

*Informations*

*complémentaires*



*Instructions concernant  
le clavier-écran*



## COMPLEMENTS CONCERNANT L'INSTRUCTION INPUT

- 1) Elle permet de lire une ou plusieurs données élémentaires composées au clavier et les affecte à une ou plusieurs variables spécifiées.

Exemple :

```
10 INPUT A,B,C,D,E,F,G
```

```
RUN
```

```
?13
```

```
??5
```

```
??20
```

```
??26
```

```
??12
```

```
??5
```

```
??18
```

Après exécution de ce programme :

```
A=13
```

```
B=5
```

```
C=20
```

```
D=26
```

```
E=12
```

```
F=5
```

```
G=18
```

- 2) L'instruction INPUT permet d'afficher un message devant le point d'interrogation (?)

Exemple :    1Ø INPUT "QUEL EST VOTRE PRENOM";N\$  
                  2Ø PRINT "BONJOUR";N\$  
                  RUN  
                  QUEL EST VOTRE PRENOM?DENISE  
                  BONJOUR DENISE

La syntaxe complète de l'instruction INPUT est donc la suivante :

INPUT "message";liste de variables

Les variables de la liste doivent être séparées par une virgule.

Exemple :

1Ø INPUT "CODE POSTAL PUIS VILLE";CP,V\$  
 RUN  
 CODE POSTAL PUIS VILLE?67200      
 ??STRASBOURG   

L'utilisateur pourra donner le code postal et la ville en une seule fois ; il suffit de les séparer par une virgule.

Avec l'exemple ci-dessus, il tapera :

67200, STRASBOURG

Attention :

Ceci implique qu'il ne faut pas taper de virgule dans un INPUT prévu pour une seule variable.

Exemple :

Pour le programme :

```
10 INPUT"ADRESSE";A$
```

```
RUN
```

```
?12,rue des Lilas
```

est interdit car

A\$ sera égal à 12.

## FONCTION SPC

Envoie un certain nombre d'espaces après le curseur sur l'écran ou l'imprimante.

SPC vient de la contraction en anglais SPACE qui veut dire espace (blanc).

Elle insère des blancs dans les instructions PRINT ou LPRINT

Syntaxe : SPC (expression numérique)

L'expression numérique est arrondie à l'entier le plus proche. Elle doit se trouver dans le domaine allant de 0 à 255 (pour éviter une erreur "Illegal function call")

```
Exemple :   10 PRINT "AU";  
            20 PRINT SPC(10);  
            30 PRINT "REVOIR"  
            RUN  
            AU          REVOIR
```



## FONCTION TAB

On obtient X espaces sur l'écran ou l'imprimante jusqu'à ce que le curseur soit la position X.

Elle déplace donc le curseur ou la tête d'impression vers une position spécifiée, dans les instructions PRINT ou LPRINT.

Syntaxe : TAB (expression numérique)

L'expression numérique est arrondie au plus proche.  
Elle doit se trouver dans le domaine allant de 0 à 255  
(pour éviter une erreur "Illegal Function Call").

1 est la limite à gauche, la largeur moins 1 est la limite à droite.

L'expression numérique spécifie donc la position du curseur.  
(ou de la tête d'impression) dans une ligne. C'est en fin de compte le numéro de la colonne.

### Exemple :

```
10 PRINT "JE";TAB(10);"VOUS"  
20 PRINT "SOUHAITE";TAB(10);"LA BIENVENUE"  
  
RUN  
  
JE           VOUS  
  
SOUHAITE LA BIENVENUE
```

## INSTRUCTION CLS

Il est souvent plus agréable de pouvoir démarrer l'exécution d'un programme avec un écran "propre".

L'instruction CLS permet d'effacer tous les caractères affichés à l'écran (sans altérer le contenu de la mémoire).

CLS vient de la contraction en anglais CLEAR SCREEN qui veut dire effacer l'écran.

Syntaxe : CLS

```
Exemple : 10 CLS
          20 PRINT "BONJOUR"
          30 GOTO 10
```

Ce petit programme permet de faire clignoter le mot BONJOUR à l'écran.

Remarquez que le clignotement est assez rapide.

Nous allons essayer de le ralentir par une petite temporisation.

Entre les instructions 20 et 30 nous allons "faire perdre" un peu de temps à l'ordinateur pour que le message BONJOUR reste affiché plus longtemps.

Comment le faire travailler pour rien ?

On fait, par exemple, une boucle FOR ; l'ordinateur perdra du temps.

On ajoute une ligne 25 :

```
25 FOR I=1 TO 300:NEXT I
```

Appréciez la différence !

Cette technique de temporisation est souvent utilisée lorsqu'on veut "faire attendre" un certain temps.

## CONSOLE

Permet de définir un écran dans l'écran.

Exemple : CONSOLE 4,5 permet de définir un écran à partir de la 5ème ligne sur une longueur de 5 lignes.  
(La première ligne a le numéro 0)  
Tapez par exemple le programme suivant :

```
5 CONSOLE 4,5
10 CLS
20 PRINT "BONJOUR"
RUN
Puis LIST
```

Vous remarquez que l'écran est réduit.

Pour revenir à l'état initial, tapez

```
CONSOLE 0, 16
```

Console désigne donc les limites de déplacement vers le bas et le haut de l'écran, la syntaxe est la suivante :

```
CONSOLE ligne de départ, longueur
```

L'écran a 16 lignes.

La ligne de départ est la ligne numéro 0 .

Exemple :

10 CONSOLE 0,16:CLS	→	efface les 16 lignes de l'écran
20 CONSOLE 4,6	→	définit l'écran à partir de la ligne 4 sur une longueur de 6 lignes
30 CLS	→	efface les lignes définies en 20
40 PRINT "BONJOUR"	→	affiche BONJOUR sur la 5ème ligne
50 CONSOLE 10,5	→	définit l'écran à partir de la ligne 10 sur une longueur de 5 lignes
60 CLS	→	efface les lignes définies en 50
70 PRINT "JE M'APPELLE PHC25"	→	affiche un message
80 GOTO 20	→	revient en 20

Autre exemple :

10 CONSOLE 0,16:CLS

efface tout l'écran

20 CONSOLE 4,2

définit l'écran à partir  
de la 5ème ligne sur une  
longueur de 2 lignes

30 CLS

40 PRINT "BONJOUR"

50 PRINT "ENCHANTE"

60 PRINT "DE VOUS"

70 PRINT "CONNAITRE"

80 GOTO 30

Vous noterez que l'affichage est très rapide, on va donc faire  
une temporisation entre chaque affichage en rajoutant les  
lignes suivantes :

45 GOSUB 100

55 GOSUB 100

65 GOSUB 100

75 GOSUB 100

100 FOR I=1 TO 500:NEXT:RETURN

## SCREEN

Lors de la mise sous tension, le PHC 25 vous pose la question suivante :

SCREEN ?

Il nous demande ici si nous voulons travailler avec une seule page écran ou avec 2.

Si nous répondons 1, la mémoire restante est de 14265 caractères.

Si nous répondons 2, la mémoire restante est de 8121 caractères.

En effet, nous pouvons choisir si nous désirons utiliser une partie de la mémoire en tant qu'écran ou en tant que mémoire pour les programmes.

Afin de bien visualiser cela, allumez le PHC 25, tapez 1.

Puis tapez sur  et  en même temps.

Il n'y a aucun effet.

Eteignez puis allumez à nouveau le PHC 25 et tapez 2.

Tapez sur  et  en même temps, vous verrez que s'affichera l'écran 2.

Vous pouvez taper des caractères, puis revenir à l'écran 1 par les touches  et .

Ainsi vous pouvez choisir le nombre d'écrans désirés lors de la mise sous tension.

A ne pas confondre avec l'instruction SCREEN que nous allons voir après le complément qui suit.

Complément :

En cours de programme, si vous ne désirez pas éteindre le PHC 25 et changer quand même le nombre de pages écran (ce qui correspond à la question posée à l'allumage "SCREEN?"), il faut appliquer le programme suivant :

Pour passer de 2 pages à 1 page :

POKE &HFB58,247

POKE &HFB56,1

CLEAR 50,&HF800

Pour passer de 1 page à 2 pages :

CLEAR 50,&HE00

POKE &HFB58,223

POKE &HFB56,20

L'instruction SCREEN permet de sélectionner une page écran suivant une définition (nombre de lignes X nombre de colonnes) voulue.

Syntaxe: SCREEN a, b, c.

Il y a 4 possibilités d'utilisation d'une page suivant le tableau suivant :

a	1	2	3	4
texte	16 lignes X 32 colonnes	16 lignes X 32 colonnes	16 lignes X 16 colonnes	16 lignes X 32 colonnes
définition:	0	64 X 48	128 X 192	256 X 192
graphique :		points	points	points

La valeur a fixe donc le nombre de caractères ou le nombre de points que l'on veut.

Si a=3, on s'aperçoit que le mode texte correspond à 16 lignes de 16 caractères, les caractères affichés seront donc 2 fois plus larges.

La valeur b désigne la page visible à l'écran.

Remarque : Evitez d'utiliser des graphiques sur la page 1, car bien qu'on puisse les afficher, ils seront détruits ; tous les dialogues se faisant par l'intermédiaire de la page 1.

La valeur c désigne la page sur laquelle on écrit.

Remarque : 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 ).



Exemples :

1Ø SCREEN 3,2,2

2Ø PRINT "BONJOUR"

Ce programme affichera BONJOUR en caractères élargis  
sur l'écran 2.

## LOCATE

Permet de positionner le curseur à une colonne et à une ligne donnée.

Syntaxe : LOCATE X,Y

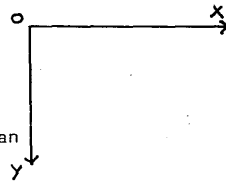
Exemple : LOCATE 5,10  
positionne le curseur à la 6ème colonne de la 11ème ligne.

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

$0 < X < 31$  pour le mode 1,2,4

$0 < X < 15$  pour le mode 3

$0 < Y < 15$  pour tous les modes écran



### Exemple :

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

### Remarque :

Si ce petit programme défile trop vite vous pouvez rajouter une temporisation :

```
35 FOR J=1 TO 100:NEXT
```

## FONCTION CSRLIN

Permet de connaître le numéro de ligne où se trouve le curseur.

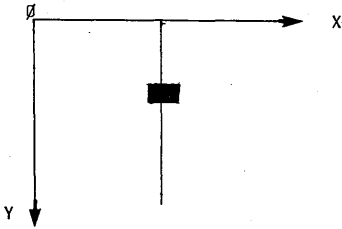
Vient de la contraction de l'anglais CURSOR LINE qui veut dire LIGNE CURSEUR.

Syntaxe : CSRLIN

Exemple : 1Ø LOCATE 12,7  
2Ø PRINT CSRLIN  
RUN  
7

## FONCTION POS

Donne la position de l'axe X du curseur.



Syntaxe : POS (nombre quelconque)

Exemple : 1Ø LOCATE 12,7

2Ø PRINT POS(X)

RUN

12

Remarque : LPOS est utilisée pour connaître la position pour l'imprimante.

POS est utilisée pour connaître la position pour l'écran.

## FONCTION SCRIN

Renvoie la valeur ASCII du caractère de coordonnées d'écran X,Y.

X étant le numéro de la colonne (la première = 0 )

Y étant le numéro de la ligne (la première = 0)

Syntaxe: SCRIN(X,Y)

Exemple : 5 CLS

```
10 PRINT "A"
```

```
20 PRINT SCRIN(0,0)
```

```
RUN
```

```
A
```

```
65
```

65 correspond bien au code ASCII de A majuscule.

Si on avait tapé le même programme avec a minuscule, on aurait obtenu la valeur 97.

## COLOR

Permet de désigner la couleur des caractères ou des graphiques affichés sur l'écran.

La couleur est désignée par une série de codes a, b, c.

Syntaxe : COLOR a,b,c

Même si on désigne les mêmes valeurs a, b, ou c, la couleur affichée peut être différente selon le mode écran choisi.

Le mode écran correspond au premier chiffre de l'instruction SCREEN, ce mode peut être égal à 1, 2, 3 ou 4.

### SCREEN 1 : Mode caractères

Dans ce mode, la couleur 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.

Il n'y a pas de graphique possible en mode 1.

a	b	c	Caractères
1		1	BLANC sur fond VERT
1		2	BLANC sur fond ORANGE
2		1	NOIR sur fond VERT
2		2	NOIR sur fond ORANGE

Exemple :

1Ø SCREEN 1, 1, 1

2Ø COLOR 1,, 2

3Ø CLS

4Ø PRINT "ABCD"

5Ø END

RUN

Remarque : Dans cet exemple COLOR 1,2 est identique à COLOR 1,1,2  
ou encore à COLOR 1,2,2

SCREEN 2 : Mode graphique (64 x 48)

Dans ce mode graphique, l'instruction COLOR est valable seulement pour les caractères et graphiques affichés sur l'écran après l'exécution de l'instruction. La couleur du graphique est déterminée par a et c. Neuf (9) couleurs peuvent être utilisées, mais seulement 4 couleurs pour les caractères.

La valeur de b affecte la couleur du fond d'écran.

a	b	c	caractère	fond	graphiques
0	0	1	vert	noir	noir
1	1	1	vert	vert	vert
2	2	1	vert (inversé)	jaune	jaune
3	3	1	orange	bleu	bleu
4	4	1	orange (inversé)	rouge	rouge
5	5	1	orange (inversé)	blanc	blanc
6	6	1	orange (inversé)	bleu clair	bleu clair
7	7	1	orange (inversé)	violet	violet
8	8	1	orange (inversé)	orange	orange

a	b	c	caractère	fond	graphiques
0	0	2	orange	noir	noir
1	1	2	orange	blanc	blanc
2	2	2	orange(inversé)	bleu clair	bleu clair
3	3	2	vert	violet	violet
4	4	2	vert (inversé)	orange	orange
5	5	2	vert (inversé)	vert	vert
6	6	2	vert (inversé)	jaune	jaune
7	7	2	vert (inversé)	bleu	bleu
8	8	2	vert (inversé)	rouge	rouge

Exemple :     10 SCREEN 2, 1, 1  
               20 COLOR 2,5,1  
               30 CLS  
               40 PRINT "a"  
               RUN

Dans cet exemple les caractères seront noir sur fond vert clair. Les graphiques seront vert clair sur fond blanc sachant que :

$1 \leq a \leq 8$ ,  $1 \leq b \leq 8$ ,  $1 \leq c \leq 2$ . Il existe 128 possibilités différentes.



SCREEN 3 : Mode graphique (128 x 192)

Dans ce mode graphique, l'instruction COLOR n'est valable que pour les caractères et graphiques affichés sur l'écran après l'exécution de ladite instruction.

a	b	c	caractère ou graphiques
1		1	vert
2		1	jaune
3		1	bleu
4		1	rouge
1		2	blanc
2		2	bleu clair
3		2	violet
4		2	orange

Exemple : 10 SCREEN 3, 1, 1  
 20 COLOR 2, 1, 1  
 30 CLS  
 RUN

La couleur de fond est déterminée par les valeurs de b et c comme dans le mode SCREEN 2, mais b est limité entre 0 et 4, ce qui procure 64 possibilités différentes.

SCREEN 4 : Mode graphique (256 x 192)

Dans ce mode graphique, l'instruction COLOR n'est valable que pour les caractères et graphiques affichés sur l'écran après l'exécution de ladite instruction.

a	b	c	caractères ou graphiques	FOND
1	0	1	VERT	NOIR
0	1	1	NOIR	VERT
1	0	2	BLANC	NOIR
0	1	2	NOIR	BLANC

NB. Dans le cas où l'on désire un changement de couleur de l'écran entier, faire la couleur b identique à a puis exécuter l'instruction COLOR et CLS.

## LINE

Permet de tracer une droite entre 2 points ou un rectangle dans une couleur déterminée.

Il y a possibilité de colorier la surface du rectangle.

Syntaxe :

LINE(X1,Y1)-(X2,Y2), couleur, BF

X1,Y1 : coordonnées du point de début de la ligne

X2,Y2 : coordonnées du point de fin de la ligne

couleur : c'est un numéro de couleur dans laquelle sera tracée la droite ou le rectangle (voir l'instruction COLOR)

La valeur par défaut est la valeur courante.

B : c'est un paramètre optionnel qui permet de tracer un rectangle (B vient de Box).  
La diagonale du rectangle est spécifiée par les coordonnées (X1,Y1) et (X2,Y2)

F : c'est un paramètre optionnel qui ne peut être utilisé que si B est également utilisé.  
(F vient de Filled qui veut dire plein).

"BF" trace un rectangle et le colorie avec la couleur sous-entendue ou avec celle indiquée dans le paramètre couleur.

Répondre 2 à la question SCREEN lors de la mise en marche, pour tester les exemples suivants.

Exemples :

```
5 SCREEN 4,2,2
10 LINE(5,5)-(20,20),3
```

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



Faire CTRL Q pour visualiser l'écran 2.

```
5 SCREEN 4,2,2
10 LINE(5,5)-(20,20),3,B
```

Les points 5,5 et 20,20 appartiennent à un carré.  
Ils sont les 2 points opposés.



```
5 SCREEN 4,2,2
10 LINE(5,5)-(20,20),3,BF
```

Les points 5,5 et 20,20 appartiennent à un carré de couleur 3.



```
5 SCREEN 4,2,2
10 LINE(10,10)-(50,10)
   trace une droite horizontale :
```



```
10 SCREEN 4,2,2
```

```
15 CLS
```

```
20 LINE(10,10)-(50,10)
```

```
30 LINE(50,10)-(50,60)
```

```
40 LINE(10,10)-(50,60)
```

Ce petit programme affiche un triangle.



Exemples :

Programmer des graphiques grâce à l'instruction LINE.

```

5  PI=3.141592654
10 ALPHA=PI:X1=-1:Y1=0
20  INPUT "COEFF.";K:DELTA=PI*K
30  SCREEN 4,2,2:CLS
40  FOR RAYO=90 TO 0 STEP-1
50  X2=COS(ALPHA):Y2=SIN(ALPHA)
60  LINE(125+X1*RAYO,90-Y1*RAYO)-(125+X2*RAYO,90-Y2*RAYO)
70  X1=X2:Y1=Y2
80  ALPHA=ALPHA+DELTA
90  NEXT RAYO
100 END

```

Vite, quelques explications ...

Ligne 5 : Initialisation de la constante PI ( $\pi$ )

Ligne 10 : Affectation des valeurs initiales pour l'angle de départ (ALPHA) et les coordonnées trigonométriques (X1 et Y1)

Ligne 20 : Lecture de l'angle de progression exprimé en radians.  
Calcul de DELTA (angle de progression) en radians : on effectue ce calcul car les fonctions SIN et COS n'acceptent que des angles en radians.

Ligne 30 : Effacement de l'écran 2 défini en haute résolution

Ligne 40 : Boucle faisant varier le rayon de 90 à 0

Ligne 50 : Calcul des nouvelles coordonnées X2 et Y2

Ligne 60 : Tracé de la ligne.  
Les constantes 125 et 90 permettent de centrer le dessin dans l'écran

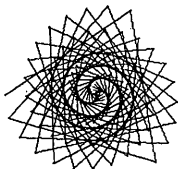
Ligne 70 : La fin de la ligne tracée devient le début de la suivante

Ligne 80 : On fait progresser l'angle ALPHA de la valeur DELTA

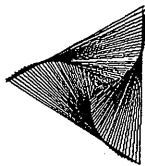
Ligne 90 : RAYON suivant

Ligne 100 : FIN

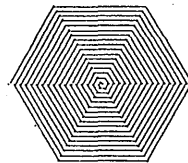
Exemples de graphiques obtenus avec le programme précédent.



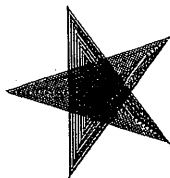
COEFF. = 0.64



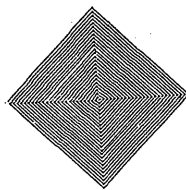
COEFF. = 0.67



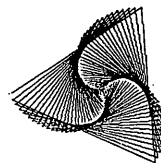
COEFF. = 0.333333



COEFF. = 0.8



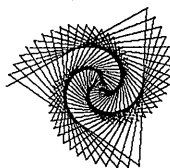
COEFF. = 0.5



COEFF. = 0.66



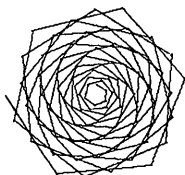
COEFF. = 0.99



COEFF. = 0.68



COEFF. = 0.1



COEFF. = 1.65

## PAINT

Permet de colorier une figure fermée, à partir du point le plus proche des coordonnées X,Y spécifiées.

PAINT (X,Y),couleurc,couleurp

Les coordonnées X et Y définissent un point se situant dans une zone de l'écran. Ce sont les coordonnées du point à partir duquel agit l'instruction PAINT. Cette zone est coloriée avec la couleur c sauf le périmètre de couleur P.

couleur c : c'est un numéro de couleur qui spécifie la couleur à utiliser pour peindre la fenêtre ou une portion fermée de celle-ci.

couleur p : c'est un numéro de couleur spécifiant la couleur à utiliser pour peindre la bordure de la figure fermée.

Remarque : Pour colorier (PAINT) une figure fermée pré-définie, assurez-vous que le point de coordonnées X,Y est à l'intérieur de la bordure de la figure.

S'il est à l'extérieur de cette bordure, seule la portion de la fenêtre qui se trouve à l'extérieur de la figure sera coloriée.



Exemple 1 :

Reprenons le programme qui affichait un triangle et colorions le.

```
10 SCREEN 4,2,2
15 CLS
20 LINE(10,10)-(50,10)
30 LINE(50,10)-(50,60)
40 LINE(10,10)-(50,60)
50 PAINT(49,11),1
```

L'instruction 50 colore l'intérieur du triangle à la couleur 1

Exemple 2 :

```
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
60 GOTO 60
```

Un cadre de couleur 2 (vert) est dessiné autour de l'écran, l'instruction 50 peint la zone d'écran auquel appartient le point 80, 80 de la couleur 4 (orange) sauf la zone de couleur 2 (vert).

Pour arrêter l'exécution du programme, faire

CTRL

et

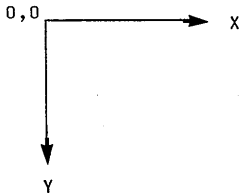
C

PRESET

PSET

PRESET : permet d'effacer le point de coordonnées (X,Y)  
sur l'écran.

X=0 et Y=0 correspond au point en haut, à gauche



Syntaxe : PRESET (X,Y)

PSET : permet d'afficher un point de coordonnées (X,Y) et  
de couleur c

Syntaxe : PSET (X,Y),C

Exemple : PSET (7,12),2  
Un point de couleur 2 est affiché aux coordonnées  
(7,12) de l'écran.

Attention : la couleur c dépend du screen 1, 2, 3, 4 et de  
la couleur choisie.

Il y a 256 possibilités en tout.

## FONCTION POINT

Permet d'obtenir le numéro de la couleur du point spécifié  
par les coordonnées (X,Y)

Syntaxe : POINT (X,Y)

Exemple : 10 PSET(7,30),3  
20 PRINT POINT(7,30)  
RUN  
3

## FONCTION INKEY\$

INKEY\$ restitue soit une chaîne d'un caractère contenant un caractère lu à partir du clavier, soit une chaîne nulle si aucun caractère n'a été appuyé.

Syntaxe : INKEY\$

Le caractère appuyé n'aura pas d'écho à l'écran, c'est-à-dire qu'il ne sera pas affiché à l'écran.

Exemple :     10 A\$=INKEY\$  
              20 PRINT A\$

Vous remarquez en exécutant ce programme, que INKEY\$ n'attend pas comme le INPUT.

Il faudra être très rapide pour arriver à frapper un caractère au moment précis où l'ordinateur exécute l'instruction 10 !

Pour améliorer le programme, on va faire attendre l'ordinateur tant qu'aucune touche ne sera appuyée :

```
10 A$=INKEY$
20 IF A$="" THEN GOTO 10
30 PRINT A$
```

Cette fonction peut être très intéressante pour faire des saisies contrôlées (exemple : saisie d'un caractère sans avoir à taper RETURN), pour faire plusieurs traitements en même temps sans bloquer le programme (l'instruction INPUT se mettant en attente et bloquant donc l'exécution), pour faire des jeux vidéo etc...

Un petit jeu pour illustrer une utilisation de la fonction  
INKEY\$.

```
5 CLS
10 PRINT "VOULEZ-VOUS LES REGLES DU JEU?"
20 A$=INKEY$
30 IF A$="O" THEN PRINT "OUI":GOTO 60
40 IF A$="N" THEN PRINT "NON":GOTO 100
50 GOTO 20
60 PRINT "CE JEU EST UN JEU DE REFLEXES"
70 PRINT "IL FAUT APPUYER SUR UNE TOUCHE AU MOMENT PRECIS"
80 PRINT "OU L'ETOILE S'AFFICHE EN DERNIERE COLONNE"
90 PRINT "APPUYEZ SUR UNE TOUCHE POUR DEMARRER"
95 IF A$="" THEN 95
100 CLS:PRINT TAB(31);":"
110 PRINT "*";:K=K+1:IF K > 33 THEN 140
120 A$=INKEY$:IF A$="" THEN 110
130 IF K=32 THEN PRINT "GAGNE":END
140 PRINT "PERDU"
```



*Instructions spécialisées*





## TOUCHES DE FONCTIONS

Lorsqu'on allume l'ordinateur, les touches F1, F2, F3, F4 sont préprogrammées.

Il suffit d'appuyer sur F1 par exemple pour afficher RUN et faire exécuter le programme qui est en mémoire.

Valeur initiale des touches :

① F1 RUN	⑤ Shift F1 COLOR
② F2 CLOAD"	⑥ Shift F2 CSAVE"
③ F3 PRINT	⑦ Shift F3 INPUT
④ F4 LIST	⑧ Shift F4 SCREEN

Ces 8 touches sont reprogrammables grâce à l'instruction KEY.

La syntaxe est la suivante :

KEY numéro de la touche, "TEXTE"

Le numéro de la touche peut prendre une valeur de 1 à 8.

Une touche peut contenir un texte de 8 caractères maximum.

Exemples :

KEY 1, "CONSOLE"

Lorsque vous taperez sur la touche F1, le message CONSOLE s'affichera.

KEY 4, "LIST"+CHR\$(13)

Lorsque vous taperez sur la touche F4, le message LIST s'affichera et provoquera le listing du programme en mémoire.

KEY 7, "CTON"+ CHR\$(13)

L'appui sur la touche

SHIFT

et

F3

mettra le moteur du magnétophone en marche.

## FONCTION DES TOUCHES DE CONTROLE

Un certain nombre de possibilités sont obtenues à l'aide de la touche CTRL (contrôle).

Contrôle C : l'exécution du programme est arrêtée

Contrôle G : avertissement sonore (Buzzer) avec synthétiseur

Contrôle I : tabulation - le curseur bouge vers la droite  
par groupe de 8 colonnes

Contrôle J : entrée de ligne

Contrôle L : effacement de l'écran

Contrôle M : retour chariot

Contrôle Q : changement de page sur l'écran

Contrôle ESC : même fonction que contrôle C

ESC : arrêt temporaire du programme.  
Pour remettre le programme en marche,  
appuyez sur ESC

## CONT

Cette commande permet de continuer l'exécution d'un programme après qu'il ait été arrêté par la commande   ou quand une instruction STOP a été rencontrée.

CONT vient de l'abréviation en anglais CONTINUE qui veut dire continuer.

Syntaxe : CONT

L'exécution reprend au point où elle s'était arrêtée.

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

Le message d'erreur "Can't continue" s'affiche (can't continue veut dire "je ne peux pas continuer")

Lorsque l'ordinateur rencontre   ou STOP, le message "BREAK IN" suivi d'un numéro de ligne s'affiche.

Le numéro de ligne correspond à la ligne que l'ordinateur était en train d'exécuter lorsqu'il s'est arrêté.

### Pour reprendre une exécution :

L'exécution peut reprendre à l'aide de CONT ou d'un GOTO immédiat, qui reprend l'exécution à partir d'un numéro de ligne spécifié.

Le contenu des variables reste indemne.

Par contre, en introduisant "RUN numéro de ligne" au lieu de "GOTO numéro de ligne", on efface toutes les variables du programme.

## DEF FN

Permet de définir une fonction et de lui donner un nom.

La syntaxe est la suivante :

DEF FN nom (liste de paramètres) = définition de la fonction

Exemple :

```
10 DEF FNA(X)=2*X+20
```

$\uparrow$     $\uparrow$     $\uparrow$   
 nom   paramètre   définition de la fonction

```
20 PRINT FNA(2)
```

Si on exécute ce programme, l'ordinateur affichera 24.

- Nom représente le nom de variable.

Le nom de la fonction est FN suivi du nom de variable.

Le premier caractère après FN doit être une lettre.

- La liste de paramètres comprend les variables qui seront remplacées par des valeurs réelles lors de l'appel de la fonction : les articles de cette liste sont séparés par des virgules

Exemple   DEF FNA1(X,Y)=2\*X+Y

- La définition de la fonction est une expression qui indique le calcul de la fonction. Les variables utilisées dans la définition sont celles citées dans la liste et éventuellement d'autres.

Attention :

L'instruction de définition DEF FN doit être rencontrée avant l'appel de la fonction, faute de quoi une erreur

UNDEFINED FN Call est signalée.

Une fonction définie par l'utilisateur peut être appelée par une autre fonction définie par l'utilisateur.

La fonction appelée doit être définie dans le même programme et précéder l'appel.

Par exemple :

10 DEF FNA(X)=(SIN(X)\*2)/180

20 DEF FNB(X)=FNA(X)\*0.5

## FONCTION TIME

TIME est un compteur qui s'incrémente automatiquement de 1 toutes les  $1/360$  secondes.

Syntaxe : TIME

Exemple :

```
5 CLS
10 LOCATE 10,10:PRINT TIME
20 LOCATE 10,11:PRINT TIME/360
30 GOTO 10
```

10 : affiche le temps en 360ème de secondes

20 : affiche le temps en secondes.

Remarque : Il ne peut y avoir de grande précision dans ce compteur.

## FONCTION FRE

Permet de connaître le nombre de caractères restant disponibles en mémoire.

Exemples :

```
PRINT FRE(X)
```

Affiche le nombre d'octets restant dans la mémoire que le BASIC n'utilise pas.

```
PRINT FRE(X$)
```

Affiche le nombre d'octets restant dans la zone de caractères.

## FONCTION RND

Cette fonction renvoie un nombre aléatoire compris entre 0 et 1.

Syntaxe : RND (expression numérique)

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

Si l'expression numérique est égale à 0, la fonction RND(0) répète le dernier nombre généré.

Exemple :

```

10 FOR I=1 TO 5
20 PRINT INT(RND(1)*100);
30 NEXT
RUN
25 6 69 89 24

```

Ce programme génère 5 nombres entiers aléatoires compris entre 1 et 100.



POKE  
PEEK

Avec les instructions vues jusqu'ici, nous ne pouvions travailler qu'avec des variables (A,A%,...) ou des valeurs constantes et l'ordinateur s'occupait tout seul de les ranger en mémoire.

Grâce aux instructions PEEK et POKE, nous pouvons lire ou écrire dans n'importe quelle case mémoire.

PEEK retourne l'octet (entier décimal de 0 à 255) lu à la position I de la mémoire.

Syntaxe :        PEEK (I)

Exemple : 10 FOR I=0 TO 100

20 PRINT PEEK(I)

30 NEXT I

Ce petit programme affiche le contenu des 100 premières cases mémoire.

POKE permet d'imposer une valeur à un octet dans la mémoire, c'est à dire de ranger un octet à une adresse désignée de la mémoire.

Syntaxe POKE I,J

L'expression I est l'adresse de l'octet, l'expression J la valeur qui doit y être placée.

CARTE MEMOIRE

	<u>Adresse hexadécimale</u>	<u>Adresse décimale</u>
ZONE DE TRAVAIL DE BASIC	FFFF	65535
PAGE VIDEO 2	F800	63488
ZONE DE TRAVAIL DU PROGRAMME	E000	57344
ESPACE LIBRE	C000	49152
ESPACE LIBRE	8000	32768
PAGE VIDEO 1	7800	30720
INTERPRETEUR BASIC	6000	24576
	0000	0

## CONTENU DU GENERATEUR DE CARACTERES

Poids fort	4									
	Poids faible	0	1	2	3	4	5	6	7	8
4	0		$\pi$		0	@	P		p	♠
	1		┐	!	1	A	Q	a	q	♥
	2		└	"	2	B	R	b	r	♣
	3		┌	#	3	C	S	c	s	♦
	4		└	§	4	D	T	d	t	○
	5		└	%	5	E	U	e	u	●
	6		└	&	6	F	V	f	v	
	7		└	'	7	G	W	g	w	
	8		┐	(	8	H	X	h	x	
	9		└	)	9	I	Y	i	y	
	A		└	*	:	J	Z	j	z	
	B		└	+	;	K	[	k	{	
	C		✕	,	<	L	¥	l		
	D			-	=	M	]	m	}	
	E			.	>	N	^	n	~	
	F			/	?	O	_	o		

Les valeurs sont données en hexadécimal.

Exemple :

```

10 FOR I=0 TO 200
20 POKE 24576+I,65
30 NEXT I

```

L'adresse 24576 est l'adresse de la 1ère case de l'écran.

On écrit le code 65 (caractère A) sur les 200 premières cases de l'écran.

On aurait pu écrire le programme suivant en donnant les codes en hexadécimal :

```

10 FOR I=0 TO 200
20 POKE &H6000+I,&H41
30 NEXT I

```

24276 en base 10 = 6000 en base 16

65 en base 10 = 41 en base 16  
(code qui correspond au caractère A).

Notez que l'exécution du programme est plus rapide lorsqu'on donne les codes en base 16.

Autre petit programme :

```

10 FOR I=0 TO 65535
20 A=PEEK(I)
30 IF A < 32 THEN PRINT " ";:GOTO 60
40 IF A > 127 THEN PRINT " ";:GOTO 60
50 PRINT CHR$(A)
60 NEXT I

```

Affiche le contenu de la mémoire

Remarque :

Les valeurs I et J peuvent être exprimées en décimal ou en hexadécimal.

Dans le cas, où on exprime les valeurs en hexadécimal, il faudra utiliser les symboles &H devant les valeurs.

Exemple :

POKE &H6710,&H81 : les valeurs sont données en hexadécimal

POKE 4053,24 : les valeurs sont données en décimal

L'opération inverse de POKE est PEEK.

Les opérations POKE et PEEK servent couramment pour une bonne utilisation de la mémoire, pour charger des sous-programmes écrits en assembleur et communiquer avec ces sous-programmes (paramètres et résultats).

## CLEAR

CLEAR permet de remettre à zéro toutes les variables numériques et d'annuler toutes les variables chaînes.

### Exemple :

```
10 A$="BONJOUR":X=100
20 PRINT A$,X
30 CLEAR
40 PRINT A$,X
```

Après l'instruction CLEAR, on voit bien que A\$ est une chaîne vide et que X=0.

- . CLEAR peut être utilisé comme une commande (en frappe directe) ou comme une instruction (dans un programme).
- . CLEAR peut être utilisé avec des arguments.

Syntaxe :        CLEAR I,J

I détermine la taille de la mémoire réservée pour y stocker des données.

J est optionnelle et détermine la limite supérieure de la mémoire utilisable sous BASIC.

J ne peut pas dépasser la valeur hexadécimale F800.

Nous vous proposons ci-après quelques manipulations devant permettre de mieux comprendre CLEAR.

Au départ, lorsqu'on allume l'ordinateur, la mémoire maximale disponible pour les variables est de 50 octets, le reste étant réservé pour le programme BASIC.

On peut visualiser cela de la manière suivante :

- allumez l'ordinateur et choisissez SCREEN 1.

L'ordinateur affiche que la mémoire totale (programme + données) est de 14265 octets

- Tapez le petit programme suivant :

```
10 PRINT FRE(0),FRE(X0)
```

N'oubliez pas que le programme prend déjà de la place en mémoire !

Faites RUN

L'ordinateur affiche : 14232 50  
ce qui signifie :

Il reste 14232 octets pour le programme BASIC et les variables numériques.

Il reste 50 octets pour les variables alphanumériques.

- Tapez directement

```
CLEAR 400
```

puis tapez RUN

Il ne reste plus que 13882 octets pour le programme BASIC et les variables numériques.

Il reste 400 octets pour les variables alphanumériques.

Tapez CLEAR 14182

Puis RUN

Nous avons maintenant davantage de mémoire disponible pour les données (14182 octets) que pour le programme BASIC (100 octets).

Si nous rajoutons à présent des lignes à notre programme BASIC, il restera encore moins d'octets en mémoire et s'affichera le message OUT OF MEMORY qui signifie "Il n'y a plus de mémoire pour le programme BASIC".

L'ordre NEW remettra la mémoire à l'état initial (14248 pour le programme et 50 pour les données).

Attention : le programme en mémoire sera perdu !

CLEAR suivi d'un nombre permet donc de réserver de la place pour les variables alphanumériques, au dépend de la mémoire programme.

A manipuler avec précaution : lorsque vous utilisez des variables alphanumériques dans les programmes, il faudra estimer la mémoire maximum nécessaire pour celles-ci.

Le message "out of string space" s'affiche s'il n'y a pas assez de place pour les variables alphanumériques.

"out of string space" signifie "Il n'y a plus de place pour les chaînes de caractères".

CLEAR suivi d'un 2ème argument permet de plus de déterminer la limite supérieure de la mémoire utilisable sous BASIC.

Le 2ème argument permet donc de protéger le haut de la mémoire afin de pouvoir y écrire un programme en langage machine non destructible par le BASIC.

Exemple :      CLEAR 400,&HE050

La taille de mémoire réservée aux variables alphanumériques est de 400 caractères (ou octets) et la limite supérieure de la mémoire est fixée à l'adresse hexadécimale E050.

On pourra écrire un programme en langage machine au-delà de l'adresse E050 puisque le programme BASIC ne dépassera pas cette valeur.

Remarque :

Le 2ème argument peut également être exprimé en décimal :

CLEAR 400, &HE050 peut s'écrire  
CLEAR 400,57424



## EXEC

L'instruction EXEC appelle un sous-programme en langage machine (microprocesseur Z80).

Syntaxe : EXEC &Hxxxx

xxxx est l'adresse exprimée en hexadécimal où ira le programme.

Cette adresse pourra également être donnée en décimal.

Le programme en BASIC ira à l'adresse indiquée par la variable xxxx (hexadécimal) pour exécuter un programme écrit en langage machine (hexadécimal).

Le programme en langage machine aura été écrit grâce à l'instruction POKE et devra se finir automatiquement par la valeur hexadécimale C9 qui correspond au code de l'instruction RET permettant le retour au langage BASIC.

Exemple 1 :

```

10 CLEAR 100,&HF000
20 J=0
30 INPUT "VALEUR HEXADECIMALE";A$
40 N=VAL("&H"+A$)
50 IF A$="FF" THEN 90
60 POKE &HF000+(J),N
70 J=J+1
80 GOTO 30
90 END

```

Ce programme permet de saisir un programme en langage machine Z80 et de l'implanter en mémoire à partir de l'adresse &HF000.

Exécutez ce programme en tapant RUN et donnez lui les valeurs suivantes :

3E 41 32 8F 60 C9 FF

L'écran doit afficher :

RUN	
VALEUR HEXADECIMALE?3E }	Met le caractère A dans
VALEUR HEXADECIMALE?41 }	l'accumulation du Z80
VALEUR HEXADECIMALE?32 }	
VALEUR HEXADECIMALE?8F }	Met A à l'adresse 608F (5ème ligne
VALEUR HEXADECIMALE?60 }	16ème colonne de l'écran)
VALEUR HEXADECIMALE?C9 }	RET (Retour au Basic)
VALEUR HEXADECIMALE?FF }	Fin

Le programme a été mis en mémoire, pour l'exécuter vous pouvez taper directement EXEC &HF000

&HF000 correspond à la case mémoire du début du programme machine.

Le caractère A doit s'afficher à la 5ème ligne, 16ème colonne de l'écran.

On peut également exécuter le programme en langage machine à partir d'un programme BASIC.

Tapez par exemple :

```
100 EXEC &HF000
```

puis RUN 100

Le caractère A doit également s'afficher.

Dans l'exemple 1, le programme en langage machine a été mis en mémoire par une saisie manuelle au clavier.

On peut également le stocker par les instructions READ DATA.

Exemple 2 :

```
10 CLEAR 100,&HF000
20 FOR K=1 TO 6
40 READ A$
50 POKE &HF000+(J-1),VAL("&H"+A$)
70 NEXT
80 DATA 3E,45,32,8F,60,C9
90 CLS
100 EXEC &HF000
120 PRINT "FIN"
```

On voit dans cet exemple que les datas servent d'instruction pour la commande EXEC.

Ne pas oublier de finir la liste de datas par le code C9 (RET).

Exemple 3 :

Ecrivons un programme en langage machine qui effectue l'effacement d'écran (équivalent de l'instruction CLS en BASIC).

Adresse mémoire	Langage machine	Langage Assembleur
F000	21 00 60	LD HL,ECRAN
F003	3E 20	LD A,&H20
F005	01 00 02	LD BC,&H0200
F008	77	B1 LD (HL),A
F009	23	INC HL
F00A	0D	DEC C
F00B	C2 08 F0	JP NZ,B1
F00E	10 F8	DJNZ B1
F010	C9	RET

Le programme BASIC chargeant ce programme machine en mémoire et l'exécutant s'écrit :

```

10 CLEAR 100,&HF000
30 FOR K=1 TO 17
40 READ A$
60 POKE &HF000+(K-1),VAL("&H")+A$
70 NEXT
80 DATA 21,00,60,3E,20,01,00,02,77,23,0D,C2,08,F0,
      10,F8,C9
100 EXEC &HF000
110 PRINT "FIN"

```

Explications :

Ligne 10 : protège le haut de la mémoire pour le programme en langage machine

Ligne 30 à 80 : Fait le programme en langage machine à partir de l'adresse hexadécimale F000

Ligne 100 : Exécute le programme se trouvant à l'adresse F000

Ligne 110 : Sans commentaire !!!

Faites RUN.

Remarquez la vitesse avec laquelle l'écran est effacé !

Ceci est encore plus net si vous exécutez uniquement le programme en langage machine en tapant directement

EXEC &HF000

Faites la comparaison entre la vitesse d'exécution de l'instruction CLS et EXEC &HF000!

Remarque :

Pour afficher des étoiles par exemple sur tout l'écran (à la place des blancs) tapez simplement :

POKE &H F004,&H2A  
EXEC &HF000

Conclusion :

Un programme en langage machine, pouvant être directement exécuté par le microprocesseur est beaucoup plus rapide à l'exécution qu'un programme en langage BASIC qui doit d'abord être interprété.

Cependant, pour faire un programme en langage machine, le programmeur doit connaître des notions supplémentaires qui ne font pas l'objet de ce manuel.

L'utilisateur expérimenté au SANYO devra se procurer un manuel traitant du langage assembleur Z80 pour en savoir plus à ce sujet.

## FONCTION USR

Exécute un programme fait par l'utilisateur en langage machine (microprocesseur Z80).

Syntaxe :            USR (expression numérique)

## FONCTION ASC

Cette fonction renvoie le code ASCII du premier caractère de la chaîne de caractères.

Syntaxe : ASC (chaîne de caractères)

Exemple :    10 A\$="TEST"  
             20 PRINT ASC(A\$)  
             RUN  
             116

## FONCTION CHR\$

C'est l'opposé de ASC : on obtient le caractère dont le code ASCII est le nombre.

Syntaxe : CHR\$ (nombre)

Exemple :    10 PRINT CHR\$(116)  
             RUN  
             T

## FONCTION INP

Obtient la lecture d'un octet sur le port I

Syntaxe : INP(I)

I doit être compris entre 0 et 255

Exemple : PRINT INP(&H8F)

## O U T

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

Syntaxe : OUT I,J

Exemple : OUT 32,100

OUT &H20,&H64

Ces deux exemples sont identifiés. Les valeurs I et J peuvent être exprimées en décimal ou en hexadécimal.

Dans ces exemples, la valeur hexadécimale 64 est envoyée sur le port 20 (hexadécimal).

Ces dernières instructions ne peuvent être utiles que pour les informaticiens confirmés car nécessitent des montages électroniques.



V - 60.

*Magnétophone*



## CTON

Permet de mettre en marche le magnétophone.

Attention : il faut que la touche PLAY soit enfoncée.

On arrive donc à commander le déroulement de la cassette à distance.

Syntaxe : CTON

### Remarque :

Cette instruction peut être intéressante lorsqu'on veut enregistrer ou lire un programme sur cassette.

Il n'est plus nécessaire de sortir la prise noire du magnétophone ; il suffit de taper CTON directement, puis de

presser sur la touche 

REW
-----

 du magnétophone pour rembobiner la cassette, par exemple.

## CTOFF

Permet de fermer l'interrupteur de commande à distance  
du magnétophone (remote control).

Syntaxe : CTOFF

On arrive ainsi à arrêter le défilement de la cassette

## INPUT#

Permet la lecture de données dans un fichier séquentiel et l'affectation des valeurs aux variables de la liste.

La syntaxe est la suivante :

INPUT# numéro de fichier, liste de variables

numéro de fichier = numéro associé au fichier au moment de son ouverture

numéro 0 correspond au clavier

numéro 1 correspond au magnétophone

La liste de variables contient le nom des variables où l'on doit affecter les données existantes sur le fichier.

Il faut que les types concordent.

**PRINT#**

Permet d'écrire des données dans un appareil désigné.

La sélection de l'appareil se fait par le numéro qui suit  
PRINT

Syntaxe : PRINT#-numéro, données

Les données ont la même structure que pour l'instruction PRINT.

:	:	:
:	Numéro de l'appareil	: Appareil sélectionné
:	:	:
:	:	:
:	Ø	: écran
:	:	:
:	1	: cassette du magnétophone
:	:	:
:	3	: imprimante
:	:	:
:	:	:

Exemple :

PRINT#-0,"BONJOUR"  
affiche BONJOUR à l'écran

PRINT#-1,A-  
La valeur de A est enregistrée sur la cassette du  
magnétophone.

## SLOAD

Permet de lire et d'afficher une image écran enregistrée sur cassette.

SLOAD vient de la contraction en anglais SCREEN LOAD qui veut dire chargement écran.

SLOAD peut servir de commande ou d'instruction.

Syntaxe : Elle est la même que pour LOAD : SLOAD"nom"

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 ; d'où l'intérêt d'utiliser l'instruction SCREEN.

N'oubliez pas d'appuyer sur la touche PLAY du magnétophone pour lire la cassette.

Exemple : SLOAD"ECRAN"

Attention : Pour l'ordinateur, ECRAN et écran représentent deux fichiers différents.

Lorsqu'on enregistre un programme avec un nom en minuscules, on ne pourra le recharger qu'en donnant le même nom en minuscules !

## SSAVE

Permet de sauvegarder une image écran complète sur cassette.

SSAVE vient de la contraction en anglais SCREEN SAVE qui veut dire enregistrer l'écran.

SSAVE peut servir de commande ou d'instruction.

Syntaxe : Elle est la même que pour SAVE :  
SSAVE"NOM"

N'oubliez pas de presser sur les touches REC et PLAY du magnétophone pour enregistrer sur cassette.

Exemple : SSAVE"ECRAN"

Attention : Lorsqu'on enregistre un programme avec un nom en minuscules, on ne pourra le recharger en mémoire, qu'en donnant le même nom en minuscules !

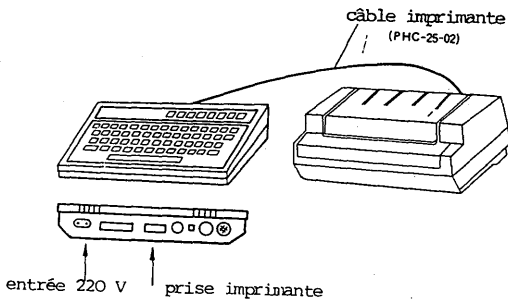


*Imprimante*



## CONNECTION AVEC L'IMPRIMANTE

Utiliser le câble spécial vendu en option (PHC 25-02) pour connecter l'imprimante au connecteur prévu à l'arrière de l'appareil.



### a) Data strobe

Il s'agit du signal de sortie fait vers l'imprimante et utilisé comme synchronisation lorsque les données sont envoyées vers l'imprimante.

### b) Data -07-

Il s'agit de la sortie des données du bus 8 bit parallèle envoyé vers l'imprimante.

### c) Ready

Le signal READY permet l'émission des instructions. Lorsque ce signal est à un niveau bas, il est possible d'envoyer des data vers l'imprimante.

### d) GND (Ground)

Prise de masse donnant le potentiel à tous les signaux.

# CONFIGURATION DU CABLE IMPRIMANTE

DATA STROBE	1 _____	1 DATA STROBE
DATA 0	2 _____	2 DATA 0
DATA 1	3 _____	3 DATA 1
DATA 2	4 _____	4 DATA 2
DATA 3	5 _____	5 DATA 3
DATA 4	6 _____	6 DATA 4
DATA 5	7 _____	7 DATA 5
DATA 6	8 _____	8 DATA 6
DATA 7	9 _____	9 DATA 7
	10 _____	10 ACKNOWLEDGE
READY	11 _____	11 READY
	12 _____	12
GND	13 _____	13
GND	14 _____	14
		15
		16
		17 GND
		18
		19 GND
		36

connecteur côté SANYO  
14 pins

connecteur côté imprimante  
36 pins

## LCOPY

Permet d'envoyer tout le contenu de l'écran vers l'imprimante.

Cette possibilité est souvent appelée "HARD COPY".

Attention : LCOPY ne marche que sur l'imprimante graphique.

## LPRINT

Permet d'envoyer des données, sur l'imprimante.

La syntaxe est la même que PRINT : la sortie se fait sur l'imprimante connectée à l'ordinateur.

Exemple :     LPRINT "BONJOUR"

Edite le mot BONJOUR sur l'imprimante.

## LPOS

Obtient la position de tête du pointeur de ligne dans le tampon de sortie, LPOS donne donc le nombre de caractères contenus dans la mémoire tampon associée à l'imprimante ligne.

Syntaxe : LPOS (argument fictif)


L'argument fictif est une expression numérique ou une chaîne quelconque.

La valeur renvoyée n'est pas affectée par la valeur de l'argument.

```
Exemple :      10 PRINT "PHC-25";
                20 PRINT LPOS(X)
                RUN
                6
```

LPOS correspond donc au numéro de colonne où se trouve positionnée la tête d'écriture de l'imprimante.

```
100 IF LPOS(X) > 60 THEN LPRINT CHR$(13)
```

LPRINT CHR\$(13)  impression d'un retour chariot :  
la tête d'impression passe à la  
ligne suivante.





*Synthétiseur*



## PLAY

L'instruction PLAY nous permet de jouer de la musique.

La musique est produite par le générateur de son (PSG-01 synthétizer vendu en option).

La syntaxe est la suivante :

PLAY "chaîne de caractères"

Dans la chaîne de caractères, on indique la lettre correspondant à la (ou les) note(s) que l'on veut jouer.

Les lettres A à G indiquent les sons suivants :



Exemple :

```
10 PLAY "cdefgab"
20 GOTO 10
```

Ce petit programme joue les notes DO RE MI FA SOL LA SI DO DO RE MI FA ... et ainsi de suite.

On va pouvoir améliorer notre musique grâce aux fonctions complémentaires du SANYO :

on peut régler la durée du son en donnant la lettre *l* suivie d'un chiffre X avant la note

Exemple :       10 PLAY "l10cdefgab"  
                 20 GOTO 10

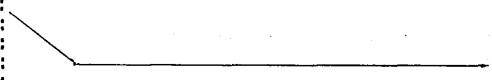

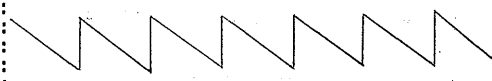

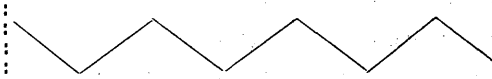
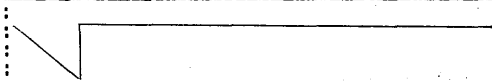
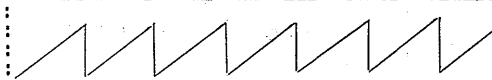
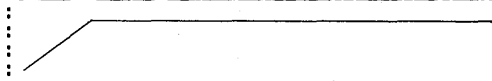
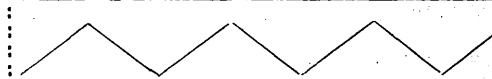
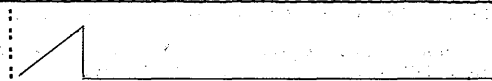
La durée du son doit être comprise entre 0 et 64.

Pour voir les différences entre les durées, on peut faire le programme suivant :

```
10 FOR I=1 TO 64
20 PLAY "l"+STR$(I)+"cdefgab"
30 NEXT I
```



# FORME DES SIGNAUX GENERES PAR LE SYNTHETISEUR MUSICAL

VALEUR DE X	FORME DE L'ENVELOPPE
0, 1, 2, 3	
4, 5, 6, 7	
8	
9	
10	
11	
12	
13	
14	
15	

Il est possible de jouer jusqu'à 3 notes en même temps :  
il suffit de séparer les chaînes de caractères contenant  
les morceaux à jouer par une virgule.

Exemple :

```
1Ø PLAY"cdefgab","bagfedc","a"  
2Ø GOTO 1Ø
```

## SOUND

L'instruction SOUND permet de produire un son avec le synthétiseur (synthesizer PSG-01).

Le générateur de son possède 14 registres numérotés de 0 à 13.

Suivant les valeurs de ces registres, le générateur produira un son différent.

La syntaxe de l'instruction SOUND est la suivante :  
SOUND numéro de registre, valeur

Le générateur de son possède 3 voies (A, B, C) ainsi qu'un générateur de bruit.

### 1) SON SIMPLE SUR 1 VOIE :

Pour sortir un son sur 1 voie, il faut donner le volume et la fréquence du son.

La fréquence nous détermine si le son est grave ou aigu.

. réglage du volume :

VOLUME DU CANAL	NUMERO DU REGISTRE	V A L E U R	
		Son simple	Son avec enveloppe
A	8	0 → 15	16
B	9	0 → 15	16
C	10	0 → 15	16

Le volume du canal A est déterminé par la valeur du registre numéro 8 (Pour B ce sera le registre 9, pour C ce sera le registre 10).

Le volume peut varier entre 0 et 15 pour un son simple.

Si le volume est égal à 16, cela signifie que l'on désire une enveloppe. Ce cas sera étudié plus loin (§ 3).

Exemple : SOUND 8, 9

On désire un volume 9 sur le canal A

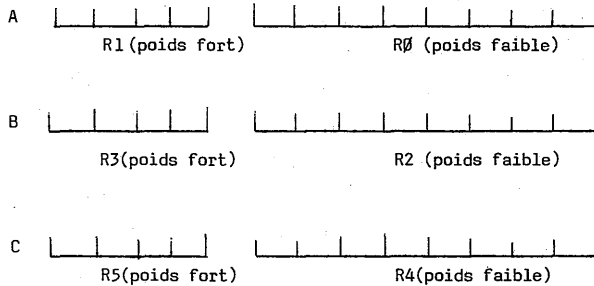


. réglage de la fréquence :

FREQUENCE DU CANAL	NUMERO DU REGISTRE	VALEUR
A	Ø	Ø → 255
	1	Ø → 15
B	2	Ø → 255
	3	Ø → 15
C	4	Ø → 255
	5	Ø → 15

Pour sortir un son d'une certaine fréquence sur le canal A, il faut charger la valeur de la fréquence dans les registres 1 et Ø.

La fréquence est représentée sur 12 bits :



Exemple :

SOUND 1,1:SOUND 0,2  
règle une fréquence sur le canal A(son aigu)

SOUND 3,15:SOUND 2,2  
règle une fréquence sur le canal B (son grave)

Attention : Nous avons uniquement réglé la fréquence, il  
faudra encore régler le volume sur le bon canal  
pour produire un son.

Pour produire un son de W Hz en canal A, voici la formule :

$P = 1.99675 \times 10^6 / 16 / X$   
P1=INT(P/256)  
R0=INT(P-R1\*256)  
SOUND 0,R0  
SOUND 1,R1

Exemple : Variation de la fréquence de 31 Hz à 12500 Hz :

5 SOUND 8.8  
10 FOR X=31 TO 12500  
20 P=1.99675 E6/16/X  
30 SOUND 1, INT(P/256)  
40 SOUND 0,P-INT(P/256)\*256  
50 NEXT X

Quelques exemples :

\* Variation de volume avec 1 fréquence constante sur le canal A :

```
10 FOR I=0 TO 15
20 SOUND 0,0:SOUND 1,1
30 REM REPRESENTE LE SON LE PLUS AIGU
40 SOUND 8,I
50 NEXT I
```

Faire SOUND 8,0 pour arrêter le volume ou CTRL C

Remarquez que c'est un peu rapide, on peut rajouter une temporisation :

```
45 FOR J=0 TO 500:NEXT
```

\* Variation de fréquence avec 1 volume constant sur le canal B :

```
5 SOUND 9,8
10 FOR I=0 TO 15
20 FOR J=0 TO 255
30 SOUND 8,I:SOUND 2,J
40 NEXT J:NEXT I
45 SOUND 9,0
```

On couvre dans cet exemple toute la gamme des fréquences, du plus aigu au plus grave.

## 2) SONS SIMPLES SUR 2 OU 3 VOIES

Remarquez que l'on a 3 voies, on peut donc sortir un ou plusieurs sons en même temps.

Il suffit de régler la fréquence et le volume sur chacune des voies.

### Exemple :

On veut sortir un son aigu sur la voie A et un son grave sur la voie B en même temps.

```
5 REM REGLAGE VOIE A
10 SOUND 8,8
20 SOUND 1,1:SOUND 0,0
25 REM REGLAGE VOIE B
30 SOUND 9,8
40 SOUND 3,15:SOUND 2,255
```

On pourrait rajouter les lignes suivantes :

```
45 REM REGLAGE VOIE C
50 SOUND 10,8
60 SOUND 5,10:SOUND 4,10
```

pour avoir un 3ème son sur la voie C

### 3) SON SIMPLE AVEC ENVELOPPE

Lorsque le volume du registre 8,9 ou 10 (correspondant au canal A, B ou C) est égal à 16, cela signifie qu'il y a présence d'une enveloppe.

Pour désigner une enveloppe sur 1 voie, il faut :

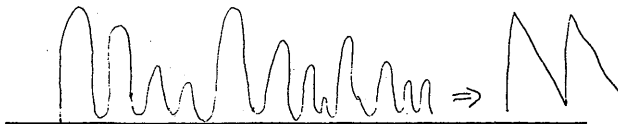
- . donner un volume égal à 16
- . donner la forme de l'enveloppe
- . donner la fréquence de l'enveloppe

Dans le cas de présence d'enveloppe, le volume est fixe et contrôlé par le générateur d'enveloppe.

Une enveloppe est une variation de volume (amplitude) automatique. On fait de la modulation d'amplitude.

Exemple de son sur le canal A avec présence d'une enveloppe n° 8.

période de  
l'enveloppe



période du son  
du canal A

forme de  
l'enveloppe n° 8

Rappel : La période T est inversement proportionnel à la fréquence F :

$$T = \frac{1}{F}$$

Dans ce cas on écrit :

SOUND 8,16 ← présence d'une enveloppe dans le canal A

SOUND 13,8 ← forme de l'enveloppe



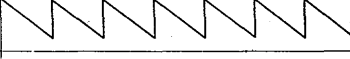
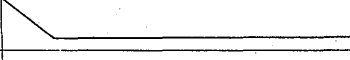
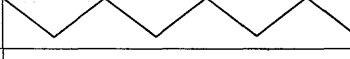
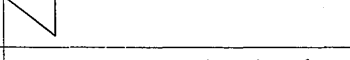
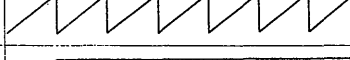
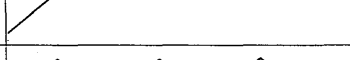
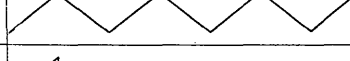
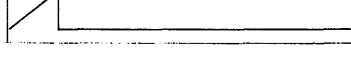
. Forme de l'enveloppe

La forme de l'enveloppe est donnée par la valeur du registre 13.

FORME DE L'ENVELOPPE DU CANAL	NUMERO DU REGISTRE	VALEUR
A, B ou C	13	0 → 15

Valeur de  
l'enveloppe

Forme de l'enveloppe

0, 1, 2, 3	
4, 5, 6, 7	
8	
9	
10	
11	
12	
13	
14	
15	



## . Fréquence de l'enveloppe :

La fréquence de l'enveloppe est donnée par la valeur des registres 11 et 12.

FREQUENCE DE L'ENVELOPPE DU CANAL	NUMERO DU REGISTRE	VALEUR
A , B ou C	11 (poids faible)	Ø → 255
	12 (poids fort)	Ø → 255

## Exemple :

On désire une enveloppe n° 8 dans le canal A à 1 fréquence fixe :

```

5 SOUND Ø,Ø:SOUND 1,1  ← fréquence du son
1Ø SOUND 8,16          ← présence d'une enveloppe
2Ø SOUND 13,8          ← forme de l'enveloppe
3Ø SOUND 11,1Ø:SOUND 12,1Ø ← fréquence de l'enveloppe

```

On pourrait faire varier la fréquence de l'enveloppe en modifiant la ligne 3Ø et rajouter :

```

3Ø FOR I=Ø TO 255
4Ø FOR J=Ø TO 255
5Ø SOUND 11,J:SOUND 12,I
6Ø NEXT
7Ø NEXT

```

Essayez de voir ce que donne le programme en changeant la forme de l'enveloppe

4) BRUIT

Le générateur de son possède également un générateur de bruit.

Le bruit est déterminé par sa fréquence dans le registre 6.

:	:	:	:
:	FREQUENCE DU BRUIT :	REGISTRE :	VALEUR :
:	:	:	:
:	A, B ou C :	6 :	0 → 31 :
:	:	:	:

Pour déterminer la fréquence du bruit, on écrira par exemple :

SOUND 6,10

Mais cela ne suffit pas, il faut déterminer sur quelle(s) voie(s) on veut sortir du bruit grâce à la valeur du registre 7.

5) MIXAGE

On peut mixer le bruit et le son grâce à la valeur du registre 7.

VALEUR	BRUIT
0	C B A
8	C B -
16	C - A
24	C - -
32	- B A
40	- B -
48	- - A
56	- - -

VALEUR	SON
0	C B A
1	C B -
2	C - A
3	C - -
4	- B A
5	- B -
6	- - A
7	- - -

Pour déterminer le mixage, il suffit de mettre dans le registre 7, la somme des valeurs bruit et son.

Exemple :

On désire un bruit uniquement dans le registre C et un son dans le registre A.

Bruit dans le registre C       $\Rightarrow$       valeur 24

Son dans le registre A       $\Rightarrow$       valeur 6

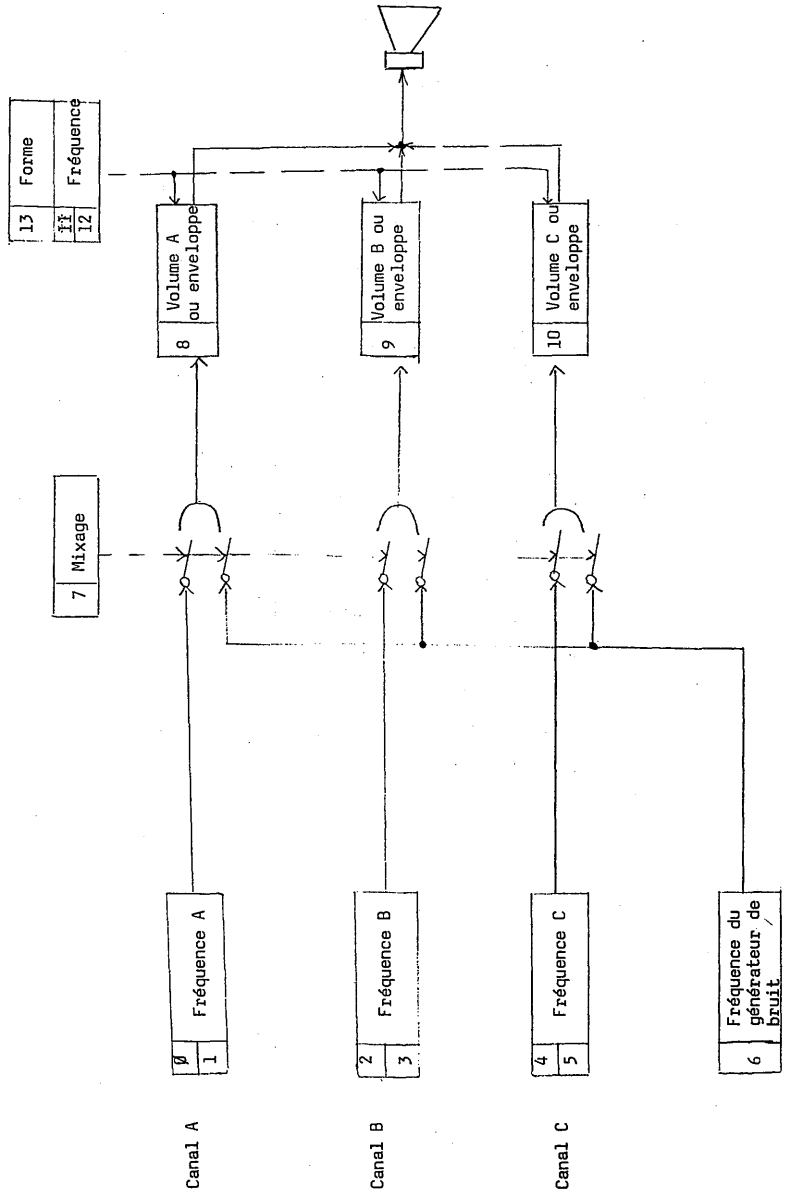
---

30

10 SOUND 7,30	}	← mixage
20 SOUND 8,8:		
30 SOUND 0,1:SOUND 1,1	}	← son dans le canal A
40 SOUND 10,8		
50 SOUND 6,10	}	← bruit dans le canal C

6) SCHEMA GENERAL

REGISTRE	FONCTIONS DU REGISTRE	VALEUR
0	{ Désigne la fréquence du son	0 255
1	{ du canal A	0 15
2	{ Désigne la fréquence du son	0 255
3	{ du canal B	0 15
4	{ Désigne la fréquence du son	0 255
5	{ du canal C	0 15
6	Désigne la fréquence du bruit	0 31
7	Mixage bruit et son	0 63
8	Désigne le volume du canal A	0 16
9	Désigne le volume du canal B	0 16
10	Désigne le volume du canal C	0 16
11	{ Désigne la fréquence	0 255
12	{ de l'enveloppe	0 255
13	Désigne la forme de l'enveloppe	0 15



Exemples :

```

150 REM *COUP DE FEU**
155 CLS:LOCATE10,10:PRINT"C O U P   D E   F E U"
200 SOUND6,1:SOUND7,28
220 FORL=15T00STEP-1
230 SOUND10,L:FORM=1T09:NEXT
240 NEXT:SOUND7,60:SOUND10,0
245 FORI=0T0500:NEXT
250 REM *BOMBE *****
255 CLS:LOCATE10,10:PRINT"B O M B E"
300 SOUND8,10:SOUND6,1:SOUND7,28
310 FORL=50T0100
320 SOUND1,INT(L/256):SOUND0,L-256*INT(L/256)
330 FORM=1T07:NEXTM
340 NEXTL
350 SOUND8,0:FORL=15T00STEP-.05
360 SOUND10,L:NEXT:SOUND10,0
400 REM ****vagues
405 CLS:LOCATE10,10:PRINT"V A G U E S"
410 SOUND7,28:SOUND6,15
420 FORL=1T03
430 D=INT(RND(1)*5)*3+30
440 FORM=1T012
450 SOUND10,M
460 FORN=1T00STEP.3
470 NEXTN
480 NEXTM
490 FORM=12T01STEP-1
500 SOUND10,M
510 FORN=1T00
520 NEXTN
530 NEXTM
540 NEXTL
550 SOUND10,0
600 REM ***cui-cui des oiseaux
605 CLS:LOCATE10,10:PRINT"C U I - C U I "
610 SOUND8,10
620 FORL=1T020
630 FORM=0T045+INT(RND(1)*10)STEP1.5
640 SOUND0,M:SOUND1,0
650 NEXTM
660 SOUND0,0:SOUND1,0
670 FORM=0T0INT(RND(1)*70)+20
680 NEXTM
690 SOUND8,8:NEXTL
800 REM ***ORDINATEUR      EN FOLIE
805 CLS:LOCATE10,10:PRINT"O R D I N A T E U R "
806 LOCATE10,12:PRINT"E N   F O L I E"
810 SOUND8,8:SOUND7,62
820 FORL=1T0100
830 SOUND0,INT(RND(1)*128)+128
840 SOUND1,1
850 FORM=1T010:NEXT
860 NEXTL
870 SOUND8,0

```

```
900 REM ***TELEPHONE OCCUPE
905 CLS:LOCATE10,10:PRINT"TELEPHONE OCCUPE"
920 FORL=1TO10
940 FORM=1TO80:NEXTM
950 SOUNDS,0:SOUNDS,8
960 SOUND0,200:SOUND1,0
970 FORM=1TO200:NEXTM
980 SOUNDS,0
990 NEXTL
995 SOUNDS,0
1000 REM ***RAYON LASER
1010 CLS:LOCATE10,10:PRINT"RAYON LASER"
1020 SOUNDS,8
1030 FORL=1TO5
1040 FORM=0TO160STEP10
1050 SOUND0,M:SOUND1,0
1060 NEXTM
1070 FORM=160TO0STEP-10
1080 SOUND0,M:SOUND1,0
1090 NEXTM
1100 SOUNDS,0
1105 FORI=1TO1000:NEXT:SOUNDS,8
1110 NEXTL
1120 SOUNDS,0
```



# **JOYSTICK**

*(manette de jeux)*



## FONCTION STICK

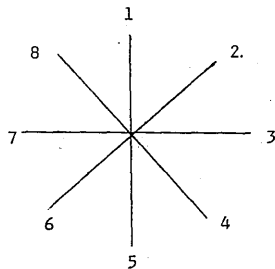
Obtient la fonction du contrôle de jeux par une manette de jeux (JOYSTICK).

On peut tester les touches du clavier ou le manche à balai connecté sur l'entrée 1 ou 2 du synthétiseur.

Syntaxe : STICK(I)

I=0 désigne les touches du clavier  
I=1 désigne le joystick 1  
I=2 désigne le joystick 2

Les directions du joystick 1 et 2 sont données par le dessin suivant :



Pour visualiser cela, branchez le joystick 1 et exécutez le programme suivant en tournant la manette de jeu :

```
10 PRINT STICK(1);GOTO 10
```

On peut également utiliser les touches du clavier.

Exécutez le programme suivant et pressez sur les touches  
fléchées ( ← → ↑ ↓ ) du clavier.

```
10 PRINT STICK(0);:GOTO 10
```

Exemple 1 : Le manche à balai est connecté sur la prise 1  
du synthétiseur

```
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"
100 GOTO 10
```

La manette de jeu indiquera les points cardinaux.

Remarque concernant l'exemple 1 :

On aurait pu optimiser ce programme en utilisant un tableau T\$ et en le chargeant avec READ DATA

```
5 DIM T$(9)
10 FOR I=1 TO 9
20 READ T$(I)
30 NEXT I
40 DATA "NORD","NORD-EST","EST","SUD-EST","SUD","SUD-OUEST",
  "OUEST"
50 DATA "NORD-OUEST","MANETTE INACTIVE"
60 PRINT T$(STICK(1))
70 GOTO 60
```

Faites des dessins avec le joystick !

Exemple 2 : la manette de jeu est connectée sur la prise 2 du synthétiseur.

```

100 SCREEN 3,1,1
102 COLOR 3,2,1
110 CLS:L=81:C=128
140 PSET(E,L),3
150 A=STICK(1)
160 IF A=1 THEN L=L-1:GOTO 140
165 IF A=2 THEN 1000
170 IF A=3 THEN C=C+1:GOTO 140
180 IF A=4 THEN 1010
190 IF A=5 THEN L=L+1:GOTO 140
200 IF A=6 THEN 1020
210 IF A=7 THEN C=C-1:GOTO 140
220 IF A=8 THEN 1030
230 GOTO 150

1000 L=L-1:C=C+1:GOTO 140
1010 L=L+1:C=C+1:GOTO 140
1020 L=L+1:C=C-1:GOTO 140
1030 L=L-1:C=C-1:GOTO 140

```

Exemple 3 :

Faites de la musique avec le joystick 1

```
5 A=STICK(1)
10 IF A=0 THEN 5
20 PLAY MID$( "cdefgabc",A,1)
30 GOTO 5
```

## FONCTION STRIG

Permet d'obtenir le contrôle du bouton poussoir rouge de commande du joystick (manche à balai).

On peut tester la touche ESPACE du clavier ou le bouton poussoir rouge du manche à balai connectée sur l'entrée 1 ou 2 du synthétiseur.

Syntaxe : STRIG(I)

I=0 désigne la touche ESPACE du clavier

I=1 désigne le joystick 1

I=2 désigne le joystick 2

Lorsque l'on appuie sur la gâchette, la valeur de STRIG=1  
sinon STRIG=0

Exemple :      10 IF STRIG(1)=1 THEN PLAY"ccc"  
                 20 GOTO 10

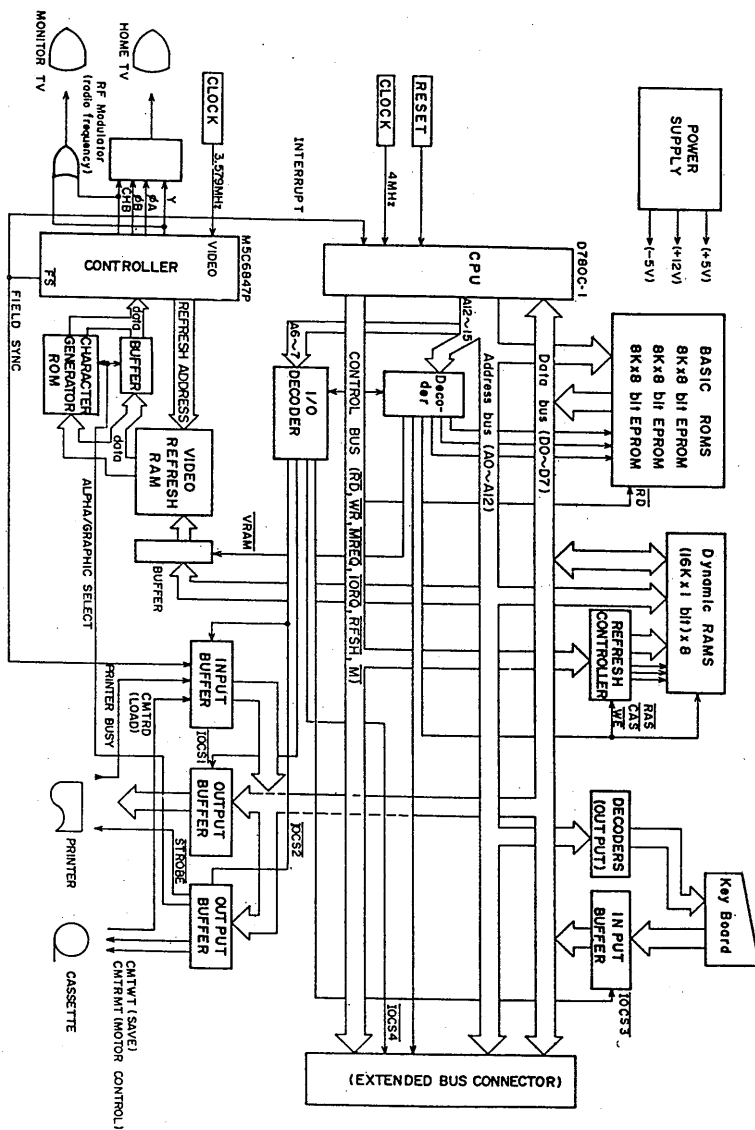


*Annexes*



## SCHEMA SYNOPTIQUE DU PHC 25

## DIAGRAMME

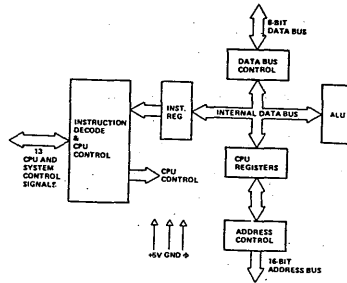




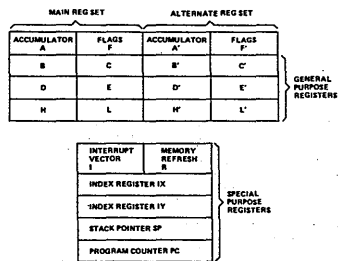
LISTE DES INSTRUCTIONS CONCERNANT

L'ASSEMBLEUR Z80





Z80-CPU BLOCK DIAGRAM



Z80-CPU REGISTER CONFIGURATION

# 8-BIT LOAD GROUP

Mnemonic	Symbolic Operation	Flags							Op-Code			No. of Bytes	No. of Cycles	Status	Comments	
		S	Z	N	P/V	M	C	76	543	210	Hex					
LD r, s	r ← s	*	*	X	*	X	*	01	r	s		1	1	4	r, s	Reg.
LD r, n	r ← n	*	*	X	*	X	*	00	r	110		2	2	7	000	B
		*	*	X	*	X	*	—	n	—					001	C
LD r, (HL)	r ← (HL)	*	*	X	*	X	*	01	r	110		1	2	7	010	D
LD r, (IX+d)	r ← (IX+d)	*	*	X	*	X	*	11	011	101	DD	3	5	19	011	E
		*	*	X	*	X	*	01	r	110					100	H
		*	*	X	*	X	*	—	d	—					101	L
LD r, (IY+d)	r ← (IY+d)	*	*	X	*	X	*	11	111	101	FD	3	5	19	111	A
		*	*	X	*	X	*	01	r	110						
		*	*	X	*	X	*	—	d	—						
LD (HL), r	(HL) ← r	*	*	X	*	X	*	01	110	r		1	2	7		
LD (IX+d), r	(IX+d) ← r	*	*	X	*	X	*	11	011	101	DD	3	5	19		
		*	*	X	*	X	*	01	110	r						
		*	*	X	*	X	*	—	d	—						
LD (IY+d), r	(IY+d) ← r	*	*	X	*	X	*	11	111	101	FD	3	5	19		
		*	*	X	*	X	*	01	110	r						
		*	*	X	*	X	*	—	d	—						
LD (HL), n	(HL) ← n	*	*	X	*	X	*	00	110	110		36	2	3	10	
		*	*	X	*	X	*	—	n	—						
LD (IX+d), n	(IX+d) ← n	*	*	X	*	X	*	11	011	101	DD	4	5	19		
		*	*	X	*	X	*	00	110	110						
		*	*	X	*	X	*	—	d	—						
		*	*	X	*	X	*	—	n	—						
LD (IY+d), n	(IY+d) ← n	*	*	X	*	X	*	11	111	101	FD	4	5	19		
		*	*	X	*	X	*	00	110	110						
		*	*	X	*	X	*	—	d	—						
		*	*	X	*	X	*	—	n	—						
LD A, (BC)	A ← (BC)	*	*	X	*	X	*	00	001	010	0A	1	2	7		
LD A, (DE)	A ← (DE)	*	*	X	*	X	*	00	011	010	1A	1	2	7		
LD A, (mn)	A ← (mn)	*	*	X	*	X	*	00	111	010	3A	3	4	13		
		*	*	X	*	X	*	—	n	—						
		*	*	X	*	X	*	—	n	—						
LD (BC), A	(BC) ← A	*	*	X	*	X	*	00	000	010	02	1	2	7		
LD (DE), A	(DE) ← A	*	*	X	*	X	*	00	010	010	12	1	2	7		
LD (mn), A	(mn) ← A	*	*	X	*	X	*	00	110	010	32	3	4	13		
		*	*	X	*	X	*	—	n	—						
		*	*	X	*	X	*	—	n	—						
LD A, I	A ← I			X	0	X	IFF	0	11	101	101	ED	2	2	9	
				X	0	X	IFF	0	01	010	111	57				
LD A, R	A ← R			X	0	X	IFF	0	11	101	101	ED	2	2	9	
				X	0	X	IFF	0	01	011	111	5F				
LD I, A	I ← A	*	*	X	*	X	*	11	101	101	ED	2	2	9		
		*	*	X	*	X	*	01	000	111	47					
		*	*	X	*	X	*	11	101	101	ED	2	2	9		
LD R, A	R ← A	*	*	X	*	X	*	01	001	111	4F					
		*	*	X	*	X	*	11	101	101	4E					

Notes: 1. s means any of the registers A, B, C, D, E, H, L  
IFF the content of the interrupt enable flip-flop (IFF) is copied into the P/V flag

Flag Notation: \* = Flag not affected, 0 = Flag reset, 1 = Flag set, X = Flag is unknown,  
I = Flag is affected according to the result of the operation.

V - 107.

## 8 BIT LOAD GROUP

		SOURCE																TEST ADDRESS	INDEX
		REGISTER								REG. ADDRESS				INDEXED					
		IMPLIED		A		B		C		D		E		H		L		REG. ADDRESS	INDEXED
		I	R	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX)	(IY)	(SP)	(PC)		
REGISTER	A	ED	ED	7F	7B	79	7A	7B	7C	7D	7E	DA	1A	DD	FD	DA	3A	05	n
	B			47	40	41	42	43	44	45	46			DD	FD	4A	3E		
	C			4F	48	49	4A	4B	4C	4D	4E			DD	FD	4E	3E		
	D			57	50	51	52	53	54	55	56			DD	FD	5A	3E		
	E			5F	58	59	5A	5B	5C	5D	5E			DD	FD	5E	3E		
	H			67	60	61	62	63	64	65	66			DD	FD	6A	3E		
	L			6F	68	69	6A	6B	6C	6D	6E			DD	FD	6E	3E		
DESTINATION	(HL)			70	71	72	73	74	75	76	77							06	n
	(BC)			02															
	(DE)			12															
INDEXED	(IX+d)			DD	DD	DD	DD	DD	DD	DD	DD							07	n
	(IY+d)			FD	FD	FD	FD	FD	FD	FD	FD								
EXT. ADDR	(mn)			32														08	n
	I			ED															
IMPLIED	R			ED														09	n
				ED															



# 16-BIT LOAD GROUP

V -108.

Mnemonic	Synthetic Operation	S	Z	N	V	H	C	Op-Code	No. of Bytes	No. of M- Cycles	No. of T States	Comments	
LD dd, nn	dd ← nn	*	*	X	*	X	*	00 d00 001 - n - - n -	3	3	10	dd Par 00 BC 01 DE 10 HL 11 SP	
LD IX, nn	IX ← nn	*	*	X	*	X	*	11 011 101 00 100 001 - n - - n -	4	4	14		
LD IV, nn	IV ← nn	*	*	X	*	X	*	11 111 101 00 100 001 - n - - n -	4	4	14		
LD HL, (nn)	H ← (nn+1) L ← (nn)	*	*	X	*	X	*	00 101 010 - n - - n -	3	5	16		
LD dd, (nn)	ddH ← (nn+1) ddL ← (nn)	*	*	X	*	X	*	11 101 101 01 d01 011 - n - - n -	4	6	20		
LD IX, (nn)	IXH ← (nn+1) IXL ← (nn)	*	*	X	*	X	*	11 011 101 00 101 010 - n - - n -	4	6	20		
LD IV, (nn)	IVH ← (nn+1) IVL ← (nn)	*	*	X	*	X	*	11 111 101 00 101 010 - n - - n -	4	6	20		
LD (nn), HL	(nn+1) ← H (nn) ← L	*	*	X	*	X	*	00 100 010 - n - - n -	3	5	16		
LD (nn), dd	(nn+1) ← ddH (nn) ← ddL	*	*	X	*	X	*	11 101 101 01 d00 011 - n - - n -	4	6	20		
LD (nn), IX	(nn+1) ← IXH (nn) ← IXL	*	*	X	*	X	*	11 011 101 00 100 010 - n - - n -	4	6	20		
LD (nn), IV	(nn+1) ← IVH (nn) ← IVL	*	*	X	*	X	*	11 111 101 00 100 010 - n - - n -	4	6	20		
LD SP, HL	SP ← HL	*	*	X	*	X	*	11 111 001 - n - - n -	F9	1	8		
LD SP, IX	SP ← IX	*	*	X	*	X	*	11 011 101 - n - - n -	D0	2	10		
LD SP, IV	SP ← IV	*	*	X	*	X	*	11 111 001 - n - - n -	F9	2	10		
PUSH qq	(SP-2) ← qqL (SP-1) ← qqH	*	*	X	*	X	*	11 000 101 - n - - n -	F9	1	3	11	qq Par 00 BC 01 DE 10 HL 11 AF
PUSH IX	(SP-2) ← IXL (SP-1) ← IXH	*	*	X	*	X	*	11 011 101 - n - - n -	D1	2	4	15	
PUSH IV	(SP-2) ← IVL (SP-1) ← IVH	*	*	X	*	X	*	11 111 101 - n - - n -	F0	2	4	1F	
POP qq	qqH ← (SP+1) qqL ← (SP)	*	*	X	*	X	*	11 000 001 - n - - n -	F9	1	3	10	
POP IX	IXH ← (SP+1) IXL ← (SP)	*	*	X	*	X	*	11 011 101 - n - - n -	D0	2	4	14	
POP IV	IVH ← (SP+1) IVL ← (SP)	*	*	X	*	X	*	11 111 101 - n - - n -	F0	2	4	14	

Notes: dd is any of the register pairs BC, DE, HL, SP  
qq is any of the register pairs AF, BC, DE, HL  
(PAIR)H, (PAIR)L refer to high order and low order eight bits of the register pair respectively.  
e.g. BC<sub>H</sub> = C, AF<sub>L</sub> = A

Flag Notations: \* = Flag not affected, 0 = Flag reset, 1 = Flag set, X = Flag unknown,  
| Flag is affected according to the result of the operation.

## 16 BIT LOAD GROUP 'LD' 'PUSH' AND 'POP'

		SOURCE								RMT. EXT.	EXT. ADDR.	REG. IMPL.
		REGISTER										
		AF	BC	DE	HL	SP	IX	IV	nn	(nn)	(PI)	
DESTINATION	REGISTER	AF										F1
		BC							0F n	ED n	ED n	E1
		DE							1F n	ED n	ED n	D1
		HL							2F n	2A n	2A n	E1
		SP				F9		D0 F9	2F n	ED n	ED n	
		IX							2D n	2A n	2A n	D0 E1
		IV							2D n	2A n	2A n	F0 E1
EXT. ADDR.		(nn)	ED n	ED n	2F n	ED n	ED n	ED n	ED n	ED n		
PUSH INSTRUCTIONS → REG. IMPL.		(PI)	0F n	0F n	0F n	0F n	0F n	0F n	0F n	0F n		

NOTE: The Push & Pop Instructions adjust the SP after every execution

POP INSTRUCTIONS

# EXCHANGE GROUP AND BLOCK TRANSFER AND SEARCH GROUP

V - 109.

Mnemonic	Symbolic Operands	Flags					Op-Code		No. of Bytes	No. of Cycles	No. of Status	Comments
		S	Z	N	P/V	C	76	543				
EX DE, HL	DE ← HL	*	*	X	X	*	11	101	011	E8	1	4
EX AF, AP	AF ← AP	*	*	X	X	*	00	001	000	08	1	4
EXX	BC ← BC' DE ← DE' HL ← HL'	*	*	X	X	*	11	011	001	D9	1	4
EX (SP), HL	H ← (SP+1) L ← (SP)	*	*	X	X	*	11	100	011	E3	-1	5
EX (SP), IX	IX ← (SP+1) IX ← (SP)	*	*	X	X	*	11	011	101	D0	2	6
EX (SP), IV	IV ← (SP+1) IV ← (SP)	*	*	X	X	*	11	111	101	F0	2	6
LDI	(DE) ← HL DE ← DE+1 HL ← HL+1 BC ← BC-1	*	*	X	0	X	1	0	11	101	011	ED
LDIR	(DE) ← HL DE ← DE+1 HL ← HL+1 BC ← BC-1 Repeat until BC = 0	*	*	X	0	X	0	0	11	101	101	ED
LDD	(DE) ← HL DE ← DE+1 HL ← HL+1 BC ← BC-1	*	*	X	0	X	1	0	11	101	101	ED
DDR	(DE) ← HL DE ← DE+1 HL ← HL+1 BC ← BC-1 Repeat until BC = 0	*	*	X	0	X	0	0	11	101	100	ED
CPI	A ← (HL) HL ← HL+1 BC ← BC-1	1	1	X	1	X	1	1	11	101	101	ED
CPH	A ← (HL) HL ← HL+1 BC ← BC-1 Repeat until A = (HL) or BC = 0	1	1	X	1	X	1	1	11	101	101	ED
CPD	A ← (HL) HL ← HL+1 BC ← BC-1	1	1	X	1	X	1	1	11	101	101	ED
CPDR	A ← (HL) HL ← HL+1 BC ← BC-1 Repeat until A = (HL) or BC = 0	1	1	X	1	X	1	1	11	101	101	ED

IMPLIED ADDRESSING					
	AF	BC, DE & HL	HL	IX	IV
AF					
BC, DE & HL					
DE					
REG. INDIR.	(SP)				

EXCHANGES 'EX' AND 'EXX'

Notes: ① P/V flag is 0 if the result of BC-1 = 0, otherwise P/V = 1  
 ② Z flag is 1 if A = (HL), otherwise Z = 0.  
 Flag Notation: \* = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,  
 1 = flag is affected according to the result of the operation.

		SOURCE	
		REG. INDIR.	(HL)
		ED	AD
		"LDI" - Load (DE) ← (HL) Inc HL & DE, Dec BC	
		ED	BD
		"LDIR" - Load (DE) ← (HL) Inc HL & DE, Dec BC, Repeat until BC = 0	
		ED	AS
		"LDD" - Load (DE) ← (HL) Dec HL & DE, Dec BC	
		ED	BS
		"DDR" - Load (DE) ← (HL) Dec HL & DE, Dec BC, Repeat until BC = 0	

Reg HL points to source  
Reg DE points to destination  
Reg BC is byte counter

BLOCK TRANSFER GROUP

SOURCE LOCATION		REG. INDIR.	(HL)
ED A1			
ED B1			'CPIR' Inc HL, Dec BC repeat until BC = 0 or find match
ED A8			'CPD' Dec HL & BC
ED B8			'CPDR' Dec HL & BC Repeat until BC = 0 or find match

HL points to location in memory  
to be compared with accumulator  
contents  
BC is byte counter

BLOCK SEARCH GROUP

# 8-BIT ARITHMETIC AND LOGICAL GROUP

Mnemonic	Symbolic Operation	S	Z	Flags								Op Code		No. of Hex	Bytes	Cycles	Status	Comments
				C	P/V	N	O	C	TS	643	210							
ADD A, r	A ← A + r	1	1	X	1	X	V	0	1	10	0001	r	1	1	4		r	Reg.
ADD A, s	A ← A + s	1	1	X	1	X	V	0	1	11	0001	110	2	2	7		000	B
												- r					001	C
																	010	D
ADD A, (HL)	A ← A + (HL)	1	1	X	1	X	V	0	1	10	0001	110	1	2	7		011	E
ADD A, (IX+d)	A ← A + (IX+d)	1	1	X	1	X	V	0	1	11	011	101	DD	3	5	19	100	H
																	101	L
																	111	A
ADD A, (IY+d)	A ← A + (IY+d)	1	1	X	1	X	V	0	1	11	111	101	FD	3	5	19		
ADC A, s	A ← A + s + CY	1	1	X	1	X	V	0	1									
SUB s	A ← A - s	1	1	X	1	X	V	1	1									
SBC A, s	A ← A - s - CY	1	1	X	1	X	V	1	1									
AND s	A ← A & s	1	1	X	1	X	P	0	0									
OR s	A ← A   s	1	1	X	0	X	P	0	0									
XOR s	A ← A ⊕ s	1	1	X	0	X	P	0	0									
CP s	A ← A - s	1	1	X	1	X	V	1	1									
INC r	r ← r + 1	1	1	X	1	X	V	0	0	00	r	100	1	1	4			
INC (HL)	(HL) ← (HL) + 1	1	1	X	1	X	V	0	0	00	110	100	1	3	11			
INC (IX+d)	(IX+d) ← (IX+d) + 1	1	1	X	1	X	V	0	0	11	011	101	DD	3	5	23		
INC (IY+d)	(IY+d) ← (IY+d) + 1	1	1	X	1	X	V	0	0	11	111	101	FD	3	5	23		
DEC s	s ← s - 1	1	1	X	1	X	V	1	0									

Notes: The V symbol in the P/V flag column indicates that the P/V flag contains the overflow of the result of the operation. Similarly the P symbol indicates parity. V = 1 means overflow, V = 0 means not overflow, P = 1 means parity of the result is even, P = 0 means parity of the result is odd.

Flag Notation: 0 = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.  
1 = flag is affected according to the result of the operation.

V - 110.

## 8 BIT ARITHMETIC AND LOGIC

	REGISTER ADDRESSING										REG. INDEX	INDEXED	IMMED.
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	r		
'ADD'	B7	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B0	B1
ADD w CARRY 'ADC'	B7	B8	B9	BA	BB	BC	BD	BE	BF	00	01	02	03
SUBTRACT 'SUB'	B7	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B0	B1
SUB w CARRY 'SBC'	B7	B8	B9	BA	BB	BC	BD	BE	BF	00	01	02	03
'AND'	A7	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A0	A1
'XOR'	A7	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A0	A1
'OR'	B7	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B0	B1
COMPARE 'CP'	B7	B8	B9	BA	BB	BC	BD	BE	BF	00	01	02	03
INCREMENT 'INC'	3C	04	0C	14	1C	24	2C	34	3C	44	4C	54	5C
DECREMENT 'DEC'	3D	05	0D	15	1D	25	2D	35	3D	45	4D	55	5D

## GENERAL PURPOSE ARITHMETIC AND CPU CONTROL GROUPS

Mnemonic	Symbolic Operation	Flags								Op-Code			No. of Bytes	No. of Cycles	No. of T States	Comments		
DAA	Converts acc. content into packed BCD following add; or subtract with packed BCD operands	S	Z	N	P	V	N	C	78	5A	21	0	100 111	27	1	1	4	Decimal adjust accumulator
CPL	$A \leftarrow \bar{A}$	*	*	X	1	X	*	1	*	00	101 111	2F	1	1	1	4		Complement accumulator (One's complement)
NEG	$A \leftarrow \bar{A} + 1$	1	1	X	1	X	V	1	1	11	101 101	ED	2	2	2	8		Negate acc. (Two's complement)
CCF	$CY \leftarrow \bar{CY}$	*	*	X	X	X	*	0	1	01	000 111	44	1	1	1	4		Complement carry flag
SCF	$CY \leftarrow 1$	*	*	X	0	X	*	0	1	00	110 111	37	1	1	1	4		Set carry flag
NOP	No operation	*	*	X	*	X	*	*	*	00	000 000	00	1	1	1	4		
HALT	CPU halted	*	*	X	*	X	*	*	*	01	110 110	7E	1	1	1	4		
DI*	IFF = 0	*	*	X	*	X	*	*	*	11	110 011	F3	1	1	1	4		
EI*	IFF = 1	*	*	X	*	X	*	*	*	11	111 011	FB	1	1	1	4		
IM 0	Set interrupt mode 0	*	*	X	*	X	*	*	*	11	101 101	ED	2	2	2	8		
IM 1	Set interrupt mode 1	*	*	X	*	X	*	*	*	01	000 110	46	2	2	2	8		
IM 2	Set interrupt mode 2	*	*	X	*	X	*	*	*	11	101 110	5E	2	2	2	8		

Notes: IFF indicates the interrupt enable flip-flop.  
CY indicates the carry flip-flop.

Flag Notation: \* = flag not effected, 0 = flag reset, 1 = flag set, X = flag is unknown,  
= flag is affected according to the result of the operation.

\*Interrupts are not sampled at the end of EI or DI

"NOP"	00
"HALT"	7E
"DISABLE INT 'DI'"	F3
"ENABLE INT 'EI'"	FB
"SET INT MODE 0 'IM0'"	ED 46
"SET INT MODE 1 'IM1'"	ED 5E
"SET INT MODE 2 'IM2'"	ED 5E

8080A MODE

CALL TO LOCATION 003H

INDIRECT CALL USING REGISTER I AND 8 BITS FROM INTERRUPTING DEVICE AS A POINTER.

Decimal Adjust Acc, "DAA"	27
Complement Acc, "CPL"	2F
Negate Acc, "NEG" (Two's complement)	ED 44
Complement Carry Flag, "CCF"	44
Set Carry Flag, "SCF"	37

## MISCELLANEOUS CPU CONTROL

## GENERAL PURPOSE AF OPERATIONS

# 16-BIT ARITHMETIC GROUP

V - 112.

Mnemonic	Symbolic Operation	Flags								Op-Code			No. of Bytes	No. of Cycles	No. of T-States	Comments	
		S	Z	N	V	P	V	M	C	16	24	25					
ADD HL, m	HL ← HL+m	*	*	X	X	X	*	*	0	1	00	m1	001	1	3	11 m 00 01 10 11 Reg. BC DE HL SP	
ADC HL, m	HL ← HL+m+CY	1	1	X	X	X	V	0	1	11	011	101	ED	2	4	15 01 10 11 SP	
SBC HL, m	HL ← HL-m-CY	1	1	X	X	X	V	1	1	11	101	101	ED	2	4	15 01 10 11 SP	
ADD IX, pp	IX ← IX+pp	*	*	X	X	X	*	*	0	1	11	011	101	DD	2	4	15 pp 00 01 10 11 Reg. BC DE IX SP
ADD IV, rr	IV ← IV+rr	*	*	X	X	X	*	*	0	1	11	111	101	FD	2	4	15 rr 00 01 10 11 Reg. BC DE IV SP
INC m	m ← m+1	*	*	X	*	X	*	*	*	0	00	m0	011	1	1	5	
INC IX	IX ← IX+1	*	*	X	*	X	*	*	*	1	01	101	101	DD	2	2	10
INC IV	IV ← IV+1	*	*	X	*	X	*	*	*	1	11	101	101	FD	2	2	10
DEC m	m ← m-1	*	*	X	*	X	*	*	*	0	00	m1	011	1	1	5	
DEC IX	IX ← IX-1	*	*	X	*	X	*	*	*	1	01	101	101	DD	2	2	10
DEC IV	IV ← IV-1	*	*	X	*	X	*	*	*	1	11	101	101	FD	2	2	10

Notes: m is any of the register pairs BC, DE, HL, SP  
pp is any of the register pairs BC, DE, IX, SP  
rr is any of the register pairs BC, DE, IV, SP.

Flag Notation: \* = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.  
1 = flag is affected according to the result of the operation.

		SOURCE					
		BC	DE	HL	SP	IX	IV
DESTINATION	'ADD'	HL 08	19	29	39		
		IX D0	D0		D0	D0	
		IV F0	F0		F0	F0	
	ADD WITH CARRY AND SET FLAGS 'ADC'	HL E0	E0	E0	E0		
		IX 4A	5A	6A	7A		
	SUB WITH CARRY AND SET FLAGS 'SBC'	HL E0	E0	E0	E0		
	INCREMENT 'INC'	03	13	23	33	D0	F0
						D0	D0
	DECREMENT 'DEC'	0B	1B	2B	3B	D0	F0

## 16 BIT ARITHMETIC



# BIT SET, RESET AND TEST GROUP

V - 114.

Monemonic	Symbolic Operation	S	Z	X	N	P/V	H	C	Op-Code	No. of Hex	Bytes	Cycles	Max of Status	Max of T	Comments
BIT b, r	$Z \rightarrow \bar{Z}_b$	X	1	X	1	X	X	0	11 001 011 001 b r	CB	2	2	8	1	000 001 010 011 100 101 111 A
BIT b, (HL)	$Z \rightarrow (\bar{HL})_b$	X	1	X	1	X	X	0	11 001 011 001 b 110	CB	2	3	12	1	000 001 010 011 100 101 111 A
BIT b, (IX+d)	$Z \rightarrow (\bar{IX+d})_b$	X	1	X	1	X	X	0	11 011 101 11 001 011 - d - 001 b 110	DD	4	5	20	1	000 001 010 011 100 101 111 A
BIT b, (IY+d)	$Z \rightarrow (\bar{IY+d})_b$	X	1	X	1	X	X	0	11 111 101 11 001 011 - d - 001 b 110	FD	4	5	20	1	000 001 010 011 100 101 111 A
SET b, r	$Z_b \rightarrow 1$	*	*	X	*	X	*	*	11 001 011 001 b r	CB	2	2	8	1	000 001 010 011 100 101 111 A
SET b, (HL)	$(HL)_b \rightarrow 1$	*	*	X	*	X	*	*	11 001 011 001 b 110	CB	2	3	15	1	000 001 010 011 100 101 111 A
SET b, (IX+d)	$(IX+d)_b \rightarrow 1$	*	*	X	*	X	*	*	11 011 101 11 001 011 - d - 001 b 110	DD	4	5	23	1	000 001 010 011 100 101 111 A
SET b, (IY+d)	$(IY+d)_b \rightarrow 1$	*	*	X	*	X	*	*	11 111 101 11 001 011 - d - 001 b 110	FD	4	5	23	1	000 001 010 011 100 101 111 A
RES b, r	$Z_b \rightarrow 0$ b, r, (HL), (IX+d), (IY+d)	*	*	X	*	X	*	*	11 001 011 001 b 110	CB	2	2	8	1	000 001 010 011 100 101 111 A

To form new Op-Code replace [ ] of SET b, r with [ ] Flags and time status for SET instructions

Notes: The notation  $z_b$  indicates bit b (0 to 7) or location  $z$ .

Flag Notation: \* = Flag not affected, 0 = Flag reset, 1 = Flag set, X = Flag is unknown,  $\bar{\phantom{x}}$  = Flag is affected according to the result of the operation.

BIT	REGISTER ADDRESSING								REG. MOD.		INDEXED	
	A	B	C	D	E	H	L	HL	HL	HL	HL	HL
TEST BIT	0	00	00	00	00	00	00	00	00	00	00	00
	1	00	00	00	00	00	00	00	00	00	00	00
	2	00	00	00	00	00	00	00	00	00	00	00
	3	00	00	00	00	00	00	00	00	00	00	00
	4	00	00	00	00	00	00	00	00	00	00	00
	5	00	00	00	00	00	00	00	00	00	00	00
	6	00	00	00	00	00	00	00	00	00	00	00
RESET BIT	0	00	00	00	00	00	00	00	00	00	00	00
	1	00	00	00	00	00	00	00	00	00	00	00
	2	00	00	00	00	00	00	00	00	00	00	00
	3	00	00	00	00	00	00	00	00	00	00	00
	4	00	00	00	00	00	00	00	00	00	00	00
	5	00	00	00	00	00	00	00	00	00	00	00
	6	00	00	00	00	00	00	00	00	00	00	00
SET BIT	0	00	00	00	00	00	00	00	00	00	00	00
	1	00	00	00	00	00	00	00	00	00	00	00
	2	00	00	00	00	00	00	00	00	00	00	00
	3	00	00	00	00	00	00	00	00	00	00	00
	4	00	00	00	00	00	00	00	00	00	00	00
	5	00	00	00	00	00	00	00	00	00	00	00
	6	00	00	00	00	00	00	00	00	00	00	00

## BIT MANIPULATION GROUP

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Op-Code	78	543	210	Hex	Bytes	Cycles	Status	Comments	
JP nn	PC ← nn	*	*	X	*	X	*	11 000 011	-	-	-	C3	3	3	10		
								-	-	-	-						
								-	-	-	-						
JP cc, nn	If condition cc is true PC ← nn, otherwise continue	*	*	X	*	X	*	11 cc 010	-	-	-		3	3	10	cc Condition 000 NZ not zero 001 Z zero 010 NC not carry 011 C carry 100 PO parity odd 101 PE parity even 110 P sign positive 111 M sign negative	
								-	-	-	-						
								-	-	-	-						
JR e	PC ← PC + e	*	*	X	*	X	*	00 011 000	-	e-2	-		18	2	3	12	
								-	e-2	-	-						
JR C, e	If C = 0, continue If C = 1, PC ← PC + e	*	*	X	*	X	*	00 111 000	-	e-2	-		38	2	2	7	If condition not met
								-	e-2	-	-						
								-	e-2	-	-		2	3	12	If condition is met	
JR NC, e	If C = 0, continue If C = 1, PC ← PC + e	*	*	X	*	X	*	00 110 000	-	e-2	-		30	2	2	7	If condition not met
								-	e-2	-	-						
								-	e-2	-	-		2	3	12	If condition is met	
JR Z, e	If Z = 0, continue If Z = 1, PC ← PC + e	*	*	X	*	X	*	00 101 000	-	e-2	-		28	2	2	7	If condition not met
								-	e-2	-	-						
								-	e-2	-	-		2	3	12	If condition is met	
JR NZ, e	If Z = 1, continue If Z = 0, PC ← PC + e	*	*	X	*	X	*	00 100 000	-	e-2	-		20	2	2	7	If condition not met
								-	e-2	-	-						
								-	e-2	-	-		2	3	12	If condition is met	
JP (HL)	PC ← HL	*	*	X	*	X	*	11 101 001				E8	1	1	4		
								-									
JP (IX)	PC ← IX	*	*	X	*	X	*	11 011 101				D3	2	2	8		
								-									
								-									
JP (IY)	PC ← IY	*	*	X	*	X	*	11 111 101				F0	2	2	8		
								-									
								-									
								-									
DJNZ, e	B ← B-1 If B = 0, continue If B ≠ 0, PC ← PC + e	*	*	X	*	X	*	00 010 000	-	e-2	-		10	2	2	8	If B = 0
								-	e-2	-	-						
								-	e-2	-	-		2	3	13	If B ≠ 0	

Notes: e represents the extension in the relative addressing mode.

e is a signed two's complement number in the range <126, 128>

e-2 in the op-code provides an effective address of pc+e as PC is incremented by 2 prior to the addition of e.

Flag Notations: \* = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown, | = flag is affected according to the result of the operation.

# CONDITION

			UNCOND.	CARRY	NON CARRY	NON ZERO	NON ZERO	PARITY EVEN	PARITY ODD	SIGN NEG	SIGN POS	REG S+0
JUMP "JP"	IMMED. EXT.	nn	C3	D3	D3	E8	E8	F0	F0	F0	F0	
JUMP "JR"	RELATIVE	PC+e	18	38	30	28	20					
			e-2	e-2	e-2	e-2	e-2					
JUMP "JP"		(HL)										
JUMP "JP"	REG. INDIR.	(IX)	D3									
			E8									
JUMP "JP"		(IY)										
"CALL"	IMMED. EXT.	nn	E8	D3	D3	E8	E8	F0	F0	F0	F0	
DECREMENT B, JUMP IF NON ZERO "DJNZ"	RELATIVE	PC+e										10
												e-2
RETURN "RET"	REGISTER INDIR.	(SP) (SP+1)	E8	D3	D3	E8	E8	F0	F0	F0	F0	
RETURN FROM INT "RETI"	REG. INDIR.	(SP) (SP+1)	E8									
			40									
RETURN FROM NON MASKABLE INT "RETI"	REG. INDIR.	(SP) (SP+1)	E8									
			45									

## JUMP, CALL AND RETURN GROUP



# CALL AND RETURN GROUP

V-116.

Mnemonic	Symbolic Description	Flags								Op-Code				No. of Bytes	Head M Cycles	Head T Cycles	Comments
		S	Z	N	P/V	H	C	TS	SAV	2YR	Non						
CALL nn	(SP-1) - PC <sub>H</sub> (SP-2) - PC <sub>L</sub> PC - nn	*	*	X	*	X	*	*	*	11	001	101	00	3	5	17	
CALL cc, nn	If condition cc is false, continue, otherwise same as CALL nn	*	*	X	*	X	*	*	*	11	cc	100	-	3	5	17	If cc is false
																	If cc is true
RET	PC <sub>L</sub> - (SP) PC <sub>H</sub> - (SP+1)	*	*	X	*	X	*	*	*	11	001	001	00	1	3	10	
RET cc	If condition cc is false, continue, otherwise same as RET	*	*	X	*	X	*	*	*	11	cc	000	-	1	1	5	If cc is false
																	If cc is true
RETI	Return from interrupt	*	*	X	*	X	*	*	*	11	101	101	ED	2	4	14	
RETN <sup>1</sup>	Return from non maskable interrupt	*	*	X	*	X	*	*	*	11	101	101	ED	2	4	14	
																	000
																	001
																	010
RST p	(SP-1) - PC <sub>H</sub> (SP-2) - PC <sub>L</sub> PC <sub>H</sub> - 0 PC <sub>L</sub> - p	*	*	X	*	X	*	*	*	11	1	111	-	1	3	11	
																	000
																	001
																	010
																	011
																	100
																	101
																	110
																	111

<sup>1</sup> RETN loads IFF<sub>2</sub> - IFF<sub>3</sub>.

Flag notation: \* = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown, - = flag is affected according to the result of the operation.

OP CODE	Mnemonic	Comments
0000 <sub>h</sub>	RET 0	
0001 <sub>h</sub>	RET 1	
0002 <sub>h</sub>	RET 2	
0003 <sub>h</sub>	RET 3	
0004 <sub>h</sub>	RET 4	
0005 <sub>h</sub>	RET 5	
0006 <sub>h</sub>	RET 6	
0007 <sub>h</sub>	RET 7	
0008 <sub>h</sub>	RET 8	
0009 <sub>h</sub>	RET 9	
000A <sub>h</sub>	RET 10	
000B <sub>h</sub>	RET 11	
000C <sub>h</sub>	RET 12	
000D <sub>h</sub>	RET 13	
000E <sub>h</sub>	RET 14	
000F <sub>h</sub>	RET 15	

## RESTART GROUP

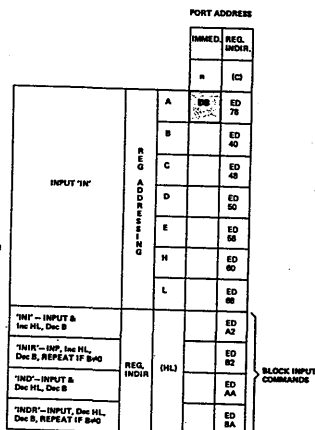
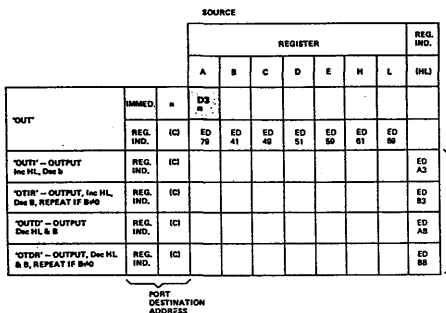
# INPUT AND OUTPUT GROUP

V -117.

Mnemonic	Symbolic Operation	S	Z	N	P	V	N	C	Op-Code	Hex	Bytes	No. of M Cycles	No. of T States	Comments
IN A, (n)	A ← (n)	*	*	X	*	*	*	*	11 011 011	D8	2	3	11	n to A <sub>0</sub> ~ A <sub>7</sub> Acc to A <sub>8</sub> ~ A <sub>15</sub>
IN r, (C)	r ← (C) If r = 110 only the flag will be affected	1	1	X	1	X	P	0	11 101 101 01 r 000	F0	2	3	12	C to A <sub>0</sub> ~ A <sub>7</sub> B to A <sub>8</sub> ~ A <sub>15</sub>
INI	(HL) ← (C) B ← B - 1 HL ← HL + 1	X	1	X	X	X	X	1	11 101 101 10 100 010	E0 A2	2	4	16	C to A <sub>0</sub> ~ A <sub>7</sub> B to A <sub>8</sub> ~ A <sub>15</sub>
INIR	(HL) ← (C) B ← B - 1 HL ← HL + 1 Repeat until B = 0	X	1	X	X	X	X	1	11 101 101 10 110 010	E0 B2	2	5	21	C to A <sub>0</sub> ~ A <sub>7</sub> B to A <sub>8</sub> ~ A <sub>15</sub>
IND	(HL) ← (C) B ← B - 1 HL ← HL - 1	X	1	X	X	X	X	1	11 101 101 10 101 010	E0 AA	2	4	16	C to A <sub>0</sub> ~ A <sub>7</sub> B to A <sub>8</sub> ~ A <sub>15</sub>
INDR	(HL) ← (C) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	1	X	X	X	X	1	11 101 101 10 111 010	E0 BA	2	5	21	C to A <sub>0</sub> ~ A <sub>7</sub> B to A <sub>8</sub> ~ A <sub>15</sub>
OUT (n), A	(n) ← A	*	*	X	*	*	*	*	11 010 011	03	2	3	11	n to A <sub>0</sub> ~ A <sub>7</sub> Acc to A <sub>8</sub> ~ A <sub>15</sub>
OUT (C), r	(C) ← r	*	*	X	*	*	*	*	11 101 101 01 r 001	E0	2	3	12	C to A <sub>0</sub> ~ A <sub>7</sub> B to A <sub>8</sub> ~ A <sub>15</sub>
OUTI	(C) ← (HL) B ← B - 1 HL ← HL + 1	X	1	X	X	X	X	1	11 101 101 10 100 011	E0 A3	2	4	16	C to A <sub>0</sub> ~ A <sub>7</sub> B to A <sub>8</sub> ~ A <sub>15</sub>
OTIR	(C) ← (HL) B ← B - 1 HL ← HL + 1 Repeat until B = 0	X	1	X	X	X	X	1	11 101 101 10 110 011	E0 B3	2	5	21	C to A <sub>0</sub> ~ A <sub>7</sub> B to A <sub>8</sub> ~ A <sub>15</sub>
OUTD	(C) ← (HL) B ← B - 1 HL ← HL - 1	X	1	X	X	X	X	1	11 101 101 10 101 011	E0 AB	2	4	16	C to A <sub>0</sub> ~ A <sub>7</sub> B to A <sub>8</sub> ~ A <sub>15</sub>
OTDR	(C) ← (HL) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	1	X	X	X	X	1	11 101 101 10 111 011	E0 BB	2	5	21	C to A <sub>0</sub> ~ A <sub>7</sub> B to A <sub>8</sub> ~ A <sub>15</sub>

Notes: ① If the result of B - 1 is zero the Z flag is set, otherwise it is reset.

Flag Notation: \* = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown, 1 = flag is affected according to the result of the operation.















OUTPUT GROUP

INPUT GROUP

## CARTE MEMOIRE

	<u>Adresse hexadécimale</u>	<u>Adresse décimale</u>
ZONE DE TRAVAIL DE BASIC	FFFF	65535
PAGE VIDEO 2	F800	63488
ZONE DE TRAVAIL DU PROGRAMME	E000	57344
ESPACE LIBRE	C000	49152
ESPACE LIBRE	8000	32768
PAGE VIDEO 1	7800	30720
INTERPRETEUR BASIC	6000	24576
	10	0

## CONTENU DU GENERATEUR DE CARACTERES

Poids fort Poids faible		4								
		0	1	2	3	4	5	6	7	8
4	0		$\pi$		0	@	P		p	♠
	1			!	1	A	Q	a	q	♥
	2			"	2	B	R	b	r	♣
	3			#	3	C	S	c	s	♦
	4			\$	4	D	T	d	t	○
	5			%	5	E	U	e	u	●
	6			&	6	F	V	f	v	
	7			'	7	G	W	g	w	
	8			(	8	H	X	h	x	
	9			)	9	I	Y	i	y	
	A			*	:	J	Z	j	z	
	B			+	;	K	(	k	{	
	C			,	<	L	¥	l		
	D			-	=	M	)	m	}	
	E			.	>	N	^	n	~	
	F			/	?	O	_	o		

Les valeurs sont données en hexadécimal.

## MESSAGES D'ERREUR

Un message d'erreur est affiché lorsqu'une erreur est détectée  
pouvant causer l'interruption du programme.

Ce message d'erreur est affiché comme suit

"message d'erreur" in "numéro de ligne"

## LISTE DES 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 ?

### Duplicate definition

- Y-a-t-il une déclaration DIM en cours utilisant un nom qui fait double emploi ?

### Illegal function call

- Le nombre utilisé est-il en dehors des paramètres autorisés ?

### 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 ?

### Too long string

- La longueur de la chaîne de caractères dépasse t-elle 255 caractères ? Vérifiez à l'endroit où la chaîne de caractères est additionné.

### Missing operand

- Un paramètre indispensable manque. Le côté droit de = commande 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 ?
- Vérifiez 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  $1.7 \times 10^{38}$
- Le PHC 25 est limité à  $1.7 \times 10^{38}$ .  
( $1.7$  multiplié par  $10$  à la puissance 38)

Return w/o gosub

- S'agit-il d'une instruction RETURN sans instruction GOSUB ?
- Vérifiez la correspondance des deux instructions

Syntax error

- La frappe a-t-elle été faite correctement ?
- Les noms variables sont-ils corrects ?

String too complex

- La chaîne de caractères est trop compliquée

Type mismatch

- Les chaînes de caractères et les valeurs numériques sont-elles mélangées ?

Tape read error

- La cassette n'a pas été lue correctement.

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

Divided by zéro

Une division par zéro a été tentée, ce qui est impossible.

Le PHC 25 considère toute valeur inférieure à

$1 \times 10^{-38}$  comme nulle.

=====



## SOLUTION AUX PROBLEMES

### ERREUR

### VERIFICATION A FAIRE

L'écran n'affiche rien

1. L'alimentation est-elle branchée sur le PHC 25 ?
2. L'alimentation est-elle branchée sur le moniteur vidéo ou poste TV ?
3. La connection entre le PHC 25 et le moniteur vidéo SG 12 est-elle en place ?
4. La connection entre le PHC 25 et l'adaptateur pour le poste TV (lorsque c'est le cas) est-elle faite ?
5. Contraste ou luminosité du poste moniteur ou TV

L'écran n'est pas stable

Effectuer les règles verticaux et horizontaux de synchronisation

Arrêt en cours d'émission

1. S'agit-il d'une coupure de courant ?
2. Interférence d'un autre équipement. Evitez l'utilisation d'un poste de radio etc...
3. Vérifiez votre programme

La commande BREAK n'a pas d'effet

Fermez l'interrupteur et recommencez à nouveau

CSAVE ou CLOAD  
ne fonctionnent pas

1. Recommencez toutes les séquences en suivant rigoureusement les indications du mode d'emploi
2. Vérifiez le magnétophone et l'état du chargement des piles si vous utilisez des piles
3. Vérifiez l'adaptateur et l'alimentation du magnétophone
4. Vérifiez le réglage de volume et de tonalité
5. Nettoyez la tête de lecture

Complément sur les variables numériques :

1. Les réels

Les variables numériques et les nombres que nous avons vus tout au long du manuel étaient des réels : les réels peuvent prendre deux formes suivant leur valeur :

Pour les nombres compris entre  $0.011111$  et  $999999$   
et  $-999999$  à  $-0.011111$

La notation en virgule fixe est utilisée.  
(le zéro à gauche du point n'est pas obligatoire :  $0.12 = .12$ )

Pour les nombres plus grands ou plus petits, on utilise la notation exponentielle ou virgule flottante.

Dans ce cas, un nombre est représenté en deux parties :

- la mantisse (qui est composée des chiffres significatifs) comprend au plus 6 chiffres)
- l'exposant (qui représente la position réelle de la virgule dans la mantisse) est compris entre -38 et +38

Exemple :  $0.0000001$  s'écrit  $1E-07$  ce qui veut dire 1, sept positions à droite de la virgule ( $10^{-7}$ )

$1000000$  s'écrit  $1E+06$  signifie 1 suivi de 6 zéros ( $1 \times 10^6$ )

## 2. Les entiers

Il existe une deuxième catégorie de variables numériques, les entiers.

Les entiers ne peuvent avoir de partie décimale et les limites sont :

-32768 et +32767

Dans le cas où les valeurs ne dépassent ces bornes, on a intérêt à utiliser des variables "entières" surtout dans les boucles FOR... NEXT.

En effet l'utilisation d'une variable "entière" comme compteur permet un gain de vitesse jusqu'à 40 %.

Essayez FOR I=1 TO 1000:NEXT  
et FOR I%=1 TO 1000:NEXT

## LISTE DES NOMS DE VARIABLES INTERDITS

ABS	ASC	CHR\$
CLEAR	CLOAD	CLOAD?
CLS	COLOR	CONSOLE
CONT	CSAVE	CSRLIN
CTON	CTOFF	
DATA	DEFFN	DIM
END	ELSE	EXEC
EXP	FOR	FRE
GOSUB	GOTO	IF
INKEY\$	INP	INPUT
INT	KEY	LCOPY
LEFT\$	LEN	LET
LINE	LIST	LLIST
LOCATE	LOG	LPOS
LPRINT	MID\$	NEXT
NEW	ON	OUT
PAINT	PEEK	PLAY
POINT	POKE	POS
PRESET	PRINT	PSET
READ	REM	
RESTORE	RETURN	RIGHT\$
RND	RUN	SCREEN
SGN	SIN	SOUND
SPC	SQR	SCRIN
SLOAD	SSAVE	STEP
STICK	STOP	STRIG
STR\$	TAB	TAN
THEN	TIME	TO
USR	VAL	FN

## LISTE DES INSTRUCTIONS DU BASIC ETENDU DU PHC 25

<u>Instruction</u>	<u>Tome</u>	<u>Page</u>	<u>Description</u>
ABS	3	26	valeur absolue
ASC	5	57	valeur ASCII
CHR\$	5	57	chaîne de caractères
CLEAR	5	49	mise à zéro
CLOAD	2	12	chargement de programme
CLOAD ?	2	12	contrôle de sauvegarde
CLS	5	7	effacement d'écran
COLOR	5	19	définition de la couleur
CONSOLE	5	8	subdivision de l'écran
CONT	5	38	continuation du programme
COS	3	31	cosinus
CSAVE	2	12	sauvegarde sur cassette
CSRLIN	5	16	lecture position curseur
CTOFF	5	62	arrêt moteur cassette
CTON	5	61	marcne moteur cassette
DATA	4	38	chaîne de données
DEFFN	5	39	définition de fonction
DIM	4	31	dimension de table
END	3	44	fin de programme
EXEC	5	52	saut à langage machine
EXP	3	31	calcul d'exponentielle
FRÉ	5	42	calcul mémoire disponible
FOR-NEXT	4	22	bouclage
GOSUB-RETURN	4	48	appel sous-programme
GOTO	3	46	branchement
IF-THEN-ELSE	3	34	exécution conditionnelle
INKEY\$	5	33	entrée de caractères
INP	5	58	entrée d'un port
INPUT	1 et 2		lecture depuis le clavier
INPUT #	5	63	lecture depuis un fichier
INT	3	27	calcul d'entiers
KEY	5	36	programmation touches F
LCOPY	5	70	recopie d'écran
LEFT\$	4	6	sélection de caractères
LEN	4	8	longueur chaîne caractères
LINE	5	23	tracage ligne écran
LIST	2	9	liste de programme
LLIST	2	14	liste sur imprimante
LOCATE	5	15	positionnement du curseur
LOG	3	31	logarithme
LPOS	5/5	17/72	pointeur de ligne imprimante
LPRINT	5	71	impression sur imprimante
MID\$	4	7	sélection de caractères

## LISTE DES INSTRUCTIONS DU BASIC ETENDU DU PHC 25

<u>Instruction</u>	<u>Tome</u>	<u>Page</u>	<u>Description</u>
NEW	2	13	effacement de la mémoire
ON GOSUB	4	55	appel sous-programme calculé
ON GOTO	4	52	branchement calculé
OUT	5	59	sortie vers un port
PAINT	5	29	coloriage d'une forme
PEEK	5	44	prise d'un octet en mémoire
PLAY	5	74	sortie musique
POINT	5	32	entrée couleur d'un point
POKE	5	44	positionnement octet mémoire
POS	5	17	lecture position verticale
PRESET	5	31	effacement d'un point
PRINT	1 et 2		écriture à l'écran
PRINT#		64	écriture sur périphérique
PSET	5	31	allumage d'un point à l'écran
READ	4	38	lecture de constantes
REM	3	59	remarques
RESTORE	4	38	réinitialisation des constantes
RIGHT\$	4	6	sélection de caractères
RND	4/5	57/43	nombres aléatoires
RUN	2	9	exécution de programme
SCREEN	5	11	définition d'écran
SCREEN 1	5	11	définition d'écran
SCREEN 2	5	11	définition d'écran
SCREEN 3	5	11	définition d'écran
SCREEN 4	5	11	définition d'écran
SCRIN	5	18	valeur ASCII d'un caractère écran
SIN	3	31	sinus
SLOAD	5	65	chargement mémoire écran
SGN	3	26	calcul du signe
SOUND	5	79	génération de sons
SPC	5	5	génération d'espaces
SQR	3	26	racine carrée
SSAVE	5	66	sauvegarde mémoire écran
STICK	5	97	contrôle de poignée de jeu
STOP	5	38	arrêt du programme
STRIG	5	102	contrôle de poignée de jeu
STR\$	4	17	conversion numérique en chaîne
TAB	5	6	tabulations
TAN	3	31	tangente
TIME	5	41	base de temps
USR	5	57	saut à langage machine
VAL	4	17	valeur chaîne de caractères







