu'à ELSE ou la fin de la ligne et sera exécutée.

Si l'expression est égale à zéro (valeur fausse), seules les instructions suivant ELSE, si elles existent, seront exécutées.

## ***Syntaxe***

```
IF <expression> THEN <instruction> ELSE <instruction>
```

## ***Exemple***

```
5 A=-5: B=-10
10 IF A<>0 THEN B=10 ELSE B=0: GOTO 100
30 PRINT A,B
40 END
100 PRINT A,B
110 A=A+1
120 GOTO 10
```

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

ELSE, IF

**TO**

***Objet***

**Syntaxe**

FOR ... TO ... STEP ...

***Exemple***

***Adresse mémoire de l'instruction***

&H

**Mots clés associés**

FOR, STEP

# USR

## *Objet*

Exécute un programme fait par l'utilisateur en langage machine (Z80).  
Le programme utilisateur renvoie une valeur (la valeur d'un registre).

## *Syntaxe*

X=USR (A)

X valeur de retour

A adresse ?

Il est nécessaire de pousser les tests avec une routine assembleur.  
Avec MAME ?

## *Exemple*

```
20 FOR I=&HE800 TO &HE802
30 READ D
40 POKE I,D
50 NEXT I
60 DATA 62,255,201
70 X=USR(&HE800)
80 PRINT X
```

```
LD A,$FF
RET
```

La ligne 70 provoque un « Illegal function call in 70 ». Ce qui permet d'être sur la voie, contrairement à un « Syntax error ».

NDR : Le nombre utilisé est-il en dehors des paramètres autorisés ?  
&HE800 pas bon.

## *Adresse mémoire de l'instruction*

&H

## ***Mots clés associés***

DATA, EXEC, PEEK, POKE, READ, RESTORE

# **VAL**

## ***Objet***

Renvoie la valeur de la chaîne de caractères fournie en argument.

Si le premier caractère de la chaîne n'est pas +, -, &, ou un chiffre, la valeur renvoyée est 0.

En notation hexadécimale [0-9],[A-F], les autres caractères sont ignorés.

## ***Syntaxe***

**VAL (x\$)**

x\$ : Chaîne à convertir

## ***Exemple***

```
10 FOR I=1 TO 3
20 READ A$
30 PRINT VAL("&H"+A$);
40 NEXT
50 DATA C3,40,FF
```

Lit les valeurs dans la ligne DATA et les convertis depuis le format hexadécimal vers le format décimal.

## ***Adresse mémoire de l'instruction***

&H

## ***Mots clés associés***

[ASC](#), [CHR\\$](#), [LEFT\\$](#), [MID\\$](#), [RIGHT\\$](#), [STR\\$](#)



# Aide à la programmation



# Mémoire du PHC-25

## Plages mémoires

### Plage générale

Zone	Adresse	
Travail du Basic	&hFFFF	
	&hF800	
	&hF7FF	
Page vidéo 2 (6ko)	&hE9FF	Screen 1,2 Attribut
	&hE800	
	&hE1FF	Screen 1,2 Data
	&hE000	
Programme Basic (8ko)	&hDFFF	
	&hC000	
Espace libre (?) (16Ko)	&hBFFF	
	&h8000	
Espace libre (?) (2ko)	&h7FFF	
	&h7800	
	&h77FF	
Page vidéo 1 (6Ko)	&h69FF	Screen 1,2 Attribut
	&h6800	
	&h61FF	Screen 1,2 Data
	&h6000	
Interpréteur BASIC (ROM 24Ko)	&h5FFF	
	&h0000	

Programme Basic correspond à l'emplacement de votre programme, soit environ 8Ko (8121 octet). Associé à la page vidéo 2, cela donne environ 14Ko (14265 octets).

14336-14265 = 71 octets. 8192-8121 = 71 Octets.

L'adresse &hC001 est le début du programme basic.

En &hC000, la valeur est &h00.

Les 70 autres octets étant réservés pour un usage ?

## **Plage Travail du BASIC**

Zone	Adresse	
Travail du Basic	&hFFFF	
	&hFB58	Mode 1,2 écrans
	&hFB56	Mode 1,2 écrans
	&hF800	

Les adresses **&hFB56** et **&hFB58** contiennent les valeurs qui déterminent si le PHC-25 utilise 1 ou 2 écrans (choix au démarrage). Voir [Forcer 1 ou 2 écrans](#).

Le reste de la plage est à décoder.

## **Table des adresses mémoire des instructions BASIC**

Toutes ces adresses sont à vérifier.

Instruction	Adresse
ABS	
AND	
ASC	
CHR\$	
CLEAR	&H1981
CLOAD	&H40F2
CLOAD?	
CLS	&H3F70
COLOR	&H3CF6
CONSOLE	&H3EA9
CONT	&H193F
COS	
CSAVE	&H4442
CSRLIN	
CTOFF	&H4555
CTON	&H454E
DATA	&H2553
DEF	&H236E
DIM	&H213F
ELSE	&H05E1
END	&H1907
EXEC	&H1070
EXP	
FN	&H2E49
FOR	&H07BF
FRE	
GOSUB	&H08E0
GOTO	&H08F5
IF	&H0979
INKEY\$	&H2483
INP	
INPUT	&H0A1F
INPUT#	
INT	
KEY	&H0F3A
LCOPY	&H0FC6
LEFT\$	
LEN	
LET	&H24F9
LINE	&H3921

LIST	&H25DB
LLIST	&H25D6
LOCATE	&H3E73
LOG	
LPOS	
LPRINT	&H266C
MID\$	
NEW	&H1866
NEXT	&H1A4C
NOT	
ON	&H095C
OR	
OUT	&H24E2
PAINT	&H37A2
PEEK	
PLAY	&H4907
POINT	
POKE	&H24EE
POS	
PRESET	&H3B56
PRINT	&H2673
PRINT#	
PSET	&H3B36
READ	&H0A66
REM	&H254D
RESTORE	&H18D1
RETURN	&H0936
RIGHT\$	
RND	
RUN	&H08C2
SCREEN	&H3F1B
SCRIN	
SGN	
SIN	
SLOAD	&H4086
SOUND	&H4679
SPC	&H1AA6
SQR	
SSAVE	&H4344
STEP	
STICK	
STOP	&H18FD
STRIG	
STR\$	
TAB	&H2E43
TAN	

TIME	
THEN	
TO	&H2E35
USR	
VAL	

Exemple :

```
EXEC &H3F70
```

Déclenche un CLS. Mais pas plus rapide que l'instruction CLS elle-même.

## Pages Vidéo

Il y a 2 pages vidéo qui correspondent au choix fait lors du démarrage de la machine.

La page 1 est toujours utilisée.

La page 2 lorsqu'elle n'est pas utilisée, est utilisée comme « Travail du basic ».

Chaque page vidéo contient 6Ko.

Pour plus de détail sur ces pages consulter l'instruction [SCREEN](#) et le [chapitre consacré à l'écran](#).

Zone	Adresse	
Page vidéo 2 (6ko)	&hF7FF	
	&hE9FF	Screen 1,2
	&hE800	Attribut
	&hE1FF	Screen 1,2
Page vidéo 1 (6Ko)	&hE000	Data
	&h77FF	
	&h69FF	Screen 1,2
	&h6800	Attribut
	&h61FF	Screen 1,2
	&h6000	Data

Travail d'analyse plus poussé à faire.

NDR : Décodage de la RAM et de la ROM.

Voir aussi [CLEAR](#).

# L'écran du SANYO PCH-25

## ***Les 2 choix au boot***

Lors du démarrage de la machine, apparait :

**SCREEN ?**

La réponse est soit 1, soit 2.

À ne pas confondre avec l'instruction [SCREEN](#).

Réponses :

1 – Un seul écran de travail

2 – 2 écrans de travail

La bascule entre les 2 écrans se fera avec **CTRL+Q**.

En programmation via l'instruction [SCREEN](#).

## **Les résolutions d'écran**

Le tableau ci-dessous donne les possibilités du PHC-25.

	1	2	3	4
Texte	16L 32C	16L 32C	16L 16C	16L 32C
Graphique	(32x16)	64x48	128x192	256x192

- Les coordonnées texte commencent en haut à gauche en 0,0.
- Les coordonnées graphiques commencent en haut à gauche en 0,0. Ce qui peut être délicat à gérer lorsqu'il faut tracer des courbes de fonction. Voir chapitre sur les [Formules Graphiques](#).
- La première position est l'axe des abscisses.
- La deuxième position est l'axe des ordonnées.
- Toujours se baser sur la résolution 256x192 quel que soit le mode pour tracer.

Exemple :

```
10 LINE(0,0)-(255,0)
```

Trace une ligne en haut de l'écran.

Selon le [SCREEN](#) choisi, la dimension de la ligne changera.

RAPPEL : Pour les instructions graphiques, il faut toujours se référer à la dimension maximum, soit 256x192 pixels.

## ***Caractères non affichables par CHR\$***

Les caractères spéciaux de **&h00** à **&h1C** (voir table page 75 de la documentation) ne sont pas affichable par **PRINT CHR\$(n)**.

Il faut donc passer par la solution du **POKE**.

Il est possible de les utiliser via la touche « GRAPH », mais ceci n'est pas du tout recommandé.

NDR : compte tenu de la pauvreté du nombre de caractères de la version PAL, une ROM alternative existe.

## Le screen 1

Le screen 1 est de 32 colonnes sur 16 lignes.

Il s'agit d'un mode texte uniquement. Son adresse mémoire va de &h6000 à &h61FF.

Soit 512 octets, ce qui correspond bien à 32x16 caractères.

La numérotation par du coin en haut à gauche de l'écran, de 0 à 31 et 0 à 15.

## Plages d'adresses vidéo

Table d'adressage (1.1) simplifiée (Data) :

Ligne	Colonne 0	Colonne 31
0	&h6000	&h601F
1	&h6020	&h603F
2	&h6040	&h605F
3	&h6060	&h607F
4	&h6080	&h609F
5	&h60A0	&h60BF
6	&h60C0	&h60DF
7	&h60E0	&h60FF
8	&h6100	&h611F
9	&h6120	&h613F
10	&h6140	&h615F
11	&h6160	&h617F
12	&h6180	&h619F
13	&h61A0	&h61BF
14	&h61C0	&h61DF
15	&h61E0	&h61FF

Il est possible de simuler le LOCATE x,y directement sur l'écran avec un POKE directement dans l'adresse avec la valeur du caractère souhaité.

La valeur de l'octet mis à l'écran correspond au générateur de caractère (voir page 75 de la documentation) ou le chapitre « Jeu de caractères ».

Chaque partie de l'écran est donc représentée par un octet.

En principe cet octet est de type ASCII, mais réduit du fait de la simplification du générateur de caractère (version PAL).

Les couleurs sont dépendantes de la deuxième plage d'adresse (&h6800 - &h69FF), appelé attributs. L'octet de cette plage va influencer la couleur et si on passe sur du texte ou du pseudo graphique.

Table d'adressage (1.2) simplifiée (Attributs) :

Ligne	Colonne 0	Colonne 31
0	&h6800	&h681F
1	&h6820	&h683F
2	&h6840	&h685F
3	&h6860	&h687F
4	&h6880	&h689F
5	&h68A0	&h68BF
6	&h68C0	&h68DF
7	&h68E0	&h68FF
8	&h6900	&h691F
9	&h6920	&h693F
10	&h6940	&h695F
11	&h6960	&h697F
12	&h6980	&h699F
13	&h69A0	&h69BF
14	&h69C0	&h69DF
15	&h69E0	&h69FF

Cependant, les éléments pseudos graphiques seront toujours sur fond noir. Cela nous permet d'avoir des couleurs supplémentaires sur le screen 1.

En fonction des bits des 2 octets, l'affichage va donc varier.

Plage d'adressage 1 : Type de caractère (texte ou pseudo graphique) dit Data.

Plage d'adressage 2 : Couleur et Texte ou pseudo graphique, dit Attributs.

Table de couleur des textes ou graphique (G).

Adressage 2 (&h6800)		Adressage 1 (&h6000)			Couleur
&x----	&x0000	&h00	&x----	&x----	0
	&x0001	&h01			1
	&x0010	&h02			G
	&x0011	&h03			G
	&x0100	&h04			2
	&x0101	&h05			3
	&x0110	&h06			G
	&x0111	&h07			G
	&x1000	&h08			0
	&x1001	&h09			1
	&x1010	&h0A			G
	&x1011	&h0B			G
	&x1100	&h0C			2
	&x1101	&h0D			3
	&x1110	&h0E			G
	&x1111	&h0F			G

On constate une répétition

Nous sommes sur un adressage 11 bits pour le mode écran 1.

&x 1??? ?c0i nnnn nnnn

Concernant l'affichage de texte :

- Les bit 0 à 8 sont la représentation des caractères ASCII.
  - Le bit numéro 10 toujours à 0.
  - Les bits 9 et 11, vont faire varier les 4 couleurs disponibles de la palette (cf. COLOR).
    - Le bit 9 (i) provoque l'inversion vidéo.
  - Le bit 16 à 1.
  - Les autres bits (?) ne semblent pas avoir d'influences.

## **Mode 1 et pseudo graphique**

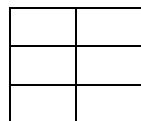
Le mode impossible, mais pas français....

Il y a 512 cases disponibles qui ont la dimension d'un caractère.

Ces cases ne peuvent avoir que 2 couleurs dont obligatoirement le noir (le fond).

L'autre couleur étant choisie dans la palette (8 possibles, voir [COLOR](#)).

Les cases ont une dimension de 2x3.



Subdivisé à la résolution :

Chaque « O » est un pixel de la résolution de 256x192.

Si la figure ci-dessus représente le coin haut, gauche :

- Elle couvre les points de (0,0) à (7,11) soit un bloc 96 pixels.

- L'adresse de cette case mémoire vidéo est &h6000.

Un POKE sur &h6000 va donc allumer 0 à 6 des portions avec une des 9 couleurs selon la palette (1 ou 2, voir [COLOR](#)).

Le codage pseudo graphique utilise l'adressage vu précédemment.

2 octets aux plages d'adresse (&h6000-&h61FF)-(&h6800-&h69FF).

Soit l'octet &xnnnnnnn. Dont nous trouvons les bits dans la matrice de résolution vue ci-dessus.

5	4
3	2
1	0

Ces 6 bits sont sur la plage (&h6000-&h61FF).

Valeurs des 6 bits :

- 0 : zone éteinte, couleur noir)
- 1 : zone allumé, couleur en fonction des autres bits

Il y a 9 couleurs disponibles, cela implique 3 bits pour le codage. Vue précédemment, l'encodage est sur 2 octets. Soit 16 bits. Après analyse et décryptage ci-dessus, le décodage donne :

- n : bit graphique
- x : bit sans influence
- c : bit pour la couleur

### **&h1xxx 0c1x ccnn nnnn**

Il y a donc 64 pseudos graphiques bicolor.

0 ; 4 ; 8 ; C

2 ; 6

TO DO :

Plage d'adressage 1 et 2 en mode 2 écran.

- &hE000 - &hE1FF (probable 2.1)
- &hE800 - &hE9FF (probable 2.2)

## Instructions graphiques sur SCREEN 1

Il est possible de tracer des lignes, des rectangles, des rectangles pleins, etc. avec les fonctions BASIC. Il faut cependant considérer que la résolution est de 32x16 pixel.

Ligne et point sont des gros « pâtés » de la taille d'un caractère.

Cela peut avoir son utilité pour éventuellement peindre une zone.

PSET et PRESET provoque des effets de bord sur l'écran qui doivent être analysés.

Ces instructions modifient l'état des octets des 2 plages d'adresse du mode.

Il est recommandé de ne pas les utiliser.

## Exemples

Exemple 1 :

```
1000 REM Poke direct sur le screen 1
1010 SCREEN 1,1,1: CLS
1020 A=0
1030 FOR I=&h10 to &h85
1040 POKE &h6000+A,I
1050 A=A+1
1060 NEXT I
1070 GOSUB 9980
1080 END
9980 K$="" : K$=INKEY$ : IF K$="" THEN 9980
9990 RETURN
```

Ce programme affiche les caractères de **&h10** à **&h85** sur un screen 1.

Ce principe permet d'aller assez vite pour afficher à l'écran car on écrit directement dans la mémoire vidéo du PHC-25.

Exemple 2 :

Pour faire un LOCATE 10,10 (colonne 10, ligne 10).

L'adresse de la ligne 10 est **&h6140**.

La colonne 10 est un incrément de **&hA**.

La localisation est donc **&h614A**.

```
POKE &h614A,&h2A
POKE &h6140+&hA,&h2A
POKE &h6140+10,&h2A
```

Le résultat de ces 3 lignes est identique.

Affichera « \* » en colonne 10 ligne 10.

L'instruction CONSOLE ne bloquera pas le POKE.

## Le Screen 2

### Description

Le Screen 2 est un mixe entre mode texte et mode graphique. La résolution texte est de 32 colonnes et 16 lignes. La résolution graphique est de 64x48 pixels.

Un graphique ne peut pas s'afficher correctement sur du texte déjà présent à l'écran.

Le fond pour la partie graphique doit toujours être BLACK (valeur 0).

Il est conseillé d'utiliser [CONSOLE](#) pour définir les lignes sur lesquelles mettre du texte ou du graphique.

Il y a 2 palettes de couleurs disponibles (voir [COLOR](#)). Un changement de palette affecte l'écran entier.

Concernant le texte, il est possible de faire des **PRINT CHR\$(n)**, sauf des caractères spéciaux de **&h10** à **&h1C**.

La mémoire vidéo est la même qu'avec le SCREEN 1.

- **&h6000** à **&h61FF**. Plage d'adressage 1 (Data).
- **&h6800** à **&h69FF**. Plage d'adressage 2 (Attributs).

Ce mode est recommandé car il apporte plus de possibilité que le mode 1.

Cependant, faire attention au texte qui n'a pas de fond noir.

### Graphique en mode 2

L'instruction SCREEN 2, met l'écran en mode 2 et effectue un **CLS** sur fond noir.

Le fonctionnement est identique à ce que nous avons vu pour le mode 1. Le fond étant noir.

Subdivisé à la résolution :

O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O

Chaque « O » est un pixel de la résolution de 256x192.

Si la figure ci-dessus représente le coin haut, gauche :

Elle couvre les points de (0,0) à (7,11) soit un bloc 96 pixels.

Un **PSET** mis sur le pixel 0,0 allume tous les pixels de (0,0) à (3,3) et affecte la couleur sur la zone (les 6 blocs).

L'adresse de cette case mémoire vidéo est **&h6000**.

Un **POKE** sur **&h6000** va donc allumer 0 à 6 des portions avec une des 9 couleurs selon la palette (1 ou 2, voir **COLOR**) et uniquement sur fond noir.

Il faudra aussi prévoir un **POKE** sur l'adresse d'attribut le cas échéant.

Quand 2 lignes graphiques se croisent, à l'intersection, la case prend la couleur de la dernière ligne qui s'y affiche Ceci pour tous les pixels déjà allumés (ils font 4x4).

Positionner un pixel (**PSET**) implique la mise à la même couleur des pixels déjà présent (0 étant non présent).

## POKE mémoire vidéo

Comme en SCREEN 1 (voir description du screen 1).

Rappel :

Octet Data : &h6800 à &h69FF.

Octet Attribut : &h6000 à &h61FF

**&x 1xxx xc0i nnnn nnnn**

- Les bit 0 à 8 sont la représentation des caractères ASCII.
- Le bit 10 toujours à 0 pour ASCII.
- Les bits 9 et 11, vont faire varier les 4 couleurs disponibles de la palette (cf. COLOR).
- Le bit 16, toujours à 1.
- x : bit sans influence

**&h 1xxx xc1x ccnn nnnn**

- Bits 0 à 6, schéma graphique
- c : bit pour la couleur
- Bit 10 : Toujours à 1 en graphique
- Le bit 16 : toujours à 1.
- x : bit sans influence

Le bit no 10 de la plage d'adresse étant à 1, il faut le mettre à 0 pour afficher les caractères de &h10 à &h1C.

NDR : TODO how to.

Il peut être intéressant pour un affichage texte de positionner le bit 10 à 1, puis par programme le positionner à 0 pour créer une animation type déchiffrement du texte.

# Le Screen 3

## Description

Le SCREEN 3 offre une résolution graphique de 128x192 pixels.

En mode texte, la résolution est de 16x16 caractères.

Il y a 4 couleurs disponibles sur un choix de 2 palettes (voir [COLOR](#)).

En fin de programme, le PHC-25 retourne automatiquement en mode 1.

Ce qui implique que tout l'écran est perdu.

D'où le mode 2 écrans du PHC-25, afin de ne pas perdre les tracés au détriment de la mémoire disponible.

Changement par les touches « Control+Q » ou par l'instruction [SCREEN](#).

Il n'est pas possible d'utiliser l'instruction [INPUT](#), cela provoque un retour en mode screen 1 (mode texte), et implique que tous les graphiques sont perdus.

## Plage d'adresse

De &h6000 à &h77FF pour l'écran 1.

De &hE000 à &hF700 pour l'écran 2.

Chaque plage d'adresse occupe 6Ko.

## Particularité du texte

Il n'est pas possible d'afficher les caractères de **&h10** à **&h1C** via **PRINT CHR\$(n)**.

## Particularité des pixels

Les pixels sont plats.

Les coordonnées commencent en haut à gauche au point (0,0).

L'octet encode les 8 pixels assemblés 2 par 2. D'où leur platitude.

2 bits donnent 4 couleurs possibles selon la palette c (voir [COLOR](#)).

bit	c(1)	c(2)
00	Vert	Blanc
01	Jaune	Rose
10	Bleu	Bleu
11	Rouge	Orange

Les 4 pixels se codent : &x aabbccdd à l'adresse mémoire.

## **Le Screen 4**

### **Description**

Le SCREEN 4 offre une résolution graphique de 256x192 pixels.

En mode texte, la résolution est de 32x16 caractères.

Il y a 2 couleurs disponibles en 4 variantes (voir [COLOR](#)).

Il n'est pas possible d'avoir du texte inversé.

En fin de programme, le PHC-25 retourne automatiquement en mode 1.

Ce qui implique que tout l'écran est perdu.

D'où le mode 2 écrans du PHC-25, afin de ne pas perdre les tracés au détriment de la mémoire disponible.

Changement par les touches « Ctrl+Q » ou par l'instruction [SCREEN](#).

Il n'est pas possible d'utiliser l'instruction [INPUT](#), cela provoque un retour en mode screen 1 (mode texte), et implique que tous les graphiques sont perdus. Il faut donc opérer une bascule, faire l'[INPUT](#) puis rebasculer.

### **Plage d'adresse**

De &h6000 à &h77FF pour l'écran 1.

De &hE000 à &hF700 pour l'écran 2.

Chaque plage d'adresse occupe 6Ko.

### **Particularité du texte**

Il n'est pas possible d'afficher les caractères de **&h10** à **&h1C** via **PRINT CHR\$(n)**.

### **Particularité des pixels**

Les pixels sont carrés.

Les coordonnées commencent en haut à gauche au point (0,0).

L'encodage est du binaire sur 1 octet.

L'adresse **&h6000**, contient 1 octet et définit les 8 pixels de la première ligne.

Un pixel allumé est un bit à 1.

**TO DO : Trouver l'adresse mémoire qui définit les 4 variantes.**

# Jeu de caractères

## Les caractères

Les caractères sur le SAYNO PHC-25 sont dans une matrice de 8x12 pixels.  
Ils occupent donc 12 octets en ROM.

Adresse mémoire d'encodage des caractères :

Début	Fin	Caractères
&h4FEC	&h516B	ASCII Control
&h516C	&h55EB	ASCII Standard
&h55EC	&h5BEB	ASCII Étendu

La zone étendue est vierge dans la ROM Basic v1.3 de SANYO.

Il n'y a donc rien à afficher.

Cette plage est utilisé par la ROM alternative pour étendre les possibilité du PHC-25 (RPUFOS Basic v1.4).

## Le générateur de caractères

De &h00 (00) à &h0F (15)

**TO DO.** : check des opérateurs ASCII à faire.

À &h10 (16)

Le caractère de PI.

NDR : il n'y a pas d'instruction PI.

Ce caractère n'est pas affichable via PRINT CHR\$(n). Cependant il est affichable via un [POKE](#) à l'adresse mémoire vidéo des mode 1 et 2.

Il est affichable via la combinaison de touche []+[] avec l'instruction [PRINT](#).

De &h11 à &h1B

Les éléments pour tracer des tableaux en mode texte.

Hexa	11	12	13	14	15	16	17	18	19	1A	1B
Déc	17	18	19	20	21	22	23	24	25	26	27
Car.	—	—	—	—	+	—	—	—	—	L	—

Tous ces caractères ne sont pas affichables via PRINT CHR\$(n). En revanche ils sont affichables via un POKE à l'adresse mémoire vidéo des modes 1 et 2.

Ils sont affichables via la combinaison de touche []+[] avec l'instruction [PRINT](#).

À &h1C (28)

Une croix bien centrée (X), différente du x (majuscule ou minuscule).

Ce caractère n'est pas affichable via PRINT CHR\$(28). Cependant il est affichable via un [POKE](#) à l'adresse mémoire vidéo des mode 1 et 2.

#### **TODO Exemple**

Il est affichable via la combinaison de touche []+[] avec l'instruction [PRINT](#).

De &h20 (32) à &h7F

Caractères ASCII alphanumériques standard.

De &h80 (128) à &h83 (131).

Les caractères pique, cœur, trèfle et carreau.

Hexa	80	81	82	83
Déc	128	129	130	131
Car.	♠	♥	♣	♦

Ces caractères sont affichables avec PRINT CHR\$(n).

De &h84 à &h85

- Rond vide (○),
- Rond plein (●).

Ces caractères sont affichables avec PRINT CHR\$(n).

Toutes les autres valeurs afficheront un espace vide.

NDR : à revoir pour la ROM Japonaise.

## **Hack de la ROM ?**

Il y a un nombre d'espace vide assez conséquent sur la ROM PAL. Ne pas oublier qu'à l'origine c'est un ordinateur Japonais et qu'il avait besoin d'un jeu de caractères large.

Cependant, rien n'empêche de modifier la ROM PAL pour ajouter un nombre conséquent de caractères au PHC-25.

Ce travail a été effectué et il existe une ROM PAL Alternate.

Voir le [GITHUB Dedié](#).

TO DO : étudier la ROM JAP.

# Trucs et Astuces

## Attendre l'appui sur une touche

```
1000 K$=INKEY$:IF K$="" THEN 1000 ELSE RETURN
```

Un GOSUB 1000, fera un saut ligne 1000, il y aura alors attente jusqu'à ce que l'utilisateur appuie sur une touche.

K\$ prend la valeur de cette touche. K\$ peut être exploitée pour la suite du programme après l'instruction RETURN.

NDR : Faire la purge de INKEY\$ pour éviter la mémorisation.

NDR : à étudier pour simuler un INPUT en SCREEN 3 et 4.

## RANDOMIZE

Comme vu dans la description de la fonction RND. Donner une valeur négative réinitialise la séquence.

```
1000 I=RND (-TIME) :RETURN
```

En utilisant TIME, le seed sera en principe toujours différent.

Il est possible de lancer le sous-programme plusieurs fois durant l'exécution du programme principal. D'où l'intérêt d'une sous-routine.

La variable I (ou tout autre) est obligatoire.

NDR : Trouver un nom de variable autre pour éviter des problèmes si appel à l'intérieur d'une boucle FOR qui utilise I.

## **Boucle WHILE WEND**

Le principe de cette boucle et d'exécuter le code si le test est valide. C'est à utiliser s'il faut faire les actions de 0 à N fois.

Le test doit donc être fait en début de boucle.

Le SANY PHC-25 ne possède pas ces instructions, ci-dessous comment convertir cette boucle.

Exemple :

```
10 REM WHILE
20 IF [TEST] THEN 110
...
100 GOTO 20: REM WEND
110 [SUITE]
```

Ligne 20 faire le test nécessaire et si faux alors suite en 110. Fin de boucle.

Ligne 100, boucle sur le test.

Ligne 110 suite du programme.

NDR : Sans le savoir ce type de boucle est utilisée. Principe applicable à une sous-routine.

## **Boucle DO UNTIL**

Le principe de la boucle DO UNTIL et d'exécuter au moins 1 fois le code.

Le test doit donc être fait en fin de boucle.

Le SANY PHC-25 ne possède pas ces instructions, ci-dessous comment convertir cette boucle.

Exemple :

```
10 REM DO
20 ...
100 IF [TEST] GOTO 20: REM UNTIL
110 [SUITE]
```

Ligne 20, début de la boucle.

Ligne 100, le test.

Ligne 110, suite du programme.

NDR : Sans le savoir ce type de boucle est utilisée. Principe applicable à une sous-routine.

## **Faire un XOR**

Voici l'algèbre pour réaliser le XOR avec les fonctions disponibles.

**A XOR B = (A AND NOT B) OR (NOT A AND B)**

Cela permet de travailler avec des masques sur des octets par exemple. Particulièrement avec les [INP](#) et [OUT](#) ou avec les [PEEK](#) et [POKE](#).

## **Faire un NAND**

Voici l'algèbre pour réaliser un NAND avec les fonctions disponibles.

**A NAND B=NOT (A AND B)**

Cela permet de travailler avec des masques sur des octets par exemple. Particulièrement avec les [INP](#) et [OUT](#) ou avec les [PEEK](#) et [POKE](#).

## **Faire un NOR**

Voici l'algèbre pour réaliser un NOR avec les fonctions disponibles.

**A NOR B = NOT (A OR B)**

Cela permet de travailler avec des masques sur des octets par exemple. Particulièrement avec les [INP](#) et [OUT](#) ou avec les [PEEK](#) et [POKE](#).

## **Forcer 1 ou 2 écrans**

Ceci peut s'avérer nécessaire dans le cas où le programme doit utiliser 2 écrans. Typiquement lors de tracé de courbe mathématique ceci afin de ne pas perdre le tracé.

Test si 1 ou 2 écrans

TO DO

Faire un PEEK(&HFB58) au plus simple. 223=2, 247=1

Bascule 1 vers 2 et inversement

2 vers 1

**1000 POKE &HFB58,247:20:POKE &HFB56,1:CLEAR 50,&HF800:RETURN**

1 vers 2

```
1000 CLEAR 50, &HE000:POKE &HFB58,223:POKE &HFB56,20:RETURN
```

La différence et l'allocation ou pas de l'espace mémoire de la mémoire vidéo 2.

## ***La fonction MOD***

La fonction MOD est le modulo (le reste) d'une division mathématique.

Le PHC-25 n'a pas cette fonction.

Pour simuler cette fonction :

Soit la formule :

```
x = y MOD z
```

En BASIC PHC-25 écrire :

```
x = y - (INT(y/z)*z)
```

**NDR :** Passer 2 paramètres dans un DEF FN ne fonctionne pas, contrairement à ce qui est écrit dans la documentation.

## ***La fonction de division entière (\)***

Cette fonction n'est pas disponible dans le PHC-25.

La fonction permet d'obtenir la valeur entière de la division de Y/Z.

Soit la formule :

```
x = y \ z.
```

En BASIC PHC-25 écrire :

```
x = INT(y/z)
```

Il suffit de s'en souvenir pour les conversions de code.

## Majuscules et Minuscules

Rappel des fonctions de manipulation de chaînes de caractères :

ASC, CHR\$, LEFT\$, MID\$, RIGHT\$, STR\$, VAL

Le PHC-25 ne dispose pas des fonctions classiques UPPER\$ et LOWER\$ que l'on trouve dans d'autres BASIC.  
Les sous-programmes suivants pallient ce manque.

```
10000 REM -----
10010 REM A$=UPPER$ (A$)
10020 REM -----
10030 FOR II=1 TO LEN(A$)
10040   CC$=MID$ (A$, II, 1)
10050   JJ=ASC(CC$)
10060   IF JJ>=97 AND JJ<=122 THEN CC$=CHR$ (JJ-32)
10070   A$=LEFT$ (A$, II-1)+CC$+RIGHT$ (A$, LEN(A$)-II)
10080 NEXT II
10090 RETURN
```

```
10100 REM -----
10110 REM A$=LOWER$ (A$)
10120 REM -----
10130 FOR II=1 TO LEN(A$)
10140   CC$=MID$ (A$, II, 1)
10150   JJ=ASC(CC$)
10160   IF JJ>=65 AND JJ<=90 THEN CC$=CHR$ (JJ+32)
10170   A$=LEFT$ (A$, II-1)+CC$+RIGHT$ (A$, LEN(A$)-II)
10180 NEXT II
10190 RETURN
```

Placer la chaîne à convertir dans A\$, appeler la sous routine. A\$ sera changé.

Utilise une chaîne intermédiaire CC\$ et deux variable II et JJ.

Ces trois variables sont donc changées durant le processus, donc attention lors de l'adaptation. Ce pourquoi elles sont doublées.

Si le caractère n'est pas alphabétique le sous-programme le reporte tel quel.

Garbage sur les intermédiaires pour éviter l'overflow mémoire à faire (Oui, mais comment ?).

Possible changement dans une version futur de ce document.

## **Conversion LOCATE et adresse mémoire vidéo**

Algorithme pour transformer un LOCATE en Adresse vidéo et inversement.

### **Pour SCREEN 1 et 2.**

#### **X,Y vers mémoire**

Pour obtenir l'adresse à partir de (x, y), on prend la base **&H6000**, puis on ajoute ( $y \times 32$ ) + x (chaque ligne contient 32 caractères).

Par calcul :

```
1000 REM Convertir (X,Y) en adresse mémoire
1010 REM Entrée : X,Y
1020 REM Sortie : AD
1030 AD=&H6000+(Y*32)+X
1040 RETURN
```

Par fonction DEF FN :

```
1000 REM Convertir (X,Y) en adresse mémoire
1010 DEF FN AD(X,Y)=&H6000+(Y*32)+X
```

Problème avec le DEF FN non fonctionnel sur le PHC-25 (NDR : contrairement à ce que dit la notice, sauf si problème sur émulateur).

#### **Mémoire vers X,Y**

Pour retrouver x à partir d'une adresse, on fait MOD 32, qui nous donne la position horizontale.

Pour retrouver y, on divise (\ division entière) par 32 pour obtenir la ligne correspondante.

Par calcul :

```
2000 REM Convertir adresse mémoire en (X,Y)
2010 REM Entrée : AD
2020 REM Sortie : X,Y
2030 X = (AD - &H6000) MOD 32
2040 Y = (AD - &H6000) \ 32
2050 RETURN
```

Par fonction DEF FN :

```
2000 REM Convertir adresse mémoire en (X,Y)
2010 DEF FN X(AD)=(AD-&H6000) MOD 32
2020 DEF FN Y(AD)=(AD-&H6000) \ 32
```

Mais notre BASIC PHC-25 n'a ni MOD ni la division entière (\).

Nous l'avons vue précédemment. Il faut donc s'adapter.

Par calcul :

```
2000 REM Memory Address to (X,Y)
2010 REM IN : AD
2020 REM OUT : X,Y
2030 X = (AD-&H6000)-(INT((AD-&H6000)/32)*32)
2040 Y = INT((AD-&H6000)/32)
2050 RETURN
```

Par DEF FN :

```
2000 REM Memory Address to (X,Y)
2010 REM AD: Address to convert
2020 REM X and Y, equal LOCATE X,Y
2030 DEF FN X(AD)=(AD-&H6000)-(INT((AD-&H6000)/32)*32)
2040 DEF FN Y(AD)=INT((AD-&H6000)/32)
```

## Tests des adresses adjacentes

```
AU=AE-32 : REM UP
AD=AE+32 : REM DOWN
AL=AE-1  : REM LEFT
AR=AE+1  : REM RIGHT
```

Tester les valeurs

Limites

Si AE<&h6000 alors hors écran

Si AE>&h61FFF alors hors écran

Tests des bords, gauche et droite à faire. En effet un +1 sur le bord droit fait passer à la ligne suivante.

## Simuler INPUT sur SCREEN 3 et 4

La fonction INPUT n'est pas fonctionnelle sur ces modes. Il faut donc contourner le système pour simuler le comportement.

Pour le PRINT, il se comporte comme un ensemble de PSET. Fonctionne comme si c'était en transparence.

NDR : Test de PRINT en mode inverse à faire.

Ci-dessous un programme de démonstration de la sous-routine en mode SCREEN 4.

```
1000 REM -----
1010 REM EXPERIMENTAL INPUT
1020 REM -----
1030 SCREEN 4,1,1:COLOR 0,0,2:CLS:COLOR 1
1040 LOCATE 0,0: PRINT"Experimental INPUT, on line 15."
1050 PRINT"Type your text then return.":PRINT"25 char max."
1060 GOSUB 7030
1070 LOCATE 0,4:PRINT"Input": PRINT K$
1080 LINE(0,96)-(255,96),1: REM To demonstrate screen 4
1090 LINE(100,100)-(120,120),1,B:LINE(110,110)-(130,130),1,B
1100 LINE(100,100)-(110,110),1:LINE(120,100)-(130,110),1
1110 LINE(100,120)-(110,130),1:LINE(120,120)-(130,130),1
1120 LINE(10,110)-(50,150),1,BF:LINE(0,179)-(255,179),1
1130 A$=INKEY$:IF A$="" THEN 1070
1140 END
7000 REM -----
7010 REM FOR SCREEN 4
7020 REM -----
7030 A$="":K$="":LOCATE 0,15:PRINT">";
7040 A$=INKEY$:IF A$="" THEN 7040
7050 IF A$=CHR$(13) THEN A$="":GOSUB 7100:RETURN:REM &h0D
7060 IF A$=CHR$(127) AND LEN(K$)>0 THEN K$=LEFT$(K$,LEN(K$)-1):GOTO 7090 :REM Suppr &h7F
7070 IF ASC(A$)<&H20 OR ASC(A$)>&H80 OR A$=CHR$(127) THEN A$="":REM Filter
7080 IF LEN(K$)<25 THEN K$=K$+A$
7090 GOSUB 7100:LOCATE 0,15:PRINT">";K$,:GOTO 7040
7100 CONSOLE 15,1:CLS:CONSOLE 0,16:RETURN
9010 REM PRINT is by transparency, warn!
```

En SCREEN 3, il faudra réduire le nombre de caractère à 14 (ligne 7080).

## **Conversion DEG et RAD**

Le PHC-25 ne connaît que les Radians.

La variable PI n'existe pas dans le BASIC du PHC-25, il est recommandé de la créer s'il y en a besoin dans votre programme. Cela implique de ne pas utiliser de variable ayant comme nom « PI ».

### **RAD vers DEG**

```
10 PI=3.141592653: REM Define PI  
1000 D=R*(180/PI)
```

Formule que l'on peut placer dans un DEF FN.

```
10 PI=3.141592653: REM Define PI  
1000 DEF FN DG(R)=R*(180/PI)
```

Utilisation.

```
2000 X=FN DG(PI/2)
```

Converti PI/2 en radian, X vaudra 90°.

### **DEG vers RAD**

```
10 PI=3.141592653: REM Define PI  
1000 R=D*(PI/180)
```

Formule que l'on peut placer dans un DEF FN.

```
10 PI=3.141592653: REM Define PI  
1000 DEF FN RD(D)=D*(PI/180)
```

Utilisation.

```
2000 X=COS(FN RD(45))
```

Calcul le cosinus de 45° et le place dans X.

# Formules Graphique

## Tracé de cercle

Pour rappel, le PHC-25 ne connaît que les radians.

Un cercle de base se trace de façon basique comme suit :

```
1000 REM -----
1010 REM Program to draw a circle
1020 REM -----
1030 X=10      : REM x-coordinate of the center
1040 Y=10      : REM y-coordinate of the center
1050 R=5       : REM Radius of the circle
1060 GOSUB 2000: REM Call the subroutine
1070 END

2000 REM -----
2010 REM Subroutine to draw the circle
2020 REM -----
2030 FOR A=0 TO 360 STEP 1
2040   XC=X+R*COS (A*3.141592653/180)
2050   YC=Y+R*SIN (A*3.141592653/180)
2060   PSET (XC, YC) ,1
2070 NEXT A
2080 RETURN
```

Cette méthode est très lente et n'est pas optimum. Elle est là uniquement à titre indicatif.

NDR : Vérifier le code.

Réduire le code en intégrant les variables XC et YC.

Cercle rapide par la méthode du polygone.

```
1000 REM -----
1010 REM Circle by Polygon
1020 REM -----
1030 SCREEN 4,1,1:CLS
1040 PI=3.141592653
1050 X=127:Y=96:R=50
1060 GOSUB 2030
1070 GOSUB 3030
1080 GOTO 9999
1090 REM
2000 REM -----
2010 REM Circle Sub-Routine
2020 REM -----
2030 A=15*PI/180:X0=X+R:Y0=Y
2040 FOR I=1 TO 24
2050   X1=X+R*COS(I*A):Y1=Y+R*SIN(I*A)
2060   LINE (X0,Y0)-(X1,Y1),1
2070   X0=X1:Y0=Y1
2080 NEXT I
2090 RETURN
3000 REM -----
3010 REM Wait key
3020 REM -----
3030 K$"":K$=INKEY$:IF K$="" THEN 3030
3040 RETURN
9999 END
```

NDR: Il faudrait ajouter un calcul pour optimiser A et le step du FOR. Selon que le cercle est grand ou petit.

Vous remarquerez que lorsque le cercle se trace, le tracé part vers le bas, dans le sens contraire trigonométrique (horaire).

Ceci est du fait que les coordonnées graphique (0,0) sont en haut à gauche.

Pour être dans les conventions, il faut modifier la ligne 2050.

Le nouveau programme ci-après.

```

1000 REM -----
1010 REM Circle by Polygon
1020 REM -----
1030 SCREEN 4,1,1:CLS
1040 PI=3.141592653
1050 X=127:Y=96:R=50
1060 GOSUB 2030
1070 GOSUB 3030
1080 GOTO 9999
1090 REM
2000 REM -----
2010 REM Circle Sub-Routine
2020 REM -----
2030 A=15*PI/180:X0=X+R:Y0=Y
2040 FOR I=1 TO 24
2050   X1=X+R*COS(I*A):Y1=Y-R*SIN(I*A)
2060   LINE (X0,Y0)-(X1,Y1),1
2070   X0=X1:Y0=Y1
2080 NEXT I
2090 RETURN
3000 REM -----
3010 REM Wait key
3020 REM -----
3030 K$="":K$=INKEY$:IF K$="" THEN 3030
3040 RETURN
9999 END

```

Une soustraction est appliquée au lieu d'une addition.

Ce changement dans l'ordonnée est important pour tracer correctement des courbes sur le PHC-25, en particulier lors de l'adaptation de programme. Voir plus loin.

## **Tracé d'ellipse**

TO DO

## Tracé de fonction

```
1000 REM -----
1010 REM Function Plotter with Center Origin
1020 REM -----
1030 SCREEN 4,1,1:CLS
1040 GOSUB 4030:GOSUB 5030: REM Define PI and test for 2 screens
1050 W=256:H=192
1060 CX=W/2 : REM Center X
1070 CY=H/2 : REM Center Y
1080 GOSUB 2000 : REM Draw Axes
1090 FOR X=-128 TO 127: REM Range for X centered
1100 Y=FN F(X): REM Call the function
1110 Y1=CY-Y: REM Change of origin to center
1120 PSET(CX+X,Y1),1: REM Set pixel color to 1
1130 NEXT X
1140 GOSUB 3030
1150 GOTO 9999
1160 REM
2000 REM -----
2010 REM Draw Axes Sub-Routine
2020 REM -----
2030 LINE (0,CY)-(W,CY),1: REM X-axis
2040 LINE (CX,0)-(CX,H),1: REM Y-axis
2050 RETURN
3000 REM -----
3010 REM Wait key
3020 REM -----
3030 K$=INKEY$:IF K$="" THEN 3030
3040 RETURN
4000 REM -----
4010 REM Function Definition
4020 REM -----
4030 PI=3.141592653: REM Define PI
4040 DEF FN F(X)=50*SIN(X*2*PI/128): REM Example function
4050 RETURN
5000 REM -----
5010 REM Change to 2 screens if necessary
5020 REM -----
5030 REM TO DO
5090 RETURN
9999 END
```

NDR: Amélioration à faire pour forcer un mode 2 écrans, 1 pour les input et l'autre le tracé.

CX et CY sont l'origine (0,0) pour la courbe à tracer.

Il est possible de les sortir de la sous-routine et de les utiliser comme variables d'entrées.

# Jeux Graphique

## 10 PRINT

Tirer du livre « **10 PRINT CHR\$(205.5+RND(1)); : GOTO 10** » publié en 2013.

Le PHC-25 n'a pas vraiment de caractère graphique pour jouer. Mais on peut faire avec ses faiblesses.

Plutôt que d'afficher les caractères à la suite des uns des autres, ils peuvent être placés de façon pseudo aléatoire sur l'écran.

Une seule ligne ne peut pas forcément être utilisée. Les programmes feront du 2 lignes.

La ligne 1 force un RANDOMIZE sinon on aurait toujours la même chose.

Toujours sur la ligne 1, passer en SCREEN 2 pour des effets de couleurs.

## Avec les 2 caractères « /\ »

Plusieurs solutions sont proposées.

Classique :

```
1 I=RND (-TIME) :COLOR INT (RND (1)*4)+1:C$="/\" :SCREEN 2,1,1:COLOR , ,INT (RND (1)*2)+1:CLS:  
10 PRINT MID$(C$,INT (RND (1)*LEN(C$))+1,1);:GOTO 10
```

Cette méthode est intéressante car on peut mettre ce que l'on veut dans C\$.

Par un LOCATE aléatoire sur l'écran :

```
1 I=RND (-TIME) :C$="/\" :SCREEN 2,1,1:COLOR , ,(RND (1)+1) :CLS  
10 LOCATE INT (RND (1)*32) ,INT (RND (1)*16) :COLOR INT (RND (1)*4)+1:PRINT  
MID$(C$,INT (RND (1)*LEN(C$))+1,1);:GOTO 10
```

Cette méthode sera utilisée pour les autres exemples, pour aligner les caractères, il suffira d'enlever le LOCATE.

Il y a un bug qui se produit lorsqu'il y a LOCATE 32,16. L'écran scroll d'une ligne vers le haut. Cela malgré le « ; » en fin de PRINT.

Version POKE mémoire.

## Avec les caractères pique, cœur, carreau et trèfle

C\$ est initialisé avec les 4 caractères.

```
1 I=RND (-TIME) :C$=CHR$ (&H80)+CHR$ (&H81)+CHR$ (&H82)+CHR$ (&H83) :SCREEN 2,1,1:COLOR  
,,INT(RND (1)*2)+1:CLS  
10 LOCATE INT(RND (1)*32),INT(RND (1)*16):COLOR INT(RND (1)*4)+1:PRINT  
MID$(C$,INT(RND (1)*LEN(C$))+1,1);:GOTO 10
```

Les 4 caractères s'afficheront de façon aléatoire sur l'écran.

## Avec les ronds

C\$ est initialisé avec les 2 caractères.

```
1 I=RND (-TIME) :C$=CHR$ (&H84)+CHR$ (&H85) :SCREEN 2,1,1:COLOR ,,(RND (1)+1):CLS  
10 PRINT :GOTO 10
```

Les 2 caractères s'afficheront de façon aléatoire sur l'écran.

## Exemple avec POKE

Le programme affiche directement sur la page écran le caractère tiré au sort.

Il s'agit des caractères de tracé de tableau (&H11 à &H1B).

```
1 REM cload; I=RND (-TIME) :CLS:RUN  
10 POKE (INT(RND(1)*(&H61FF-&H6000+1))+&H6000),(INT(RND(1)*(&H1B-&H11+1))+&H11):GOTO 10
```

La ligne 1 est une indication pour charger et lancer le programme.

Si la première ligne écran n'est pas prise en compte, elle peut être utilisée pour un score ou un décompte. Le joueur part du haut en haut à gauche vers le point d'arrivée en bas à droite (&H61FF). Ce point étant lui aussi exclu du random.

Utiliser les fonctions de position du curseur, [CSRLIN](#) et [POS](#).

La fonction test de l'élément texte sur l'écran, [SCRIN](#).

Etc.

## **Mosaïque de bloc**

TO DO

# **INP et OUT**

La documentation est vierge sur ce domaine. Cependant voici quelques informations.

Elles sont issues du GitLab :

[https://gitlab.com/mokona/phc25\\_tools/-/blob/main/documentation/phc25-tech-information.md](https://gitlab.com/mokona/phc25_tools/-/blob/main/documentation/phc25-tech-information.md)

## **Les Ports I/O**

<b>Port</b>	<b>Description</b>
&h00	centronics port (printer), write
&h40	read/write
&h80-&h88	Keyboard, read
&hC0	AY8910 (sound), write
&hC1	AY8910 (sound), read/write

NDR : pas d'autres informations sur les autres ports (rappel &H00 à &HFF)

## **Port &H00**

### **INP**

### **OUT**

## **Port &H40**

Lecture écriture Magnétophone et Imprimante.

Synchro Verticale et Horizontale écran ?

### **INP**

<b>Bit</b>	<b>Information</b>
<b>0</b>	
<b>1</b>	
<b>2</b>	
<b>3</b>	
<b>4</b>	vertical sync
<b>5</b>	cassette input data
<b>6</b>	centronics busy state
<b>7</b>	horizontal sync

### **OUT**

<b>Bit</b>	<b>Action</b>
<b>0</b>	cassette output data
<b>1</b>	cassette motor on/off
<b>2</b>	Caps Lock LED on/off
<b>3</b>	Centronics Strobe
<b>4</b>	MC6847 GM1 (Graphic Mode)
<b>5</b>	MC6847 GM0 (Graphic Mode)
<b>6</b>	MC6847 CSS (Color Set Select)
<b>7</b>	MC6847 A/G (Alpha/Graphic Select)

Les bits 4, 5, 6 pour les modes écran.

S'ajoute à cela une adresse mémoire spécifique.

## ***Ports &H80-&H88***

### **INP**

<b>Bit</b>	<b>Action</b>
<b>0</b>	
<b>1</b>	
<b>2</b>	
<b>3</b>	
<b>4</b>	
<b>5</b>	
<b>6</b>	
<b>7</b>	

### **OUT**

<b>Bit</b>	<b>Action</b>
<b>0</b>	
<b>1</b>	
<b>2</b>	
<b>3</b>	
<b>4</b>	
<b>5</b>	
<b>6</b>	
<b>7</b>	

## **Ports &HC0 et &HC1**

**INP**

<b>Bit</b>	<b>Action</b>
<b>0</b>	
<b>1</b>	
<b>2</b>	
<b>3</b>	
<b>4</b>	
<b>5</b>	
<b>6</b>	
<b>7</b>	

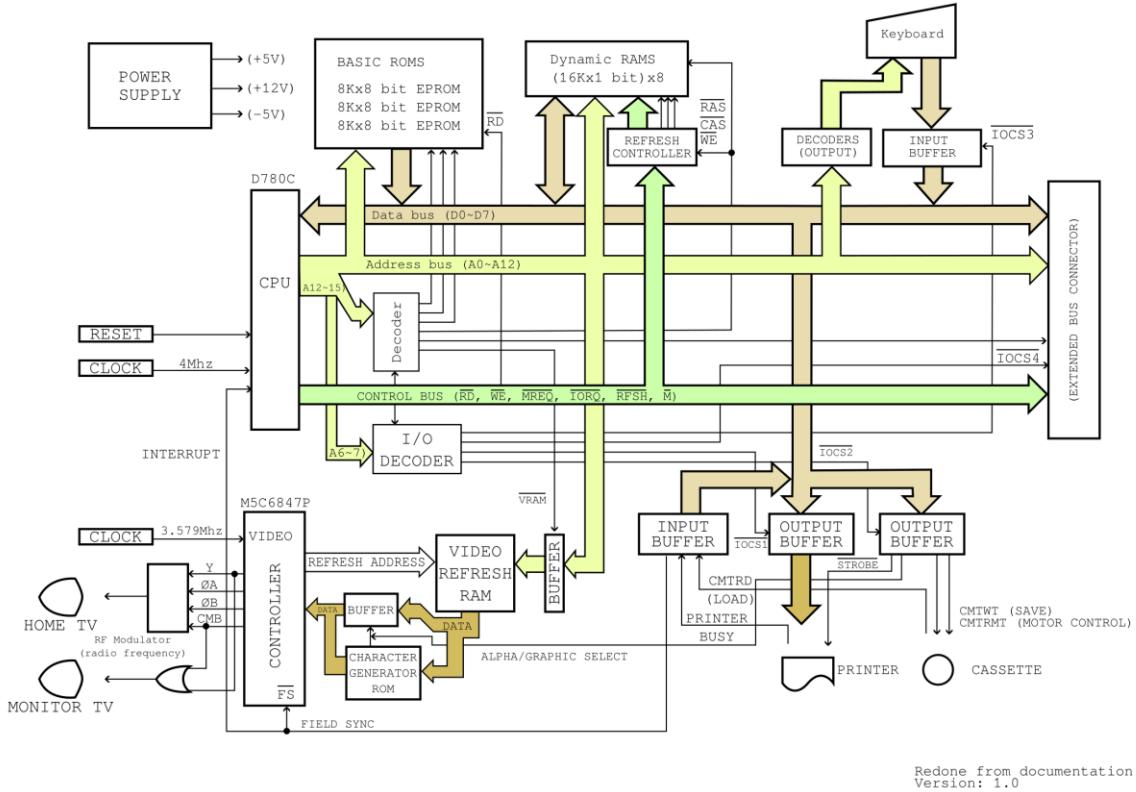
**OUT**

<b>Bit</b>	<b>Action</b>
<b>0</b>	
<b>1</b>	
<b>2</b>	
<b>3</b>	
<b>4</b>	
<b>5</b>	
<b>6</b>	
<b>7</b>	

# Dossier technique

## Block Diagram

SANYO PHC-25 BLOCK DIAGRAM



Source :

[https://gitlab.com/mokona/phc25\\_tools/-/blob/main/documentation/phc25-tech-information.md](https://gitlab.com/mokona/phc25_tools/-/blob/main/documentation/phc25-tech-information.md)

## **Les connecteurs**



De gauche à droite :

Alimentation, Extension, Imprimante, Vidéo, Inutilisé, Péritel, Magnétophone.

### **Alimentation**

La prise est une fiche **IEC 60320 C8 mâle**.

Câble électrique broche Type C (Europlug) (2 broches) mâle pour la version EU d'un côté et une IEC 60320 C8 (2 broches) femelle de l'autre côté.

Ce type de câble est très facile à trouver.

## Extension

Fiche normalisée **IDC 34 broches mâle.**

Pin	Signal	Pin	Signal
1	D0	2	Masse (GND)
3	D2	4	
5		6	
7	A12	8	
9	A10	10	A11
11	A6	12	A9
13	A8	14	A7
15	A4	16	A5
17	A2	18	A3
19	A0	20	A1
21	IOCS4	22	
23	D6	24	
25	D4	26	
27	IORQ	28	
29	RESET	30	
31	Vcc	32	Vcc
33	Masse (GND)	34	Masse (GND)

Ce modèle de fiche est très courant.

L'extension XXX se branche sur ce port, elle permet d'avoir le son et 2 ports joystick.

NDR : un lecteur de disquette était annoncé par SANYO, Format 5 1/4. Il n'a probablement jamais vu le jour. Mais cela laisse penser qu'il est possible d'en créer un. Comme la possibilité d'avoir un lecteur de SD card qui existe sur d'autres systèmes de la même période.

## **Imprimante**

Fiche Centronics (14 broches)

NDR : Ces fiches ne sont pas normalisées IEC.

<b>Pin</b>	<b>Signal</b>
1	Data strobe
2	Data 0
3	Data 1
4	Data 2
5	Data 3
6	Data 4
7	Data 5
8	Data 6
9	Data 7
10	<b>ACKNOWLEDGE</b>
11	READY
12	
13	GND
14	GND

Selon la documentation, le Pin 10 n'est pas raccordé.

## **Prise Vidéo**

Prise CINCH.

TO DO.

## **Inutilisé bleu**

On ne sait pas pourquoi c'est là.

Si vous savez, on veut bien la solution.

## Prise vidéo Péritel

Il s'agit d'une fiche DIN 8 broches femelle.



Vue côté connecteur (inverse pour soudure)

Pin	Signal	Couleur
1	Bleu	Bleu
2	Commutation Rapide	Orange
3	Masse	Noir
4	Rouge	Rouge
5	-	
6	Vert	Vert
7	+12V	-
8	Vidéo Composite	Jaune

La couleur est en principe la couleur du fil.

C'est informatif et à vérifier.

Dans la pratique, un câble RJ45 possède 8 fils. Vous pouvez donc utiliser cela pour faire la réalisation,  
Ou bien dépiauter une PERITEL pour refaire le connecteur.

**TO DO :** Mettre la connectique PERITEL pour aider à fabriquer le câble.

## Magnétophone

Il s'agit d'une fiche DIN 8 broches femelle.



Vue côté connecteur (inverse pour soudure)

Pin	Signal	Couleur
1	Masse	?
2	Masse	?
3	Masse	?
4	CMTOUT	Rouge
5	CMTIN	Blanc
6	REMOTE +	Noir
7	REMOTE -	?
8	Masse	?

La couleur est en principe la couleur du fil. C'est informatif et à vérifier.

Pour utiliser un magnétophone standard, veuillez consulter la documentation de ce dernier pour réaliser le câble. Dans la pratique, un câble RJ45 possède 8 fils. Vous pouvez donc utiliser cela pour faire la réalisation, préférence pour un Ethernet blindé et utiliser les 8 fils sur le connecteur.

Pour un usage moderne, les cassettes sont numérisées en fichier WAV.

NDR : Voir s'il existe un lecteur numérique qui accepte des commandes via câble. Ou en fabriquer un ?

# Messages d'erreur

## ***Bad subscript***

- Les valeurs désignées par des indices sont-elles en dehors des dimensions du tableau déclaré par [DIM](#) ?
- Les nombres sont-ils différents des nombres déclarés par [DIM](#) ?

## ***Can't continue***

- La commande [CONT](#) ne peut pas être utilisée après les corrections d'un programme.
- Y a-t-il des erreurs dans le programme ?

## ***Divided by zero***

- Une division par zéro a été tentée, ce qui est impossible.

Le BASIC PHC-25 considère toutes valeur inférieure à  $1 \times 10^{-38}$  comme zéro.

## ***Duplicate definition***

- Y a-t-il une déclaration [DIM](#) en cours utilisant un nom qui fait double emploi ?

Rappel : le nom des variables se limite à 2 caractères. DIM AB(n) et DIM ABC(n) sont la même variable.

## ***Illegal direct call***

- Une commande incorrecte a été faite. Les instructions [DEFFN](#) sont-elles utilisées en direct mode alors que ce n'est pas autorisé
- La longueur de la chaîne de caractères dépasse-t-elle 255 caractères ?  
Vérifiez à l'endroit où la chaîne de caractères est additionnée.

## ***Illegal function call***

- Le nombre utilisé est-il en dehors des paramètres autorisés ?

Exemple :

```
PRINT SPC(512)
```

## ***Missing operand***

- Un opérande indispensable manque.  
Le côté droit de command de substitution est-il en place ?
- Les opérateurs sont présents, mais les valeurs et caractères sont-ils correctement inscrits ?

## ***Next w/o for***

- S'agit-il d'une instruction NEXT sans l'instruction FOR ?
- Vérifier la boucle FOR – NEXT.

## ***Not enough data***

Pas assez d'information

- Lorsque l'ordinateur procède à la lecture, il n'y a pas de data.
- Vérifiez le nombre de data dans l'instruction DATA et le nombre de variables dans l'instruction READ.

## ***Out of memory***

- Le programme est trop grand, il dépasse la mémoire.  
Lorsque le programme est long, il faut toujours vérifier la mémoire disponible en faisant appel à la fonction FRE(X).

## ***Out of string space***

- La chaîne de caractères est insuffisante.

Augmenter la zone de la chaîne de caractères en utilisant la commande [CLEAR](#).

## ***Overflow***

Dépassement de capacité.

- Le résultat du calcul est supérieur à 1.7E38.
- Le PHC 25 est limité à 1.7E38.

## ***Return w/o gosub***

- S'agit-il d'une instruction [RETURN](#) sans instruction [GOSUB](#) ?
- Vérifiez la correspondance des deux instructions.

Exemple :

```
100 GOSUB 1000
...
1000 X=X+1
```

## ***Syntax error***

- La frappe a-t-elle été faite correctement ?
- Les noms variables sont-ils corrects ?

Exemple :

```
10 1X=2
```

## ***Tape read error***

- La cassette n'a pas été lue correctement.

## ***Too long string***

???

## ***Type mismatched***

- Les chaînes de caractères et les valeurs numériques sont-elles mélangées ?

## ***Undefined FN call***

- S'agit-il d'une utilisation d'une fonction FN qui n'est pas définie par l'instruction [DEFFN](#) ?

## ***Undefined line number***

- Le numéro de ligne désigné en [GOTO](#), [GOSUB](#), instruction [RESTORE](#) ou commande [RUN](#) n'existe pas.

## ***String too complex***

- La chaîne de caractères est trop compliquée.

### **3eme couverture**

Fin du livre.

***4<sup>ème</sup> de couverture à faire.***