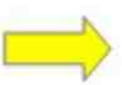
DAI

PERSONAL COMPUTER

MANUEL D'UTILISATION

DAI PERSONAL COMPUTER

Manuel d'utilisation



Première partie

deuxième partie

La société MULTISOFT remercie particulièrement Monsieur Thierry MANTOPOULOS pour sa précieuse collaboration dans l'élaboration de cet ouvrage.

1re édition : 1982

Imprimé en France

INTRODUCTION

Ce manuel est divisé en deux parties. Dans la première vous trouverez un guide qui vous permettra d'avancer pas à pas dans la programmation de votre machine.

Cette partie du manuel part de la considération que vous ignorez tout de la programmation. Elle vous guidera dans vos premiers pas en BASIC (Beginners All Purpose Symbolic Instructions Code – un langage de programmation), et en même temps elle vous présentera les caractéristiques particulières du D.A.I., caractéristiques indispensables à connaître pour utiliser pleinement ses possibilités.

Toutefois, le but de ce manuel n'est pas de vous donner un cours complet de programmation. L'auteur espère qu'après avoir lu ce livre, vous vous sentirez assez stimulé pour approfondir vos connaissances grâce aux nombreux livres existants sur le sujet, que ce manuel ne veut ni ne peut remplacer.

La deuxième partie du manuel contient les explications nécessaires à l'utilisation du BASIC du D.A.I. et vous aurez souvent besoin de vous y référer quand vous programmerez sur cette machine.

Ecrire un manuel accessible à un grand nombre d'utilisateurs n'est pas chose facile. Il est difficile de contenter tout le monde. Nous espérons que vous voudrez bien nous excuser de paraître trop compliqués par moments et trop superficiels à d'autres.

Bien sûr, si vous avez la moindre suggestion pour améliorer ce manuel, faites-le nous savoir.

MANUEL D'UTILISATION
1re Partie

TABLE DES MATIERES

I - BRANCHEMENTS

retour

La première chose à faire en arrivant chez vous est de trouver un endroit dégagé près d'une prise de courant.

Dans le carton où vous avez déjà trouvé ce manuel, vous trouverez aussi votre ordinateur (cette drôle de boîte blanche qui ressemble à une machine à écrire) et 3 câbles munis de prises.

Il s'agit de :

1) Le câble noir d'alimentation. Branchez-le à la prise marquée 220, à l'arrière du DAI et à votre prise de courant. Assurez-vous d'abord que votre courant est bien du 220 volts.

2) Le câble péritélévision. Il vous permettra de raccorder le D.A.I. directement sur le tube de votre téléviseur, sans passer par le démodulateur et le décodeur SECAM. Les résultats sont excellents. Branchez la prise longue à 21 broches à l'arrière de votre téléviseur et la DIN 6 sur la sortie VIDEO de votre DAI.

3) Le câble cassette. Il va relier votre magnétophone à cassettes au DAI. Pour ce faire, branchez la DIN 6 sur la sortie CAS 1 de votre DAI.

▶ ATTENTION : côté magnétophone, le JACK BLANC est à brancher sur l'entrée ECOUTEUR. Les 2 autres JACK sont à brancher sur «MICRO» et selon les cas, sur «CONTROLE MOTEUR».

Si vous devez le faire vous-même, voyez le schéma dans la 2e partie.

Si vous n'avez pas de magnétophone à cassettes, ne paniquez pas, vous pouvez toujours utiliser ce manuel (au moins en grande partie) sans en avoir besoin. Mais vous aurez besoin d'un magnétophone à cassettes pour sauvegarder (SAVE) les programmes que vous allez écrire et pour charger (LOAD) du magnétophone à l'ordinateur, vos programmes.

Vous n'avez pas besoin d'un modèle cher ; mais essayer d'en trouver un muni de :

- réglage de la tonalité
- trois sorties (MICRO-ECOUTEUR-TELECOMMANDE)
- éventuellement d'un compteur.

II - MISE EN MARCHE

Considérons maintenant que vous avez correctement branché tous les câbles et que vous êtes assis devant votre D.A.I... Vous devez allumer votre téléviseur et attendre l'image. Appuyez sur l'interrupteur situé à l'arrière du D.A.I. et vérifiez que l'interrupteur et que le voyant vert sont allumés.

Avec la prise Péritel, *pas de réglage* (sauf éventuellement, la luminosité), car la prise Péritélévision déconnecte le système de réception. Lorsque le poste sera allumé, vous verrez, sur un fond vert, en capitales blanches au milieu de l'écran, les mots suivants :

DAI PERSONAL
COMPUTER

Si, au lieu de cela, vous voyez un écran gris avec dans le coin supérieur gauche :

```
BASIC V1,1
*_
```

C'est que, par inadvertance, vous avez fait ce que nous allons vous demander dans un instant, c'est-à-dire, d'appuyer sur une touche du D.A.I. après l'avoir mis en marche. Si vous désirez revenir à l'étape précédente, vous devez simplement éteindre puis rallumer l'ordinateur, soit appuyer sur le bouton «RESET» à gauche du clavier à l'aide d'un stylo. Voilà ! maintenant l'écran est vert et affiche le bon message.

Si vous appuyez sur n'importe quelle touche, vous obtiendrez :

```
BASIC V1,1
*_
```

▶ Sur certains postes il vous faudra peut être ajuster le cadrage horizontal et vertical de votre image pour obtenir ce message, faites-le avant d'accuser votre ordinateur de mauvais fonctionnement.

retour

III - VOUS ETES PRET

L'ordinateur est prêt à accepter vos ordres en BASIC.

L'astérisque que vous pouvez voir sous le «B» est appelé «Indicateur de langage BASIC». Il vous signale qu'il attend des instructions en BASIC. Remarquez dès maintenant que tout de suite après l'astérisque ou Indicateur, il y a un trait qui clignote. C'est le CURSEUR (CURSOR) ; il indique l'emplacement du prochain caractère que vous taperez.

Chaque fois que vous verrez l'astérisque et le CURSEUR clignotant sous la dernière ligne de texte, cela signifiera que vous êtes sous contrôle de la machine ; c'est-à-dire que l'ordinateur n'exécute aucun programme et attend que vous tapiez un ordre ou un programme.

▶ Avant de continuer, prenez note de ce qui suit : c'est très important ! vous n'abîmerez pas votre ordinateur en jouant sur le clavier, mais n'essayez pas de RETIRER LES TOUCHES DE LEUR SUPPORT, cela provoquerait de GRAVES DOMMAGES à votre ordinateur.

Ceci dit, vous pouvez essayer tout ce que vous voulez pour voir les réponses de l'ordinateur. De toute façon, que vous tapiez des lettres au hasard ou des phrases parfaitement correctes, il y a beaucoup de chances pour que la réponse soit (après avoir appuyé sur la touche RETURN – retour du chariot ↵ expliqué plus loin) :

```
SYNTAX ERROR (erreur de syntaxe)
```

ou tout autre message d'erreur, montrant clairement que l'ordinateur ne «comprend» que le BASIC et non un véritable langage humain.

Ce manuel a été écrit pour vous donner les premiers éléments du langage que votre ordinateur comprend. Comme tous les autres langages, celui-ci ne s'apprend que par la pratique.

Aussi, nous vous invitons à essayer TOUS LES EXEMPLES donnés dans les pages qui suivent et même à en faire vous-même. Si vous le faites, ne soyez pas découragé par le premier message d'erreur que vous obtiendrez : vous aurez à vous y habituer, le programmeur qui n'a jamais eu de message d'erreur n'est pas encore né !

Lorsque nous voudrons que vous tapiez quelque chose, nous l'écrirons sur une ligne séparée, précédée d'une flèche. Vous taperez alors le texte exactement comme il sera écrit, y compris les espaces. Très souvent, les espaces jouent un rôle très important dans la syntaxe du BASIC, exactement commee nfran çais (bien sûr, vous n'avez eu aucun mal à me lire, mais souvenez-vous que les ordinateurs sont stupides !).

La première chose à faire si vous obtenez un message d'erreur (SYNTAX ERROR) est de vérifier que vous avez bien respecté les espaces présents dans le texte du manuel.


retour

IV - INTRODUCTION AU CLAVIER DU D.A.I.

Pour communiquer avec le D.A.I. vous aurez besoin de connaître son clavier. Il ressemble beaucoup à un clavier de machine à écrire mais il comporte des touches que même une dactylo expérimentée ne peut connaître, et d'autres dont le résultat sera différent de celui que vous pourriez escompter.

Si vous n'avez jamais utilisé de machine à écrire, cela importe peu, au contraire, un programme ne se tape pas sur un clavier à la même vitesse qu'une lettre, et, de plus, il est nécessaire lorsqu'on se sert d'un clavier d'ordinateur, de perdre certaines habitudes, comme taper un «O» à la place du zéro (sur ordinateur, le zéro est représenté par un «O» barré : Ø).

Aussi trouverez-vous ci-dessous la liste des principales touches que l'on ne retrouve pas sur une machine à écrire :
1) Les quatre touches grises de contrôle du curseur dans le coin gauche
2) La touche CTRL au-dessus de la touche SHIFT de gauche
3) La touche RETURN au-dessus de la touche DELCHAR
4) La touche DELCHAR au-dessus de la touche SHIFT de droite
5) La touche BREAK dans le coin supérieur droit
6) La touche REPT à droite de la touche DELCHAR

De plus, les touches suivantes ont deux rôles :
* représente la multiplication
/ représente la division
< signifie plus petit que
> signifie plus grand que
∧ signifie à la puissance (ex. : 2^3 s'écrira 2 ∧ 3).

1) Les touches de contrôle du curseur (CURSOR)

Elles sont utilisées en mode «édition» (EDIT) qui sera expliqué plus loin. La flèche de gauche est aussi utilisée par le moniteur langage machine (voir la deuxième partie du manuel). Elles n'ont normalement aucune autre fonction à moins que vous ne décidiez de les inclure dans un de vos programmes pour réaliser des rôles que vous voudriez leur attribuer (par exemple, dans un jeu pour décider de la direction d'un objet).

2) La touche CTRL

Elle vous permet de passer des MAJUSCULES aux minuscules (et inversement). Lorsque vous mettez le D.A.I. en marche, ou après avoir appuyé sur RESET (voir § 5), toutes les lettres seront en MAJUSCULES puisque tous les programmes en BASIC sont écrits en majuscules. Cependant, en écrivant des programmes, il est possible que vous désiriez que votre computer affiche quelques lignes de texte en minuscules. Il y a deux façons de les obtenir sur l'écran :

a) pour 1 lettre (ou plus), tapez la lettre désirée en maintenant enfoncée une des 2 touches SHIFT

b) pour une chaîne de caractères, appuyer sur la touche CTRL une fois, et toutes les lettres que vous taperez ensuite seront en minuscules.

Si vous choisissez la solution b), le clavier réagira exactement comme celui d'une machine à écrire, en redonnant des majuscules lorsque la touche CTRL sera enfoncée de nouveau.

Toutefois, il est à noter que la touche CTRL n'affecte pas les touches non alphabétiques qui ne donnent le caractère situé au-dessus que lorsque l'on appuie sur SHIFT. Par exemple, pour obtenir le signe ? (point d'interrogation) il faut à la fois, taper la touche qui porte le point d'interrogation et le signe de la division ?/ et maintenir enfoncée la touche SHIFT. Essayez.

Quelques minutes de pratique vous feront comprendre tout cela, aussi, essayez les 2 touches SHIFT et CTRL.

Lorsque vous aurez fini, votre écran sera rempli de choses bizarres. Bien que le D.A.I. vous permette d'ajouter de nouvelles lignes indéfiniment en décalant tout l'écran chaque fois d'une ligne vers le haut, il vous sera peut être utile de savoir maintenant comment il est possible d'effacer tout l'écran.

Appuyez d'abord sur la touche RETURN, – ne prêtez pas attention au message-erreur SYNTAX ERROR – pour être sûr de vous positionner sur une nouvelle ligne, puis tapez :
→?CHR$(12)
et appuyez ensuite sur RETURN. L'écran est vide. Notez simplement comment on efface l'écran sans chercher à en comprendre le fonctionnement.

Remplissez de nouveau l'écran et effacez-le de façon à vous habituer à cette procédure.

3) La touche RETURN

Une bonne définition de la touche RETURN pourrait être la «touche de non-retour» : en effet, cette touche enfoncée, il n'y a aucun moyen d'empêcher l'ordinateur d'analyser la ligne que vous venez de taper.

▶ Un mot d'explication : lorsque vous tapez quelque chose, l'ordinateur n'a aucun moyen de savoir quand vous avez fini, à moins que vous ne le lui indiquiez d'une façon ou d'une autre. C'est le rôle de la touche RETURN. Jusqu'à ce que vous l'ayiez enfoncée vous pouvez changer d'avis des millions de fois, mais une fois tapée...

Il vous faudra un certain temps pour vous habituer à cette touche. Nous vous y aiderons en inscrivant ce symbole spécial ↵ et en vous l'indiquant en toutes lettres.

4) La touche DELCHAR

Supposons que vous ayez tapé COMPUTET (!).
C'est alors que la touche DELCHAR prend tout son intérêt. Enfoncez-la une fois et COMPUTET devient COMPUTE, puis tapez «R», le résultat sera : «COMPUTER». Si vous tapez plusieurs fois DELCHAR, vous pouvez même effacer toute la ligne si vous le désirez, ceci, tant que vous n'avez pas appuyé sur RETURN

Le mode d'édition – EDIT – vous permet de corriger toute faute d'orthographe et de syntaxe aisément ; mais nous en reparlerons plus loin.

5) La touche BREAK

Lorsque vous mettez l'ordinateur en marche, vous en avez le contrôle. Il n'exécute aucun programme sauf, évidemment, ceux qui sont contenus dans les ROM (Read Only Memory) qui permettent au micro-processeur de comprendre vos ordres en BASIC.

Quand vous entrez un programme et que vous demandez à l'ordinateur de l'exécuter – RUN – ce dernier étant occupé, vous ne pouvez plus lui donner d'ordre au clavier. La seule façon d'arrêter le déroulement du programme et de retrouver le contrôle de l'ordinateur, est de presser la touche BREAK.

Parfois l'ordinateur est si occupé à exécuter ses calculs, qu'il ne peut «sentir» que vous avez pressé la touche BREAK. Aussi ne s'arrête-t-il pas.

Quand cela arrive (presque à chaque fois à cause d'une erreur de programmation ou «bug») la seule chose (dommage !) qu'il reste à faire est d'appuyer sur le bouton RESET à gauche du clavier. Précisons dans ce cas, que TOUT PROGRAMME que vous aurez soigneusement tapé pendant les 3 dernières heures sera perdu, à moins que vous n'ayez appris à le récupérer (cf 2e partie du manuel).

▶ Pour éviter cela, il est bon de sauvegarder – SAVE – régulièrement, sur cassette, le programme sur lequel vous travaillez, disons toutes les 20 ou 30 minutes, même si vous pensez qu'il n'est pas terminé.

En commençant au début de la cassette, sauvegardez ce qu'il y a dans la mémoire à ce moment-là, puis rembobinez la cassette pour sauvegarder la nouvelle version de votre programme sur la précédente qui sera ainsi effacée.

Vous ne perdrez pas beaucoup de temps et la dernière version du programme se trouvera toujours présente au début de la cassette où il est possible de la retrouver facilement.

Cette précaution se révèlera très utile lors d'une panne de courant ou d'une interruption imprévue (l'épouse ou l'époux qui s'ennuie et tire «accidentellement» sur le câble d'alimentation).

Si cela devait vous arriver, vous seriez content de ne pas avoir tout à recommencer, mais d'avoir simplement à charger – LOAD – la dernière version de votre programme.

Mais pour le moment, nous n'allons pas détailler la façon de sauvegarder – SAVE – et de charger – LOAD – des programmes, d'autant qu'il n'y a pas de programme en mémoire et que vous êtes censé ne pas savoir ce qu'est un programme (vous êtes un débutant, vous souvenez-vous ?).

6) La touche REPT

Maintenue enfoncée en même temps qu'une autre touche, elle permet sa répétition automatique. Essayez-la en pressant en même temps chacune des touches. Cela remplira l'écran de caractères que vous pourrez alors effacer pour passer au chapitre suivant.

retour

V - COMMENÇONS A TRAVAILLER

Maintenant que vous connaissez à peu près votre clavier, vous pouvez commencer à utiliser votre ordinateur.

Vous avez acheté un ordinateur capable de générer des couleurs pour le graphisme, le texte et le fond. Vous voyez pour l'instant l'image triste d'un texte noir sur un fond gris clair. Voici comment vous devez faire pour obtenir un écran en couleur : tapez l'instruction BASIC suivante.

▶ (Veuillez noter que toutes les instructions doivent être écrites en MAJUSCULES).

→ COLORT 5 Ø Ø Ø.

Bien sûr, rien ne se passera tant que vous n'aurez pas tapé RETURN. Une fois que vous l'aurez fait, vous verrez un texte noir sur fond vert. Si, au lieu de cela vous obtenez : SYNTAX ERROR, vérifiez que vous avez bien rentré cette instruction comme elle est écrite, y compris les espaces entre les chiffres.

Vous préféreriez peut être une autre combinaison de couleurs. Aussi, pourquoi ne pas sélectionner votre couleur préférée en essayant toutes les possibilités ?

▶ Pour cela souvenez-vous que le premier chiffre après COLORT (ici le 5) déterminera la couleur du fond et que le second (ici Ø) doit être séparé du premier par un espace et déterminera la couleur du texte.

(L'ordinateur considère le Ø comme un chiffre puisque les 16 couleurs présentes sur le D.A.I. sont numérotées de Ø à 15).

Les deux chiffres qui suivent (Ø Ø) doivent aussi être présents, bien qu'ils n'aient aucune fonction apparente. (La raison en est, si vous voulez vraiment le savoir, que l'ordre COLORT doit avoir la même syntaxe que l'ordre COLORG, dont nous reparlerons plus loin).

C'est facile, n'est-ce pas ?

Mais si vous avez acheté un ordinateur, c'est pour qu'il exécute à votre place des choses particulièrement ennuyeuses, comme taper :

COLORT quelque chose quelque chose.

256 fois, pour voir toutes les combinaisons texte/fond.

Ne serait-il pas agréable de voir l'ordinateur faire ce travail ?

Et bien, c'est possible. Cela sera votre premier exercice de programmation. Essayez de résister à l'envie pressante de vous précipiter sur les pages où se trouve le LISTING (suite des instructions) du programme final, et essayez plutôt de suivre les différentes étapes comme elles sont présentées, de façon à bien comprendre les caractéristiques du langage BASIC et du système D.A.I. qu'un court programme comme celui-ci va nous permettre de vous présenter.

retour

VI - POUR COMMENCER A PROGRAMMER

La première chose que vous devez savoir pour écrire un programme en BASIC, est qu'après avoir tapé RUN et appuyé sur RETURN l'ordinateur exécute dans l'ordre les instructions que vous avez stockées dans sa mémoire. L'ordre d'exécution de ces instructions est déterminé par le numéro qui les précède (appelé «numéro de ligne» – LINE NUMBER).

Chaque fois que vous tapez une instruction sans la faire précéder d'un numéro de ligne, elle sera EXECUTEE IMMEDIATEMENT après enfoncement de la touche RETURN. Si vous mettez un numéro de ligne avant celle-ci (n'importe quel nombre entre 1 et 65535), elle sera mise en mémoire après utilisation de RETURN et elle ne sera exécutée que si vous tapez RUN ↵.

Vous n'êtes pas obligé de nous croire sur parole, aussi tapez :

→ NEW↵

(puis RETURN) pour dire à l'ordinateur que vous voulez écrire un nouveau programme puis tapez la ligne suivante :

→100 COLORT 10 5 0 0↵

(souvenez-vous d'utiliser RETURN). Rien ne s'est passé, exact ? Faux !, quelque chose s'est passé : VOUS VENEZ D'ENTRER VOTRE PREMIER PROGRAMME dans le D.A.I. ! (mais nous reconnaissons que rien n'a semblé se passer).

→ LIST↵

et le D.A.I. affichera le LISTing du programme actuellement en mémoire, c'est-à-dire :

100 COLORT 10 5 0 0

Demander le LISTING du programme avant de le faire exécuter – RUN – est un bon moyen de voir s'il est correct. Maintenant tapez :

→ RUN↵

Voyez ce qui arrive. L'aviez-vous prévu ? Le texte est devenu vert et le fond orange : votre D.A.I. vient juste d'exécuter l'instruction COLORT 10 5 0 0 de la ligne 100.

Encouragé par ce succès, continuons à écrire le programme qui fera apparaître toutes les combinaisons de couleurs entre le texte et le fond. Remarquez que nous avons choisi 100 et non 1 comme numéro pour la première ligne. Pourquoi ? Parce qu'une des caractéristiques agréables du BASIC est qu'il ne tient pas compte de l'ordre dans lequel vous rentrez les instructions, mais seulement du numéro de ligne. Ainsi, après avoir tapé notre première ligne 100 COLORT 10 5 0 0 nous pouvons décider que le D.A.I. doit exécuter une autre instruction avant de changer la couleur du texte et du fond, en lui donnant un numéro de ligne PLUS PETIT que 100. Si notre première instruction avait été : 1 COLORT 10 5 0 0, comme le numéro 0 est ILLEGAL (interdit), nous aurions à retaper à la fois la nouvelle ligne et l'ancienne ligne.

▶ Aussi serait-il bon que vous preniez l'habitude de numéroter vos lignes en commençant à 100 puis en incrémentant de 10 à chaque nouvelle ligne. Cela laissera la place à des instructions supplémentaires intercalées, si besoin est.

Pour l'instant l'ordinateur ne nous fait pas vraiment gagner du temps puisque pour un texte vert sur fond orange nous devons taper plus de choses que précédémment. Exact, mais faites-nous confiance.

VARIABLES

Pour poursuivre plus avant, il vous faut connaître l'utilisation des VARIABLES. Les variables sont les noms des cases contenant les valeurs que vous avez besoin de stocker pour une utilisation future dans votre programme.

Pour créer une telle case, il suffit que vous lui trouviez un nom et que vous lui trouviez une valeur. Par exemple : LET A = 1 stockera la valeur 1 dans la case A, (lisez «LET A = 1» comme : décidons que «A a la valeur de 1») ce qui montre bien que c'est *vous* qui avez décidé qu'à la case que vous avez appelée A doit être attribuée la valeur 1.

Ce nom peut être formé aussi bien d'une lettre que d'une ligne de programme. Dans les deux cas, le D.A.I. ne reconnaîtra que les 14 premières lettres (ce qui est bien plus que les autres versions de BASIC qui ne reconnaissent que les 2 premiers caractères) ; aussi, en pratique, CELLECIESTTRESLONGUE et CELLECIESTTRESCOMPLIQUEE se rapporte à une seule case (parce que les 14 premiers caractères sont identiques). Mais nous espérons que vous n'aurez pas souvent à vous servir de noms de variables aussi longs !

La raison pour laquelle il est préférable d'utiliser un nom explicite pour une variable et non une simple lettre de l'alphabet est qu'il est plus simple de retrouver le rôle de la valeur contenue dans cette variable. Par exemple, dans notre programme, nous allons avoir besoin de variables pour stocker les codes des couleurs du texte et du fond que nous reporterons dans l'inscription COLORT

Nous pourrions décider d'appeler ces deux variables F (abréviation pour FOND) et T (abréviation pour TEXTE) mais nous pouvons aussi bien les appeler FOND et TEXTE ce qui éclaircira leur rôle pour tout le monde. Tapez :

→ LET TEXTE = 5↵

et surtout n'oubliez pas **RETURN**. Cela a deux résultats :
1) quelque part dans les mémoires du D.A.I., il y a une case qui s'appelle «TEXTE»
2) la valeur 5 est stockée dans cette case.

Comme d'habitude, nous ne pensons pas que vous allez nous croire sur parole. Alors comment vérifiez que nous avons raison ? Voici comment faire confirmer par votre D.A.I. que c'est bien ce qui s'est passé :

L'INSTRUCTION PRINT

Comment demandez-vous quelque chose d'habitude ? Avec une question, bien sûr ! C'est exactement ce que vous allez faire maintenant, sauf que le point d'interrogation vient en premier. Tapez :

→ ? TEXTE↵

Le point d'interrogation équivaut à taper le mot PRINT (mais il est plus court) et c'est de cette façon que vous demandez à votre ordinateur d'imprimer – PRINT – (sur l'écran ou sur une imprimante) la valeur de ce qui est contenu dans sa mémoire : Si vous n'oubliez pas la touche **RETURN**, la dernière ligne de l'écran doit être :

5.Ø

qui est la valeur stockée dans la case appelée TEXTE créée par l'instruction LET TEXTE = 5.
Pour vérifier que le résultat aurait été le même si vous aviez tapé PRINT au lieu de ?, taper :

→ PRINT TEXTE↵

(avons-nous toujours besoin de vous parler ce **RETURN** ?).

La raison pour laquelle vous avez obtenu 5.0 dans les deux cas, (?TEXTE et PRINT TEXTE) plutôt qu'un simple 5, est qu'à moins que vous ne le spécifiez, toutes les variables sont censées contenir des nombres réels (à virgule flottante) et non des nombres entiers. Si vous voulez qu'une variable ne contienne que des nombres entiers, vous devez ajouter le signe % à la fin de son nom, par exemple tapez :

→ LET TEXTE% = 5 ↵
puis,
→ ? TEXTE% ↵

qui vous donnera :
5

L'utilisation de nombres entiers au lieu de nombres réels permet au programme de se dérouler (-RUN-) plus rapidement, parce que les calculs sur les nombres entiers sont plus rapidement effectués.

Pour vous épargnez l'effort de taper %, après chaque nom de variable entière, le BASIC du D.A.I. vous permet de le décider avant de commencer à écrire un programme. Si par exemple, vous vouliez que toutes les variables dont le nom commence par A,B,C,D,E,F, soient entières, vous taperez :

→ IMP INT A-F ↵

IMP est l'instruction d'IMPlication et INT est une abréviation pour ENTIERE (INTeger en anglais).

Il y a beaucoup plus de choses à apprendre sur les types de variables et leur utilisation, mais pour le moment, c'est tout ce que vous devez savoir pour comprendre ce qui suit. Maintenant tapez :

→ NEW ↵

pour commencer un nouveau programme, puis tapez ces lignes :

→ 80 LET TEXTE = 10 ↵
→ 90 LET FOND = 9 ↵
→ 100 COLORT FOND TEXTE 0 0 ↵

Vous devez d'abord vérifier que ce que vous avez entré dans l'ordinateur correspond aux lignes écrites dans ce manuel. Pour ce faire, tapez :

→ LIST ↵

Dès que vous aurez tapé **RETURN**, l'écran sera effacé et la LISTe des instructions du programme apparaîtra. Essayez de deviner ce qui peut arriver lorsque ce programme sera exécuté. Puis, pour le voir se réaliser, tapez :

→ RUN ↵
et n'oubliez pas **RETURN**.

Le but de ce programme était de vous montrer que l'utilisation de *variables* au lieu de *nombres constants* a le même résultat.

(Au fait, n'auriez-vous pas obtenu un texte orange sur fond bleu ?).

A partir de maintenant, pour changer la couleur du texte et/ou du fond, il suffit de changer les valeurs obtenues dans les variables des lignes 80 et/ou 90 (sans retaper la ligne 100) et de faire exécuter (RUN) le programme pour obtenir le changement de couleur désiré.

Comment faire pour changer les valeurs aux lignes 80 et/ou 90 ? Pour le moment vous aurez à le faire par le moyen le plus difficile, c'est-à-dire, en retapant toute la ligne. Nous voulons que vous le fassiez, parce que cela vous fera comprendre que si jamais vous tapez une ligne ayant le *même numéro de ligne* qu'une ligne déjà existante, celle-ci est supprimée (DELETED) et la nouvelle prend sa place. Pour la même raison, si jamais vous avez besoin de supprimer (DELETE) une ligne de programme, il vous suffit de retaper le numéro de ligne et d'appuyer sur **RETURN**. Puisque la nouvelle ligne est

vide et que l'ordinateur n'accepte pas de ligne blanche, ce numéro de ligne ne sera plus présent dans le programme.

Essayez pendant quelques minutes de changer les couleurs de cette façon, puis revenez au manuel pour en savoir plus.

Le programme que nous écrivons ensemble doit vous montrer toutes les combinaisons de couleurs les unes après les autres. Aussi nous devons désigner une valeur de départ aux variables TEXTE et FOND (c'est le rôle des lignes 80 et 90) puis changer les couleurs de l'écran (c'est le rôle de la ligne 100).

Nous allons demander à l'ordinateur de changer les valeurs contenues dans les variables TEXTE et FOND, au lieu de le faire nous-mêmes.

Pour obtenir de l'ordinateur toutes les couleurs qu'il possède, commençons par lui faire ajouter 1 à la variable FOND de façon à ce qu'elle contienne le code de la couleur suivante.

Pour ce faire, vous devez lui dire d'attribuer à la variable sa valeur actuelle plus le nombre que vous désirez y ajouter. Exemple : si la variable est *fond* et le nombre que vous voulez additionner est 1, vous écrivez :

→ LET FOND = FOND + 1 ↵

Avant de continuer le programme il faut que vous essayez cela plusieurs fois pour voir que cela fonctionne comme vous l'imaginez.
Vous pouvez vous exercer de la façon suivante :

1) Vous imaginez un nom de variable, par exemple : MAMAN ou JEAN ou TOMBOUCTOU ou CE QUI VOUS PLAIT (jusqu'à 14 caractères)

2) Vous stockez dans la variable une valeur quelconque. Pour ce faire, tapez :

→ LET MAMAN = 30 ↵

3) Vérifiez que MAMAN contient la valeur 30 en tapant :

→ ? MAMAN ↵

la réponse sera :
30.0

4) Tapez par exemple :

→ LET MAMAN = MAMAN + 4 ↵

Là, rien n'a semblé se passer après RETURN. Mais vous pouvez vérifier que le D.A.I. a vraiment additionné 4 à la valeur présente dans MAMAN en tapant :

→? MAMAN ↵

ce qui donnera :
34.0
exactement comme prévu.

Essayez de recommencer ces différentes étapes en changeant le nom des variables, et la valeur initiale de la variable, en lui ajoutant des nombres, puis vérifiez le résultat. Cet exercice devrait vous familiariser avec un des concepts les plus difficiles à saisir quand on apprend le BASIC.

Maintenant, essayons d'utiliser ce nouveau petit bout de savoir pour additionner des valeurs à une variable dans notre programme.

▶ L'utilisation du mot LET est facultative et est souvent omise dans les programmes pour économiser ce byte supplémentaire. Mais comme vous êtes au début de votre carrière de programmeur, il serait peut-être plus sage d'utiliser LET pour se souvenir de ce qui se passe exactement. Aussi, ajoutez cette ligne supplémentaire au programme :

→ 110 LET FOND = FOND + 1 ↵

A l'exécution de la ligne 110, la valeur de l'expression mathématique située à droite du signe = sera calculée et stockée dans la variable FOND. Dans le calcul de l'expression à droite du signe = la valeur de FOND est évidemment la valeur stockée à ce moment dans cette case, celle-ci étant mise à 9 à la ligne 90. Cette valeur (9) sera ajoutée à 1 et le résultat stocké dans la même variable FOND. A partir de maintenant, la valeur de FOND sera 10, à moins qu'elle ne soit changée dans les lignes de programme qui suivent.

Avant de faire exécuter (-RUN-) le programme dans sa forme actuelle, vérifiez ce qu'il y a en mémoire ; tapez :

→ LIST ↵

et l'ordinateur vous affichera le programme en mémoire. Cela devrait donner :

```
 80 LET TEXTE = 10
 90 LET FOND = 9
100 COLORT FOND TEXTE 0 0
110 LET FOND = FOND + 1
```

Si vous avez effectivement essayé de changer les valeurs de TEXTE et FOND, les lignes 80 et 90 devront sûrement être changées de façon à indiquer les mêmes valeurs que ci-dessus. Vous pourriez retaper la totalité des deux lignes (comme ci-dessus) mais ce serait encore la façon la plus difficile. Au lieu de cela, nous allons vous expliquer comment changer un ou plusieurs caractères d'un programme, en utilisant :

retour

VII - LES INSTRUCTIONS D'EDITION

Tapez :

→ EDIT ↵

Si vous n'avez pas oublié RETURN, vous devez voir sur l'écran le LISTing de votre programme avec quelques différences :

– sur la gauche il y a une grande ligne verticale commençant juste sous la dernière ligne du programme.

– A la fin de chaque ligne vous pouvez voir le symbole ↵ , qui indique l'endroit où vous avez tapé la touche RETURN.

– Cette fois le curseur qui clignote dans le coin supérieur gauche peut être déplacé dans tout l'écran à l'aide des 4 touches de contrôle du curseur présentées plus haut.

Il faut savoir que :

– tout caractère tapé sera inséré à gauche du caractère clignotant avec le curseur.

– La touche DELCHAR effacera le caractère clignotant et comblera le vide en provoquant le déplacement des caractères situés à droite du curseur, d'une case vers la gauche.

Tout ceci peut paraître confus, mais le mode EDIT (pour EDITeur) vous apparaîtra très puissant (utile) et vraiment si facile à utiliser qu'il deviendra bientôt indispensable.

Entraînez-vous à modifier les valeurs de FOND et de TEXTE en procédant comme suit :

– placez le curseur sur la ligne 90 à droite du signe =. Pour le déplacer sur de grandes distances, vous pouvez enfoncer à la fois une des touches grises et la touche REPT, ce qui provoquera un déplacement rapide du curseur.

– appuyez sur DELCHAR autant de fois que nécessaire pour effacer la valeur actuelle.
– enfin, tapez la nouvelle valeur.

Une fois que vous avez modifié le programme, il vous faut sortir du mode EDIT.

Il y a deux façon de procéder :

▶ 1) Si vos modifications sont bonnes, appuyez sur BREAK UNE FOIS, puis sur la barre d'espacement. Pour les longs programmes, l'indicateur * ne réapparaîtra qu'après quelques secondes pendant lesquelles nous vous conseillons de n'appuyer sur aucune touche. Dans le cas de notre court programme, l'astérisque réapparaîtra tout de suite.

▶ 2) Si par contre vous voulez récupérer le programme initial (exempt de modifications) appuyez simplement DEUX FOIS sur BREAK.

Maintenant essayez la totalité de la procédure pour changer les valeurs de FOND et TEXTE en mode EDIT, puis sortez du mode EDIT avec BREAK suivi de la barre d'espacement et faites tourner (– RUN –) le programme pour voir le changement de couleur. Remarquez que si FOND et TEXTE se voient attribuer la même valeur, le texte semblera disparaître. Il est toujours là, mais ne peut se dissocier du fond (et pour cause).

Quand vous aurez fini de vous exercer, revenez au programme suivant :

```
 80 LET TEXTE = 8
 90 LET FOND = 0
100 COLORT FOND TEXTE 0 0
110 LET FOND = FOND + 1
```

retour

Sinon, vous savez comment le modifier.

Les boucles

La dernière ligne que nous avons ajoutée (110) fait que la valeur de la variable FOND devient le code de la couleur suivante. De toute façon, cela ne sert à rien de l'écrire si elle n'est pas suivie d'une instruction COLORT qui changera effectivement la couleur du fond. Tapez :

→ 120 COLORT FOND TEXTE 0 0 ↵

et faites - RUN -. Vous voyez ? Cette fois-ci vous avez obtenu un texte gris sur fond bleu parce que la ligne 110 a ajouté 1 à la valeur de FOND qui était 0 à la ligne 90.

Maintenant, pour obtenir un texte gris sur un fond violet vous pouvez ajouter ces deux lignes :

→ 130 LET FOND = FOND + 1 ↵

→ 140 COLORT FOND TEXTE 0 0 ↵

faites RUN.

Nous pourrions vous demander de continuer à ajouter des lignes identiques aux lignes 110 et 120 pour faire apparaître les 16 couleurs possibles pour le fond. Mais cela serait une tâche fatigante et trop répétitive. Aussi allons-nous vous expliquer une des techniques les plus courantes en programmation : la boucle.

Une des façons de programmer une boucle en BASIC est d'utiliser l'instruction GOTO. Tapez :

→ 120 GOTO 100 ↵

Avant de faire RUN, étudions le programme ligne à ligne.

Les lignes 80 et 90 assignent aux variables TEXTE et FOND une valeur initiale de façon à ce que la première combinaison de couleurs donne un texte gris sur fond noir.

La ligne 100 change les couleurs à partir des valeurs stockées actuellement dans FOND.

La première fois elle ajoutera 1 à 0 et stockera le résultat dans FOND, la deuxième fois, elle ajoutera 1 à la valeur actuelle de FOND c'est-à-dire 1 et stockera le résultat dans FOND et ainsi de suite.

La ligne 120 renvoie le programme à la ligne 100 qui changera la couleur du fond suivant la valeur de la variable FOND.

De cette façon, la variable FOND prendra toutes les valeurs possibles.

Maintenant imaginez ce qui va se passer si vous tapez RUN, et faites-le.

→ RUN ↵

L'aviez-vous prévu ? (non !). La couleur du fond a changé si rapidement que vous n'avez eu le temps de voir aucune des 16 couleurs et que le résultat final est un texte gris sur fond blanc, disant :

NUMBER OUT OF RANGE IN LINE 100
(nombre dépassant limite en ligne 100)

que s'est-il passé ?

Certaines actions de l'ordinateur sont beaucoup trop rapides pour être perçues par l'œil humain. Pour palier à cet inconvénient, le BASIC du D.A.I. comprend une instruction qui marque une pose de déroulement du programme.

Aussi, ajoutez ce qui suit :

→ 105 WAIT TIME 100 ↵

C'est-à-dire : «attends 100 unités de temps».

▶ Une unité de temps pour le D.A.I. vaut 20 millisecondes. Ici, le programme s'arrêtera pendant 100 fois 20 millisecondes, c'est-à-dire 2 secondes.

Le message d'erreur est dû au fait que l'ordinateur a essayé d'exécuter la ligne 100 avec une valeur de 16 pour la variable FOND. La liste de code des couleurs va de 0 à 15 et comme dans la mémoire de l'ordinateur il n'y a pas de couleur correspondant à la valeur 16, le D.A.I. vous dit que ce nombre n'appartient pas à sa liste. C'est ce qui est survenu à la 16e exécution de la ligne 100.

Comment éviter ce message d'erreur ? Continuez à lire...

L'instruction IF

Ce qui est intéressant dans les ordinateurs c'est leur capacité de prendre une décision à partir de certains facteurs définis par avance.

L'instruction BASIC IF...THEN (SI... ALORS) utilise cette particularité.

Mettons-la dans notre programme et étudions ses résultats.

Ajoutez cette ligne :

→ 120 IF FOND < 16 THEN GOTO 100 ↵

Alors, au lieu de renvoyer le programme à la ligne 100 de façon inconditionnée avec un GOTO 100, la ligne 120 ne le fera que si la variable FOND est < 16 (inférieure à 16), ce qui évitera l'erreur qui a provoqué ce message :

NUMBER OUT OF RANGE IN LINE 100.

Quand la valeur FOND sera égale à 16, la condition ne sera plus satisfaite et l'ordinateur n'exécutera plus l'instruction qui suit THEN (dans notre cas, il n'ira pas à la ligne 100). Au lieu de cela, il passera à la ligne suivante et l'exécutera. Ici il n'y a plus d'instruction et le programme s'arrêtera.

Maintenant LISTez le programme une dernière fois avant de le faire exécuter. (RUN).

```
 80 LET TEXTE = 8
 90 LET FOND = 0
100 COLORT FOND TEXTE 0 0
105 WAIT TIME 100
110 LET FOND = FOND + 1
120 IF FOND < 16 THEN GOTO 100
```

Vérifiez qu'il est correct, puis tapez :

→ RUN↵

Joli, n'est-ce pas ? Mais pourquoi ne pas demander à l'ordinateur de vous donner les valeurs de FOND et de TEXTE pour que vous puissiez les noter ?

Cette ligne fera le travail.

→ 101 PRINT «FOND = » ; FOND, «TEXTE = » ; TEXTE ↵

Comme vous le verrez à l'exécution du programme, l'ordinateur affichera FOND = parce que cette expression est située entre guillemets, puis juste à côté (à cause du point-virgule (;) situé après les guillemets) il vous donnera la valeur de la variable FOND.

Pour plus de précisions, retenez qu'après l'instruction PRINT, tout ce qui se situe entre les guillemets est affiché sur l'écran sans être interprété.

Par contre, ce qui n'est pas entre guillemets est considéré comme une variable.

Une virgule est utilisée après le nom de variable FOND au lieu d'un point-virgule et ce qui suivra sera affiché au début du champ suivant.

▶ Chaque ligne de l'écran est divisée en 5 champs contenant 12 caractères. Et c'est pourquoi TEXTE = (qui est aussi situé entre guillemets) n'est pas affiché à côté de la valeur FOND mais commence à 24 espaces du début de la ligne. Le point virgule qui suit fera afficher la valeur de TEXTE juste après le signe égal (=) et non au début du champ suivant.

Si cela vous paraît trop compliqué, essayez de remplacer les virgules par des points virgules et vice versa pour en voir les effets. Comme d'habitude, il est plus facile de comprendre quelque chose en le voyant se dérouler, qu'en nous croyant sur parole.

Poursuivons. Jusqu'à présent le programme fait changer la couleur du FOND et lui donne les 16 couleurs, mais la couleur du texte ne change pas. Pour finir notre programme et pour toutes les combinaisons de couleurs et du texte et du fond, nous devons changer la couleur du texte à chaque fois que le fond a pris toutes les couleurs.

Trois lignes s'en acquitteront :

→ 130 LET TEXTE = TEXTE + 1 ↵

→ 140 IF TEXTE < 16 THEN GOTO 90 ↵

→ 150 COLORT 15 0 0 0 : REM TEXTE NOIR SUR FOND BLANC ↵

La ligne 130 ne sera exécutée que si la condition imposée pour que l'ordinateur revienne à la ligne 100 (à partir de la ligne 120) n'est pas satisfaite, c'est-à-dire quand FOND vaut 16, ce qui indique que nous avons vu toutes les couleurs de 0 à 15 pour le fond et pour cette couleur du texte. C'est alors le moment de changer la couleur du texte et la ligne 130 fait exactement cela.

La ligne 140 arrête la variable TEXTE à 15, pour éviter le message :

NUMBER OUT OF RANGE IN LINE 100

Puis elle renvoie le programme à la ligne 90 où FOND se voit attribuer la valeur 0 de façon à pouvoir parcourir de nouveau les valeurs de 0 à 15.

Quand TEXTE vaudra 16, le programme devrait normalement finir, vous laissant un écran totalement blanc, puisque la dernière instruction COLORT de la ligne 100 a été exécutée avec FOND = 15 et TEXTE = 15 (texte blanc sur fond blanc). La ligne 150 évite cela en imposant un texte noir sur fond blanc, comme il est expliqué sur la ligne elle-même après l'instruction REM.

REM est l'abréviation de REMarque.

▶ Tout ce qui suit un REM dans un programme, n'est pas interprété et sert uniquement à expliciter certaines instructions. Les ordinateurs ignorent les REM en cours d'exécution de programme mais les impriment sur le listing des programmes. Ceci est bien pratique pour vous rappeler le pourquoi de telle ou telle instruction lorsque vous relirez votre programme longtemps après l'avoir écrit.

▶ Remarquez aussi à la ligne 150 que 2 points (:) séparent 2 instructions sur une même ligne. Il est évidemment possible de le faire, mais nous vous prévenons que toute interprétation d'un programme devient difficile s'il y a plus d'une instruction par ligne, sauf bien sûr s'il s'agit d'un REM.

Mettre plusieurs instructions sur une seule ligne permet d'économiser et/ou de les faire tourner plus rapidement. Nous avons enfin un programme qui fait tout ce que nous voulions. LISTons-le dans sa forme définitive.

```
80   LET TEXTE = 0
90   LET FOND = 0
100  COLORT FOND TEXTE 0 0
101  PRINT «FOND = » ; FOND, «TEXTE = » ; TEXTE
105  WAIT TIME 100
110  LET FOND = FOND + 1
120  IF FOND < 16 THEN GOTO 100
130  LET TEXTE = TEXTE + 1
140  IF TEXTE < 16 THEN GOTO 90
150  COLORT 15 0 0 0 : REM TEXTE NOIR SUR FOND BLANC
999  END
```

Faites RUN encore une fois pour voir fonctionner le programme et pour décider de la meilleure combinaison de couleurs texte/fond.

Si vous pensez avoir compris le fonctionnement de ce programme, vous êtes sur le bon chemin pour programmer vos propres applications.

VIII - POUR SAUVEGARDER ET CHARGER LES PROGRAMMES

C'est maintenant qu'il vous faut utiliser votre magnétophone à cassettes.

Le raccordement ayant été fait comme il a été indiqué plus haut, vous allez pouvoir sauvegarder vos programmes.

Avant de passer à d'autres choses intéressantes, apprenez à sauvegardez (SAVE) vos programmes en vue d'un usage ultérieur. Rien n'est plus simple, mais faisons-le pas à pas :

1) vous devez bien sûr avoir un programme en mémoire, et si vous avez bien suivi ce manuel, vous devez en avoir un.

retour

2) Mettez la cassette qui va servir à sauvegarder votre programme dans le magnétophone raccordé au D.A.I. et préparez-vous à enregistrer lorsque l'ordinateur vous l'indiquera.

D'une manière générale, réglez la tonalité au plus aigu et le son au plus fort.

3) Donnez un nom au programme, cela vous permettra de ne charger (LOAD) que ce programme parmi ceux qui sont sur la cassette. Appelons ce programme : «PREMIER»

4) Tapez :

→ SAVE «PREMIER» ↵

Après RETURN l'ordinateur répondra :

SET RECORD, START TAPE, TYPE SPACE
(Préparez le magnétophone, mettez-le en marche, appuyez sur la barre-espace).

Faisons ce qu'il nous dit...

5) Quand le signe (*) sera revenu, vous saurez que l'ordinateur a fini d'enregistrer le programme sur la cassette.

Vous devriez maintenant recommencer les opérations 4 et 5 pour sauvegarder le programme une seconde fois après le 1er enregistrement. C'est une sécurité. Enfin, pour être tout à fait sûr de l'enregistrement, il est possible de vérifier (CHECK) que le programme a été enregistré correctement.

Pour ce faire, vous rembobinerez la cassette à son début, puis tapez :

→ CHECK ↵

Mettez en marche (PLAY) le magnétophone.

Si tout est OK, l'ordinateur répondra :

PREMIER OK

Sinon le message sera :

PREMIER BAD
(Premier mauvais)

Si vous obtenez ce message, voici quelques-unes des causes possibles :

- il est nécessaire de nettoyer les têtes de lecture et d'enregistrement.
- la qualité de la cassette n'est pas assez bonne, nous recommandons des cassettes de bonne qualité.
- le niveau sonore était mauvais pendant l'enregistrement ou la lecture.

Dans le 3e cas il faut recommencer les étapes 4 et 5 avec des réglages du volume différents (en gardant la tonalité au plus aigu) et faire CHECK pour vérifier les enregistrements.

▶ Notez alors le niveau sonore et vous ne devriez plus avoir de problème.

Attention pour sortir de CHECK appuyez sur BREAK 2 secondes.

Lorsque vous serez sûr d'avoir sauvegardé «PREMIER» sur cassette, vous pouvez éteindre puis rallumer l'ordinateur (ce qui vous assure qu'il ne reste aucune trace du programme dans les mémoires) et chargez (LOAD) alors le programme dans les mémoires de l'ordinateur pour voir s'il a bien été sauvegardé sur cassette.

Voici les étapes :

1) Rembobinez la cassette jusqu'au début de l'enregistrement :

2) Tapez :

→ LOAD ↵

3) Appuyez sur RETURN et mettez en marche le magnétophone.

Le programme trouvé, le curseur s'arrêtera de clignoter ; quand il reviendra, vous pourrez demander le LISTing du programme pour vous assurer de son retour. C'est tout...

Maintenant que vous avez quelques notions fondamentales sur votre ordinateur et son langage, il est temps de connaître une des particularités qui distinguent cet ordinateur des autres :

LE GRAPHISME DE COULEUR.

IX - UNE APPROCHE de la RESOLUTION

Comme une photo sur du papier, une image graphique sur un écran est composée de nombreux points. Plus les points sont fins, plus ils sont nombreux sur un espace donné, plus la qualité de l'image est élevée. En termes techniques, on dit qu'une pellicule photographique a une HAUTE RESOLUTION quand le nombre de points qui forment l'image est si élevé que celle-ci ne paraît pas du tout formée de points.

Le D.A.I. vous permet de faire des dessins sur votre écran de télévision avec un choix de 3 niveaux de résolution qui sont :

en basse résolution : 72 points en largeur × 65 en hauteur
en moyenne résolution : 160 points en largeur × 130 en hauteur
en haute résolution : 336 points en largeur × 256 en hauteur

Pour chacun de ces 3 niveaux. vous pouvez choisir soit 4 couleurs parmi les 16, soit les 16 couleurs. Vous pouvez aussi décider d'utiliser tout l'écran pour le graphisme ou laisser de la place pour 4 lignes de texte en bas de l'écran. Vous disposez ainsi de 13 options ou modes si vous comptez le mode 0 qui est pour le texte pleine page.

MODE	DIMENSIONS DU GRAPHIQUE	DIMENSIONS DU TEXTE	COULEURS
0	–	24 × 60	2 parmi 16
1	72 × 65	–	16
1A	72 × 65	4 × 60	16
2	72 × 65	–	4 parmi 16
2A	72 × 65	4 × 60	4 parmi 16
3	160 × 130	–	16
3A	160 × 130	4 × 60	16
4	160 × 130	–	4 parmi 16
4A	160 × 130	4 × 60	4 parmi 16
5	336 × 256	–	16
5A	336 × 256	4 × 60	16
6	336 × 256	–	4 parmi 16
6A	336 × 256	4 × 60	4 parmi 16

Votre machine ayant 48 K – octets (un K-octets = 1024 octets) de mémoire RAM, tous les modes vous sont accessibles. La façon de passer d'un mode à un autre est très aisée. Par exemple, pour sélectionner le mode 1, tapez :

→ MODE 1 ↵

Ainsi vous passez du mode texte au mode graphique basse résolution 16 couleurs. Remarquez que le bas de l'écran est toujours disponible pour afficher 4 lignes de texte.

Si l'on se réfère au tableau ci-dessus, il ne devait pas en être ainsi puisque c'est le MODE 1A et non le MODE 1 qui réserve l'affichage de 4 lignes de texte. Toutefois, il est indispensable de pouvoir afficher les instructions que vous rentrez dans l'ordinateur.

C'est pourquoi, la possibilité d'utiliser un MODE sans texte ne sert qu'au cours du déroulement d'un programme (et lorsque vous ne tapez plus d'instructions). Tapez par exemple :

→ 10 MODE 1 ↵
→ 20 GOTO 20 ↵
→ RUN ↵

Vous voyez ! Maintenant tout l'écran est utilisé pour le graphisme et non pour des caractères alphanumériques. Ce programme ne s'arrêtera jamais de lui-même puisque la ligne 20 le fait BOUCLER CONTINUELLEMENT. Arrêtez-le en appuyant sur BREAK et l'espace graphique sera décalé vers le haut pour laisser place à 4 lignes de texte.

Si vous aviez utilisé ce programme pour dessiner un magnifique paysage, vous pourriez penser que le haut de votre chef-d'œuvre a disparu. Et bien, non ! Quand le D.A.I. fait de la place au texte, la partie supérieure du dessin glisse simplement dans une mémoire qui n'est pas affichée sur votre écran. Votre dessin sera facilement redécalé vers le bas par une simple «astuce» détaillée plus loin.

Tous les exemples qui vont suivre seront réalisés à l'aide du MODE de résolution le plus bas, mais s'adaptent parfaitement à tous les MODES.

LA HAUTE RESOLUTION

Mais quelle est cette histoire de 4 couleurs et de 16 couleurs ?

Avant de penser que vous avez perdu 12 couleurs en quittant le magasin, laissez-nous vous expliquer ce qui se passe. Vous n'allez peut être pas comprendre parfaitement tout ce qui va suivre, et ce n'est pas essentiel pour commencer à faire du graphisme, mais il est important de savoir pourquoi il y a quelques restrictions dans l'utilisation du graphisme en couleur. (Une description complète du système graphique est donnée dans la 2e partie du manuel).

En substance, ces restrictions sont dues au fait que le système adopté vous permet de travailler avec 16 couleurs en haute résolution avec LA MOITIE DE LA MEMOIRE qui aurait été nécessaire pour travailler sans restriction avec les 16 couleurs. Nous pensons avoir adopté la seule approche pratique permettant à un petit système d'avoir un graphisme si spectaculaire.

Pour afficher des points de couleurs sur l'écran, il existe dans le D.A.I. un circuit électronique spécial que nous appelerons «circuit vidéo» ; ce circuit balaye continuellement un compartiment de la mémoire RAM du D.A.I. pour savoir la couleur de chaque point sur l'écran. Dans cette partie de RAM (connue sous le nom de «rafraîchisseur de mémoire-écran») est stockée une «image» des points de l'écran. Plusieurs fois par seconde le circuit vidéo «peint» l'image sur l'écran en utilisant le modèle en mémoire.

Prenons un exemple. Supposons que nous ne travaillons qu'avec deux couleurs ; vous savez peut-être que la plus petite unité de mémoire dans l'ordinateur est le bit. Il ne peut avoir que la valeur 0 ou 1. Chaque bit correspond à un point sur l'écran. Par convention le circuit vidéo saura qu'il devra afficher un point d'une couleur (par exemple la couleur du fond) si le bit en mémoire correspondant à ce point sur l'écran est 0. Chaque fois qu'il rencontre un bit de valeur 1, notre peintre électronique affichera le point correspondant avec la couleur demandée que nous appelerons «couleur du premier plan». Le choix des 2 couleurs n'a pas d'importance mais vous serez toujours limité à ces deux-là.

Ce que nous venons d'expliquer est le procédé qu'utilise effectivement la plupart des systèmes fonctionnant en noir et blanc.

Comme vous pouvez le constater, le nombre de bits de mémoire correspond exactement au nombre de points sur l'écran. Dans ce cas, pour afficher 86 016 points (336 × 256) il vous faut 86 016 points de mémoire RAM ; pour exprimer ce nombre en termes d'octets (1 octet = 8 bits), 10 752 octets seront suffisants pour permettre au circuit vidéo de reconstituer l'image, même en haute résolution. (Le D.A.I. possède 49 152 octets).

Mais nous souhaitons obtenir des points de toutes les couleurs. Or, il est impossible de représenter des nombres plus grands que 1 avec un simple bit. Deux sont nécessaires pour représenter les nombres de 0 à 3 (permettant un choix de 4 couleurs) et 4 pour les nombres de 0 à 15, ce qui permet de demander une des 16 couleurs. Aussi, pour préciser la couleur de chaque point de l'écran, on aurait besoin de 4 bits par point. En haute résolution cela demanderait l'utilisation massive 344 064 bits de mémoire pour les 86 016 points (336 × 256).

En d'autres termes, 43.008 octets de mémoire RAM seront réservés à l'image en couleur. Dans ce cas, même avec le D.AI. qui est fourni avec 48 K (49.152 octets) il ne restera qu'un très petit espace disponible pour vos programmes. Pour réduire cette quantité de mémoire nous devions réduire soit le nombre de couleurs, soit la résolution. Nous avons décidé de garder la haute résolution, donc nous avons dû réduire d'une façon ou d'une autre le choix des couleurs. Nous avons réalisé un système qui, d'une part, divise l'espace mémoire nécessaire en deux, et d'autre part vous permet toujours de travailler avec 16 couleurs, compte tenu de quelques restrictions. Nous disposons actuellement de deux méthodes pour économiser de la place-mémoire. Nous avons pensé que nous devions vous laisser choisir celle qui convient le mieux à votre application.

Les deux façons d'utiliser le graphisme sur le D.A.I. sont les suivantes :

- le MODE 16 couleurs

et

- le MODE 4 couleurs

Nous parlerons du MODE 4 couleurs tout d'abord en gardant à l'esprit que la plupart des instructions que nous allons introduire sont tout aussi valables pour le MODE 16 couleurs.

Dans ce cas, la consommation de mémoire est limitée par la réduction du nombre de couleurs visibles à chaque instant sur l'écran. Deux bits de mémoire sont associés à chaque point (ce qui nécessite 21.500 octets dans la plus haute résolution) et indique au circuit vidéo la couleur de ce point. Cela ne veut pas dire que vos dessins dans le MODE 4 couleurs seront toujours limités aux mêmes 4 couleurs. Vous pouvez déterminer à chaque instant vos 4 couleurs en les choisissant parmi les 16 et en plaçant leur code dans 4 espaces-mémoire spéciaux que nous appelons les REGISTRES COULEURS.

Ainsi, à chaque instant, il ne peut y avoir plus de 4 couleurs sur l'écran. Mais en changeant une nouvelle couleur dans un des registres, tous les points correspondant à ce registre (ce qui est différent d'une couleur prédéterminée) changeront immédiatement de couleur.

Cela signifie que la même image peut-être vue avec une des 16 couleurs, tout à tour, en changeant seulement le contenu d'un registre ou plus.

Ceci peut être utilisé pour des effets intéressants, notamment pour faire de l'animation. (Voir la 2e partie du manuel).

Et puisque, nous l'admettons, tout cela est un peu compliqué, passons à la pratique. Tous les exemples seront construits sur le MODE de résolution le plus bas. Mais vous pouvez utiliser les mêmes exemples dans les résolutions plus élevées, en les sélectionnant avec le code qui leur correspond. Tapez :

→ MODE 2 ⏎

et tout l'écran sera noir, excepté le bas de l'écran réservé aux 4 lignes de texte. La couleur du fond (noir) correspond à la couleur qui se trouve dans le premier des 4 registres-couleurs. Puisque vous ne l'avez pas changé depuis la mise en marche du D.A.I., ce premier contient toujours le nombre 0. Ceci explique pourquoi l'écran est devenu noir et non bleu ou orangé (voyez plus loin la liste des codes-couleurs).

Au moment où vous changez le code du premier registre tout l'écran deviendra immédiatement de la couleur de ce code. Pour avoir un écran bleu, tapez :

→ COLORG 1 0 0 0 ↵

L'instruction DOT

Maintenant tapez :

→ DOT 5,5 14 ↵

et l'ordinateur répondra :

COLOR NOT AVAILABLE
(couleur non disponible)

Pourquoi ? Parce que vous avez demandé de placer un point jaune (DOT) sur l'écran, mais le jaune (code 14) n'est pas stocké dans un des 4 registres-couleur et n'est donc pas disponible.

Essayez plutôt cela :

→ DOT 5,5 0 ↵

Et vous aurez alors un point noir sur l'écran (bien sûr, il ressemble plus à un carré qu'à un point, mais souvenez-vous que vous utilisez pour l'instant le mode de résolution le plus bas du D.A.I.). La position du point est déterminé par les 2 nombres séparés par une virgule (5,5). Si vous vous souvenez de vos cours de géométrie, vous devez savoir que 5,5 sont respectivement les coordonnées X et Y de la position d'un point, l'origine 0,0 a été située dans le coin inférieur gauche de l'espace réservé au graphisme.

D'autre part, l'écran a été divisé en un certain nombre de colonnes verticales et de rangées horizontales.

DOT 5,5 indique à l'ordinateur de placer un point à l'intersection de la 5e colonne en partant de la gauche de l'écran et de la 5e rangée en partant du bas. La couleur du point est indiquée par le nombre qui suit les coordonnées. Pour être utilisable, le code couleur DOIT avoir été stocké dans un des 4 registres-couleurs.

L'instruction COLORG

Comment charger ces registres ? C'est facile.

Pour charger les registres avec le jaune (14), le bleu (1), le vert (5) et le blanc (15), tapez :

→ COLORG 14 1 5 15 ↵

L'écran est devenu jaune puisque le premier registre contient le numéro 14. Vous devez aussi voir un point aux coordonnées 5,5. C'est le point qui était noir quand l'écran était bleu avant que vous ne changiez les registres. Maintenant ce point est bleu parce que vous avez chargé le bleu (1) dans le second registre qui contenait précédemment le noir (0).

Pour mettre un point vert dans le coin inférieur gauche de l'écran, tapez :

→ DOT 0,0 5 ↵

Vous pouvez maintenant mettre un point blanc à côté du point bleu en tapant :

→ DOT 5,6 15 ↵

Essayez maintenant de mettre un point bleu à droite du point blanc. Nous ne vous donnerons pas la réponse cette fois-ci. Faites-le, même si cela doit vous prendre 2 heures !

C'est en programmant que l'on devient programmeur...

XMAX et YMAX

Comment mettre un point dans l'angle inférieur droit ?

Vous pourriez regarder sur la table des modes pour savoir le nombre maximum de colonnes de ce mode. Puisque vous êtes en mode 2 vous trouverez que c'est 71 (oui, il y a 72 points, mais souvenez-vous qu'ils sont numérotés de 0 à 71 et non de 1 à 72). Aussi, mettez un point bleu dans ce coin en tapant :

→ DOT 71,0 1 ↵

Toutefois, il existe un moyen pour connaître la valeur maximum des colonnes et des rangées pour chacun des 3 modes de résolutions. Remplacez le nombre actuel (71) par XMAX pour MAXimum (à droite) ou YMAX pour MAXimum (vers le haut).

C'est important, car cela vous évite non seulement d'avoir à vérifier les 6 valeurs, mais aussi vous permet d'écrire des programmes qui ne dépendent pas du mode de résolution choisi au moment de l'exécution.

Quelques exemples :

→ DOT XMAX, 0 14 ↵

ceci supprimera le point bleu de droite qui sera recouvert par un point jaune de la même couleur que le fond. Pour placer un point au centre de l'écran, quelque soit le mode de résolution, vous pouvez taper :

→ DOT XMAX/2, YMAX/2 5 ↵

Le point vert qui vient d'apparaître n'est pas vraiment au centre de l'écran, n'est-ce pas ? Ceci est dû au fait que, comme nous l'avons expliqué plus haut, pour vous permettre de voir 4 lignes de texte en bas de l'écran, l'espace graphique a glissé vers le haut pour permettre de taper les instructions qui doivent être exécutées directement (mode direct) et non durant l'exécution d'un programme. Nous vous avions dit qu'il existait une «astuce» pour récupérer l'image dans sa totalité. La voici, tapez :

→ 60000 MODE 2 ↵
→ 60010 GOTO 60010 ↵
→ RUN 60000 ↵

(Remarquez que vous pouvez ordonner au D.A.I. de commencer l'exécution d'un programme à n'importe quelle ligne).

Vous avez maintenant un écran totalement réservé au graphisme et le point vert est au centre de l'écran. L'astuce consiste simplement à faire exécuter un programme «idiot» qui restitue le mode dans lequel votre image a été construite (vous devrez changer la ligne 60000 pour les autres modes) puis boucle indéfiniment (ligne 60010) pour ne pas reprendre le contrôle de la machine (avec l'espace au bas de l'écran). Le programme pourrait avoir n'importe quelle numéro de ligne cependant, en le plaçant à 60000, vous vous assurez qu'il n'interférera pas avec le vrai programme actuellement en mémoire (ces numéros de lignes ne sont pas courants). Si vous êtes lassé de regarder un écran jaune parsemé de points verts, appuyez donc sur BREAK et le dessin glissera vers le haut pour faire à nouveau place aux 4 lignes de texte.

Mais vous disposez de deux autres instructions d'aide au dessin :

L'instruction DRAW

Pour obtenir une ligne horizontale bleue qui aille du point défini par la colonne 0 (à gauche) et par la 10e rangée, au point défini par la même rangée et la dernière colonne (XMAX), tapez :

→ DRAW 0,9 XMAX, 9 1 ↵

ou vous pouvez couper l'écran en diagonale en faisant :

→ DRAW 0,0 XMAX, YMAX 5 ↵

En d'autres termes, pour tracer une ligne, vous tapez le mot DRAW (dessin) suivi d'un espace, puis de la position du point de départ de la ligne (défini comme dans DOT), puis un espace, puis la position du point où doit se terminer la droite. Après un autre espace, vous tapez le code de la couleur choisie pour la ligne.

L'instruction Fill

Pour remplir un carré ou un rectangle sur l'écran avec une couleur, vous pouvez le faire avec l'instruction FILL (remplir). Si par exemple, vous vouliez remplir un carré vert dont un des angles est en 7,7 et l'autre (en diagonale) en 20,20 faites :

→ FILL 7,7 20,20 5↵

essayez les diverses commandes. Faites des figures avec des points, des lignes et des rectangles. Essayez en haute résolution en ne prenant que les MODES en 4 couleurs. Si vous voyez apparaître :

SYNTAX ERROR

C'est que vous avez mal écrit une instruction (attention aux espaces).

Si vous voyez apparaître :

OFF SCREEN
(hors de l'écran)

C'est que vous avez adressé un point en dehors de l'écran, par exemple supérieur à XMAX.

Quand vous serez familiarisé avec ces instructions, continuez la lecture de ce manuel pour l'utiliser en 16 couleurs.

X - LE MODE 16 COULEURS

Dans ce MODE il est possible d'obtenir 16 couleurs simultanément. La seule limitation (pour laisser de la mémoire disponible) est que vous ne pouvez avoir 16 points de 16 couleurs différentes les uns à côté des autres.

Voici comment fonctionne le système :
chaque ligne horizontale est divisée en segments de 8 huit points, selon la résolution, on a 9, 20, ou 42 segments par ligne horizontale.

Dans chacun de ces segments on ne peut utiliser que *2 couleurs à la fois.*

On peut donc mettre 84 couleurs sur une ligne.

Dans ce graphisme bi-couleur (par champ ou segment) les points sont affectés d'un bit de couleur qui indique suivant sa valeur (1 ou 0) la couleur au circuit vidéo. Pour chaque segment on a donc 2 couleurs disponibles parmi les 16. Au lieu d'adopter un système de registre où vous stockez le code des couleurs, comme dans le MODE 4 couleurs, dans le MODE 16 couleurs, chaque champ a 2 registres indépendants (2 octets sont réservés en mémoire par champ).

Le premier indique la répartition des 2 couleurs dans le segment ; et le 2e choisit ses 2 couleurs parmi les 16 possibles (les couleurs sont codées sur 4 bits). Dans le mode 16 couleurs, vous n'avez donc pas besoin de les choisir auparavant comme dans le mode 4 couleurs, puisque la sélection se fait automatiquement pour chaque point.

Quand vous vous mettez en MODE 16 couleurs, le fond est d'une seule couleur (celle du premier registre du MODE 4 couleurs). Cela veut dire que dans chaque champ les bits du choix de couleur (0 ou 1) ont la valeur correspondant à la couleur du fond. Maintenant, vous pouvez placer un 1er point de n'importe quelle couleur où vous voulez. Vous pouvez aussi placer des points de couleurs différentes, au-dessus ou au-dessous du premier point. Essayez.

Voici un dessin qui vous aidera à comprendre ce qui est autorisé.

POSSIBLE

X	X	O	O	X	O	O	O

IMPOSSIBLE

X	O	O	□	□	X	X	X

1 champ d'écran

X	O	X	O	X	X	X	X
O	X	O	O	O	O	X	X
X	□	X	X	X	□	□	□

1 champ d'écran

POSSIBLE

Dans un champ, vous ne pouvez mettre une 3e couleur. La couleur du fond correspond à la couleur codée dans la première moitié du 2e octet affecté au champ correspondant. Ce système est restrictif mais nécessaire si vous voulez qu'il reste de l'espace mémoire pour programmer.

Malgré ces limitations, vous pourrez sûrement créer de beaux graphismes en 16 COULEURS. Pour vous exercer, utilisez toutes les instructions du MODE 4 couleurs. Le seul effet de COLORG en MODE 16 couleurs sera, que le code, chargé dans le premier registre, déterminera la couleur du fond quand vous sélectionnerez un MODE 16 couleurs pour la première fois, sinon il n'aura aucun effet.

S'il manque un point ou une partie de ligne, souvenez-vous que cela est dû au système des champs et non à une erreur de programme ou à une panne d'ordinateur. Mettez-vous en mode 16 couleurs en basse résolution, tapez :

```
MODE 1
```

Puis, faites les autres modes.

retour

LISTE DES COULEURS

0 NOIR
1 BLEU VIF
2 PARME
3 ROUGE VIF
4 VERT KAKI
5 VERT PRE
6 ROUGE FONCE
7 VIOLET
8 GRIS
9 BLEU MARINE
10 ORANGE
11 ROSE SAUMON
12 BLEU CLAIR
13 VERT CLAIR
14 JAUNE
15 BLANC

XI - BREVE DIGRESSION

retour

A. L'EDITEUR

1) La réalisation de programmes nécessite toujours de nombreuses modifications. Aussi, pour vous y aider, votre D.A.I. possède un EDITEUR.

– Charger en mémoire un programme (cf. LOAD) puis tapez : E D I T ↵

EDIT met dans la mémoire de l'EDITeur tout le programme pour d'éventuelles modifications.

EDIT 100 ne met dans l'éditeur que la ligne 100.
EDIT-100 met dans l'éditeur le début du programme jusqu'à la ligne 100.
EDIT 100 – met dans l'éditeur le programme à partir de la ligne 100 jusqu'à la fin.
EDIT 100 – 1300 met dans l'éditeur le programme à partir de la ligne 100 jusqu'à la ligne 1300.

Vous êtes maintenant sous éditeur :
– un trait se trouve à la gauche du texte ainsi que le signe ↵ à l'extrême droite de toutes les lignes (représentation symbolique du RETURN)
– aucun ordre ne peut être donné à l'ordinateur, vous n'êtes plus sous BASIC
– à l'aide des touches ↓ et REPT le curseur se déplace vers le bas. Arrivé au bas de l'écran (au cas où le programme fait plus de 24 lignes) votre programme défile progressivement, le curseur s'arrêtant à la fin. Retournez au début avec les touches ↑ et REPT
– sous éditeur les lignes de programmation qui dépassent 60 caractères n'apparaissent pas entièrement sur l'écran, aussi pour les visualiser, utilisez les touches → et REPT pour déplacer le curseur. Arrivé à l'extrême droite de l'écran, tout votre texte glissera vers la gauche, faisant apparaître progressivement la suite des lignes. Retournez au début à l'aide des touches ← et REPT
– les déplacements en diagonale sont obtenus en appuyant simultanément sur les touches → ↑ et REPT (soit toute autre combinaison dans les quatre directions).
– Pour obtenir seulement un mouvement de page, utiliser simultanément les touches ↑ REPT SHIFT.

2) Pour modifier votre programme, l'éditeur offre les possibilités suivantes :

INSERTION :

ex : 10 PRINT «DAI COMPUTER» ↵

En se positionnant sur le «C» et en tapant PERSONAL vous obtenez :

10 PRINT «DAI PERSONAL COMPUTER» ↵

EFFACEMENT : pour effacer des instructions, positionnez-vous sur le 1er caractère de la chaîne à effacer et appuyez sur les touches DEL CHAR et REPT.

La chaîne de caractères *à droite* du curseur se décale vers la gauche en disparaissant au niveau du curseur.

Contrairement au mode direct, la touche DEL CHAR efface le *caractère clignotant*, au lieu du caractère précédent.

3) Quatre propriétés intéressantes de l'éditeur sont obtenues grâce au caractère ↵ (RETURN)

a) Chaînage de lignes
ex : 10 FOR I = 1 TO 10↵
20 PRINT «DAI» ↵
30 NEXT ↵
Positionnez-vous sur ↵ de la ligne 10. Tapez DEL CHAR, vous obtenez :
10 FOR I = 1 TO 1020 PRINT «DAI» ↵
30 NEXT
Effacez «20» en tapant deux fois sur DEL CHAR, et mettez «:»
Vous venez de faire du chaînage
C'est une propriété intéressante de l'éditeur ; en mettant un maximum d'instructions sur une ligne, vous gagnez de la place-mémoire et une rapidité d'exécution du programme appréciable.

▶ b) Séparation d'instructions.
C'est l'inverse de la propriété précédente.
ex. : 10 FOR I = 1 TO 10 : PRINT «DAI» : NEXT ↵
Positionnez-vous sur le 1er «:», tapez RETURN.
Vous obtenez :
10 FOR I = 1 TO 10↵
: PRINT «DAI» : NEXT ↵
Tapez 20 et DEL CHAR pour effacer les deux points.

▶ Cette propriété permet, par exemple, de récupérer des instructions après un IF. THEN alors qu'elles ne devaient pas l'être.

c) Création d'espace.
Positionnez-vous sur un caractère↵ et tapez RETURN.
Vous venez de gagner une ligne. L'opération peut se répéter.

d) Suppression de lignes entières.
Tapez un RETURN après un numéro de ligne, et celle-ci sera supprimée lors de l'interprétation du programme, en renvoyant un «SYNTAX ERROR» dont il ne faut pas tenir compte.

▶ Notez qu'après un ↵ aucun caractère ne peut être marqué.

4) Une fois vos modifications effectuées, deux possibilités s'offrent à vous :

a) Vous voulez enregistrer vos modifications.

Appuyez sur BREAK puis sur la barre d'espacement ; le retour du signe * signifie que votre programme est enregistré avec ses modifications.

b) Vous avez effacé des instructions ou des données par erreur et vous tenez à les récupérer :
- Appuyez deux fois sur BREAK : aucune de vos modifications n'a été enregistrée.
- Tapez LIST ou EDIT pour le vérifier.

B. LIST

retour

Exemples :

LIST — Affiche le programme en entier à partir du début. Les lignes comportant plus de 60 caractères apparaissent par sections de 50 caractères disposées les unes au-dessous des autres et commençant par «C».
Pendant l'affichage une pause peut être marquée en appuyant sur n'im-

porte quelle touche. En appuyant sur la barre d'espacement le listing continue.

LIST 100	Affiche la ligne 100
LIST 100	Affiche le programme de la ligne 100 à la fin
LIST-100	Affiche le programme du début jusqu'à la ligne 100
LIST 100-130	Affiché le programme de la ligne 100 à la ligne 100 à 130

retour

C. NEW

En tapant NEW vous effacez le programme qui est en mémoire et vous remettez à zéro toutes les variables.

CONCLUSION

Maintenant, vous avez encore beaucoup de choses à découvrir a propos du D.A.I. Si vous avez bien suivi cette première partie du manuel en essayant tous les exemples, vous devriez pouvoir comprendre les explications plus détaillées et plus techniques du 2e manuel.

Inspirez une bonne fois, tournez la page et plongez-vous dans le second manuel pour découvrir toutes les possibilités de votre machine...

MANUEL D'UTILISATION
2e Partie

Table des matières

1. Description générale

2. Le basic résident du D.A.I.

3. Les commandes utilitaires du moniteur

7. Les entrées/sorties

8. Les facilités du système D.A.I.

1. Description générale

retour

Le micro-ordinateur D.A.I. a été conçu pour offrir le maximum de possibilités pour un coût accessible au grand public. Ce projet a été réalisé de telle façon que l'utilisateur puisse charger un programme à partir d'un minicassette ou d'un floppy-disque. Le résultat de l'exécution d'un programme est visualisé sur une télévision couleur via la prise péritel. Le générateur de son utilise le haut-parleur du téléviseur mais peut aussi être branché en stéréo sur un amplificateur de chaîne HI-FI (entrée : AUX).

Les possibilités du DAI sont réparties en quatre chapitres : la partie micro-ordinateur, le graphisme programmable, le générateur de son et les entrées/sorties. La figure n° 1 montre le bloc-synoptique de la logique du micro-ordinateur DAI. (Pour ne pas trop rentrer dans les détails, nous considèrerons que la gestion logique du système existe déjà dans la machine).

Le programme résidant en ROM est composé de six modules ; l'interpréteur BASIC, la partie calcul arithmétique, le contrôleur d'écran, la scrutation du clavier, le moniteur et le programme principal qui coordonne les différents modules entre eux.

L'interpréteur basic comporte bien plus de possibilités et de caractéristiques que la plupart des micro-ordinateurs actuels ; il offre, par exemple,
- des instructions spéciales pour contrôler l'écran et le générateur de son,
- l'interactivité avec des programmes écrits en langage machine, ou bien encore
- l'édition du programme BASIC source, en mode insertion permanente, avec gestion du déplacement du curseur. Un des principaux objectifs était de disposer d'un basic puissant et rapide. Lorsque l'utilisateur rentre une ligne de programme, le système commence à l'interpréter dès que la touche RETURN a été appuyée (ce qui permet d'avoir une détection immédiate des erreurs de syntaxe), puis stocke en mémoire des codes permettant une interprétation rapide lors de l'exécution.

Le module «calcul arithmétique» est divisé en deux parties ; les calculs sur les nombres entiers et les calculs sur les nombres réels (à virgule flottante). Les calculs autorisent toutes les fonctions mathématiques courantes comme sinus, cosinus, tag. log. La résolution pour les nombres entiers est de neuf chiffres significatifs et de six chiffres pour les nombres réels ; ceux-ci sont compris entre $\pm\ 10^{-11}$ et $\pm\ 10^{18}$. Lorsque le circuit processeur arithmétique (9511) est placé sur son support, les routines de calcul l'utilisent automatiquement et deviennent plus rapides.

Le module contrôleur d'écran s'occupe de l'arrangement des données en vidéo-RAM pour permettre une image correcte dans tous les modes. Il se charge aussi des changements de couleurs, des possibilités graphiques (DOT, DRAW, FILL) relatives à l'écran.

Le clavier du micro-ordinateur DAI est du type matriciel (8×7) composé de 56 touches. La scrutation du clavier intervient à intervalles réguliers et détecte si une touche est appuyée. Le clavier a été conçu de telle façon qu'il est possible de changer sa configuration et son codage.

Le moniteur langage machine permet, par l'intermédiaire du clavier, d'appeler des sous-programmes «utilitaires» gérant l'écriture, la visualisation et la mise au point de programmes en langage machine. L'interactivité entre le basic et les programmes machines est totale. Un nombre illimité de programmes-machine peut être appelé par un programme BASIC.

Le programme principal coordonne tous ces différents modules entre eux et contrôle les accès mémoires. Ceci permet, avec un microprocesseur 8080, de travailler sur 72 K octets alors que le bus d'adresse ne comporte que 16 bits (soit 64 K octets adressables).

1.1 Sommaire des caractéristiques

1.1.1. Le micro-ordinateur

- microprocesseur 8080A à 2 MHz
- 48 K octets de mémoire RAM (dynamique)
- 24 K octets de ROM
- entrées/sorties banalisées
- possibilité d'un processeur arithmétique (9511)
- circuit générateur de nombres aléatoires
- logique de détection de débordement de la pile.

1.1.2. Les entrées/sorties

- un clavier de 56 touches
- une prise péritel pour l'utilisation d'un téléviseur comme unité de visualisation
- deux interfaces pour minicassettes avec contrôle des moteurs
- sortie du son stéréophonique sur prise Hi-Fi
- entrée droite et gauche pour 2 paddles de jeu (6 contrôles) : ces deux entrées sont des convertisseurs analogiques/digitaux de 8 bits chacun
- bus d'interface (DCE BUS) pour :
  - floppy disques
  - imprimante parallèle
  - cartes industrielles de chez DAI
    - adaptateur IEEE. BUS
    - programmateur d'EPROM
    - entrées/sorties analogiques
    - etc...
- RS 232 standard : baud Rate programmable pour imprimante série, modem, table traçante, etc. et tous les périphériques susceptibles de fonctionner avec cette interface.

1.1.3. La Vidéo

- 60 caractères par 24 lignes en mode alphanumérique (0)
- 16 couleurs
- 3 résolutions graphiques

72 × 65	soit 4 680 points adressables
160 × 130	soit 20 800 points adressables
336 × 256	soit 86 016 points adressables

* plus des modes intermédiaires permettant de combiner graphiques et caractères.

1.1.4. Le son

- 3 générateurs de fréquence programmables
- 1 générateur de bruit programmable
- sélection par programme de l'amplitude et de la fréquence
  - fréquence aléatoire
  - génération d'enveloppe,
  - synthèse vocale

1.1.5. Programmes résidents

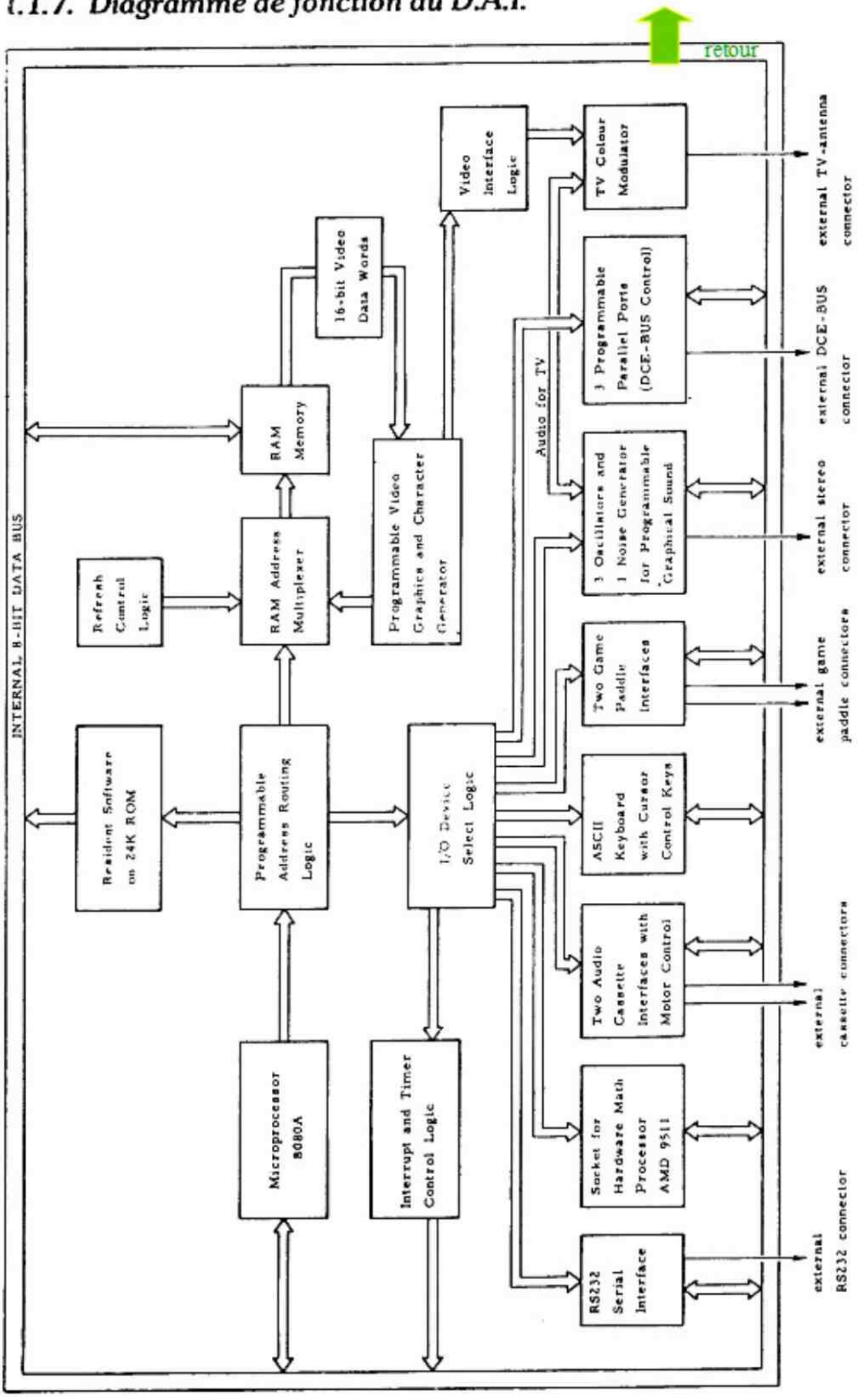
- BASIC interprété (semi-compilateur)
- les algorythmes de calculs scientifiques
- détection automatique du processeur arithmétique
- commande de dessin – point – droite – rectangle en fonction de coordonnées
- commande de son
  - spécification de l'enveloppe volume
  - spécification de la fréquence
  - spécification du volume

  - spécification de l'utilisation du tremolo
  - spécification de l'utilisation du glissando
- le moniteur-langage machine.

1.1.6. programmes compatibles

- DAI Assembleur
- programmes machines en 8080A
- programmes pour MDS/Intellec
- programme compilateur fortran.

1.1.7. Diagramme de fonction du D.A.I.



2. Le BASIC résident du D.A.I.

2.1. Liste alphabétique des instructions du BASIC

2.1.1. Les commandes et instructions BASIC

retour

2.1.2. Les fonctions et les opérateurs BASIC

2.2. Les caractéristiques du BASIC du DAI

2.2.1. L'écran

Lorsque le micro-ordinateur D.A.I. affiche le message «BASIC V1.1» suivi de l'étoile, l'écran est défini en mode 0 c'est-à-dire avec 24 lignes de 60 caractères alphanumériques chacune.

• Vous pouvez, à tout moment, redéfinir l'écran en mode 0 en tapant la commande «MODE 0».

La position du prochain caractère est visualisée par un curseur représenté par un trait clignotant.

Chaque ligne de l'écran peut contenir 60 caractères mais si un caractère dépasse la 60e position, la ligne de programme est augmentée d'une ligne supplémentaire pour continuer. Une ligne peut avoir jusqu'à 3 lignes supplémentaires.

Ces lignes sont facilement identifiables par la lettre C (comme Continu) sur la colonne 0. Le premier caractère supplémentaire est inscrit 7 «espaces» plus loin (soit sur la colonne 7).

Le curseur se déplace vers la gauche lorsqu'un caractère est entré et vers la droite pour le caractère DELCHAR (code hexa 8, #8) provoquant l'effacement du dernier caractère. La touche RETURN (code #C) termine une ligne et repositionne le curseur à la ligne suivante derrière une étoile. La touche BREAK (code #D) permet de ne pas valider les caractères tapés et repositionne le curseur comme pour RETURN. En appuyant sur BREAK on obtient un «!». Le caractère «TAB» (code #9) n'a pas de fonction mais peut-être utilisé par programme.

Lorsque la 3e ligne supplémentaire est remplie, l'écran ignore tous les caractères rentrés, excepté RETURN, DELCHAR, BREAK. Le BASIC accepte toutes les commandes (ou lignes de programme) comprises entre l'étoile et un RETURN. Si l'étoile avait été effacée par un DELCHAR, les caractères tapés seraient ignorés et provoqueraient un message d'erreur.

Les caractères effacés par DELCHAR sont ignorés.

Les couleurs des caractères sont initialisées à la mise sous tension. Elles peuvent être changées par la commande «COLORT».

2.2.2. Entrée de données par programme

Lorsque le système attend des caractères, il affiche normalement une étoile «*», mais pendant l'exécution de l'instruction «INPUT» le caractère affiché est un point d'interrogation «?». L'utilisateur peut alors rentrer les caractères qu'il désire.

Pour effacer le dernier caractère tapé, utilisez la touche «DEL CHAR». Si les caractères rentrés dépassent le bout de la ligne, une ligne supplémentaire sera créée. 3 lignes supplémentaires peuvent utilisées. Ceci donne une longueur totale de 59 + 53 + 53 + 53 = 218 caractères.

La touche BREAK provoque l'affichage d'un «!» et la ligne est ignorée ; cependant, pendant l'exécution de l'instruction «INPUT», la touche BREAK provoque l'arrêt du programme.

2.2.3. L'écriture, la modification et l'exécution d'un programme

Lorsque le D.A.I. est prêt à recevoir une commande, ou l'écriture d'une ligne de programme, le système affiche une étoile. L'utilisateur peut alors taper une ligne d'une ou plusieurs commandes BASIC séparées par «:» et terminées par RETURN. Les commandes seront décodées et, si elles sont justes, exécutées immédiatement.

Si les commandes sont précédées par un nombre, la ligne est décodée et stockée dans la mémoire – programme suivant son numéro. Si une ligne avait déjà été enregistrée sous ce numéro, l'ancienne ligne serait détruite par la nouvelle.

Les commandes BASIC précédées par un numéro sont appelées «lignes de programme».

Les commandes BASIC pouvant être écrites dans une ligne de programme sont appelées des instructions. Par exemple, RUN est une commande mais PRINT est une instruction.

Si la syntaxe de la ligne est incorrecte, un message d'erreur apparaît.

S'il s'agissait d'une commande BASIC (sans numéro) la ligne est perdue.

S'il s'agissait d'une ligne de programme, trois étoiles «***» sont insérées entre le numéro de ligne et la première instruction puis les 121 premiers caractères de la ligne sont stockés en mémoire (codification identique à l'instruction REM).

L'exécution de cette ligne erronée provoquera l'affichage du message d'erreur suivant :

«ERROR LINE RUN IN LINE X»

ou X est le numéro de la ligne concernée. Un point d'interrogation est inséré dans la ligne derrière l'erreur détectée.

La commande «RUN» permet d'exécuter à partir de la première ligne un programme stocké en mémoire. La commande «RUN 100» permet d'exécuter un programme à partir de la ligne 100.

Un programme sera exécuté jusqu'à ce qu'une erreur, ou une des instructions suivantes, soient trouvées :

1 - Après l'exécution de l'instruction «END», le programme affiche le message «END PROGRAM» et rend le contrôle au système qui affiche l'étoile. Le programme ne peut être exécuté que par la commande «RUN»

2 - Après l'exécution de l'instruction «STOP», le programme affiche le message «STOPPED IN LINE X», où X est le numéro de ligne contenant l'instruction STOP et rend le contrôle au système (étoile). On dit alors que le programme est suspendu

3 - Si l'enfoncement de la touche **BREAK** est détecté, une des deux solutions suivantes peut se présenter :
a - le message «BREAK IN LINE X» apparaitra, immédiatement suivi de l'étoile. *Le programme est alors suspendu*
b - après une pause, le système répond «*** BREAK». Le programme ne peut être réexécuté que par la commande «RUN» (ce cas est rare).

Lorsqu'un programme est suspendu, il peut être relancé par la commande «CONT». Le programme, alors, continue à partir de la séquence où il s'était arrêté.

Toutefois, les variables changées par l'utilisateur pendant la suspension du programme ne sont pas réinitialisées.

Le fait d'exécuter un autre programme ou d'utiliser les commandes SAVE, LOAD, EDIT, CLEAR ou NEW ne permet plus de relancer le programme suspendu. Si l'utilisateur tape quand même la commande CONT, le message «CAN'T CONT» apparaîtra sur l'écran.

Les commandes RUN, SAVE, LOAD, EDIT, CLEAR et NEW réinitialisent toutes les variables numériques à zéro et toutes les chaînes alphanumériques ($) à 0 caractère.

La commande «NEW» sert à détruire le programme stocké en mémoire. Après cela, il n'y a plus aucune ligne de programme dans le système et la zone mémoire définissant les variables est remise à zéro.

Lorsqu'un programme est suspendu, la commande STEP peut être utilisée pour continuer le programme en PAS A PAS (ligne par ligne). Chaque ligne est affichée sur l'écran et le système attend un **SPACE** (espace) sur le clavier pour l'exécuter et passer à la ligne suivante.

A la mise sous tension, les variables numériques du DAI sont définies comme des nombres réels (virgule flottante), aussi, les noms des variables entières devront être suivis du caractère « % » pour ne pas confondre entiers et réels.

Exemple :

```
A = 1.2
PRINT A
→ 1.2
```

```
A% = 1.2
PRINT A%
→ 1
```

Remarque : les chiffres sont représentés sous la forme américaine c'est-à-dire avec un point à la place d'une virgule.

2.2.4. Réunion de 2 programmes BASIC

Exemple : LOAD : chargement du premier programme
CLEAR FRE : affichage → «NUMBER OUT OF RANGE»
EDIT + BREAK + BREAK
LOAD : chargement du deuxième programme
POKE # 135,2

Remarque : les deux programmes doivent avoir des numéros de ligne différents.

Exemple :

1er programme

```
LIST
10  CLEAR 1000
20  A$ = «BONJOUR»
30  PRINT A$
40  END
```

2e programme

```
LIST
15  B$ = «MESSIEURS»
25  PRINT B$
40  PRINT A$ + B$
50  END
```

programme résultant

```
LIST
10  CLEAR 1000
15  B$ = «MESSIEURS»
20  A$ = «BONJOUR»
25  PRINT B$
30  PRINT A$
40  PRINT A$ + B$
50  END
```

2.2.5. Sélection d'une partie d'un programme

Vous voulez seulement garder la partie du programme contenue entre la ligne 100 et 300 inclus.

N.B. : Nous considérons que votre programme est déjà en mémoire

procédure :

* CLEAR FRE :	affichage → «NUMBER OUT OF RANGE»
* EDIT 100-130	: transfert dans le buffer d'édition les lignes de programme de 100 à 300
BREAK + BREAK	: deux coups sur la touche BREAK
* NEW	: efface le programme en mémoire mais pas le buffer d'édition
* POKE # 135,2	: mise en mémoire du buffer d'édition
* LIST	: maintenant, seul le programme de 100 à 300 est en mémoire.

2.2.6. Comment échanger toutes les variables réelles en entières et toutes les variables entières en réelles.

```
*IMP FPT
*LIST
10  FOR A=0.0 TO 10.0 STEP 1.5
20  PRINT A;« ,»;                    dans ce programme la
30  NEXT                             variable A est REELLE
*RUN
0.0, 1.5, 3.0, 4.5, 6.0, 7.5, 9.0,
*EDIT
```

puis deux **BREAK**

```
*IMP INT
*POKE#135,2
*RUN
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,    maintenant la variable
*LIST                                  A est ENTIERE
10  FOR A=0.0 TO 10.0 STEP 1.5
20  PRINT A;« ,»;
30  NEXT
```

2.2.7. Réunion d'un programme basic et d'un programme en langage-machine

Préparez le MLP/R (Machine Language Programs/Routines) et sauvez-le sur une cassette après le programme basic que vous avez l'intention d'utiliser avec cette MLP/R.

exemple : Sauvez d'abord votre programme BASIC sur cassette puis tapez un programme vous permettant de sauver sur cassette le MLP/R

```
LIST
10  CLEAR 2000
20  DIM A(20.0,20.0)
30  FOR I% = 0 TO 9
40  READ B%: POKE (#2F1 + I%),B%:NEXT
50  SAVEA A «TEST»: STOP
60  DATA #F5, #3E, #FF, #32, #50, #BE, #F1, C9, 0,0
```

Le programme en langage-machine est sous la forme de DATA

N.B. le nombre d'octets réversé en mémoire pour une dimension maximum → est de (256 × 4) 1024 octets. Dans notre exemple, le nombre d'octets réservé pour la variable A est de 21 × 21 × 4 soit 1764 octets.

Le programme BASIC que vous avez l'intention d'utiliser avec le MPL/R doit contenir la séquence suivante :

– Un CLEAR et un DIM avec le même nom de variable et le même dimensionnement que dans le MPL/R.
– Un LOADA avec le même nom de fichier que dans le MLP/R (ici «TEST»)

Exemple de programme BASIC que vous pouviez mettre sur la cassette avant le MPL/R.

```
LIST
10  CLEAR 2000:DIM A(20.0,20.0)
20  LOADA A «TEST»
30  CALLM #2F1
40  STOP
```

Ce programme chargera le MLP/R (instruction LOADA) puis exécutera le MLP/R (instruction CALLM-ligne 30).

Vous pouvez maintenant taper RUN 30 à chaque fois que vous désirez exécuter le MLP/R. Pour information, le MLP/R de l'exemple positionne l'octet de l'adresse BE50 à FF. (Hexa).

Important : lorsque le MLP/R est chargé par l'instruction LOADA, le fait d'exécuter une instruction «CLEAR», «DIM» (avec le même nom de variable que celui utilisé pour le MLP/R) ou «LOADA» (avec la même variable mais un nom de fichier différent) détruit le MLP/R. L'utilisation du mode «EDIT» effacera aussi le MLP/R préalablement chargé.

Lorsque vous avez oublié l'adresse de début du MLP/R, vous pouvez la retrouver par la commande :

```
PRINT HEX$ (VARPTR (A(0,0)))
```

Cette adresse est souvent #2F0 pour la première variable à une seule dimension et # 2F1 pour la première variable à deux dimensions (ceci est vrai lorsque les disquettes ne sont pas utilisées. Le DOS modifie les en-têtes).

2.3. Les commandes de contrôle de programme

2.3.1. EDIT

retour

Exemples de différents formats

I) EDIT : transfert entièrement le programme BASIC dans le buffer d'édition pour d'éventuelles visualisations ou modifications

II) EDIT 100 : transfert la ligne 100 dans le buffer d'édition

III) EDIT 100 – : transfert le programme BASIC de la ligne 100 jusqu'à la dernière ligne dans le buffer d'édition

IV) EDIT 100 – 130 : transfert le programme BASIC de la ligne 100 jusqu'à la ligne 130 dans le buffer d'édition.

V) EDIT – 130 : transfert le programme BASIC de la première ligne jusqu'à la ligne 130 dans le buffer d'édition.

Le mode «EDIT» facilite l'écriture et les modifications d'un programme BASIC. Les 24 premières lignes du programme sont visualisées sur l'écran, le curseur étant placé sur le premier caractère de la première ligne.

Le curseur, en mode EDIT, peut facilement être déplacé sur l'écran par l'intermédiaire des touches ↑, ↓, ←, →. Si vous essayez de sortir le curseur de l'écran, la partie du programme visualisée sur l'écran se déplacera pour garder le curseur toujours visible.

La partie visualisée du programme s'appelle une «fenêtre».

Vous pouvez déplacer cette fenêtre tout le long du programme à l'aide des commandes curseur + la touche SHIFT, le curseur restant à la même place dans le programme (sauf dans le cas où le curseur sort de l'écran).

La touche DELCHAR supprime le caractère pointé par le curseur.

N'importe quel autre caractère tapé est inséré avant la position du curseur, si celui-ci est à GAUCHE d'un RETURN (représenté par ↵) ou du caractère semi-graphique «■».

Aucun caractère n'a d'effet à DROITE d'un RETURN ou de ce caractère semi-graphique, l'éditeur vous permet de travailler avec 255 caractères par ligne mais lors de l'interprétation, chaque ligne devra pouvoir être codée sans excéder 128 caractères par ligne stockée.

La touche BREAK permet de sortir du mode EDIT. Si elle est suivie par un second BREAK, la totalité des modifications est ignorée.

Si elle est suivie par un SPACE, les modifications effectuées sous éditeur sont prises en compte.

Si nécessaire, un message d'erreur apparaît, suivi de l'étoile * signifiant l'attente d'une commande.

La commande EDIT est aussi utilisée pour assembler deux programmes différents ; la commande POKE # 135,2 provoque la réunion du programme BASIC avec le programme existant dans le buffer d'édition.

exemple de procédure
* CLEAR FRE
* EDIT

- modification(s) (ex. : la ligne 20 devient la ligne 200)
- BREAK + BREAK
- POKE # 135,2
- la ligne 20 n'a pas été effacée et la ligne 200 a quand même été créée.

Sans cette procédure, la ligne 20 était effacée.

2.3.2. IMP

Se reporter aux exemples du paragraphe 2.5.1

2.3.3. LIST

I) LIST visualise entièrement le programme BASIC contenu en mémoire. L'enfoncement de n'importe quelle touche arrête le défilement des instructions. Seul SPACE redémarre l'affichage.

II) LIST 100 visualise uniquement la ligne 100

III) LIST 100 – visualise le programme BASIC de la ligne 100 jusqu'à la fin

IV) LIST 100-130 visualise le programme BASIC de la ligne 100 à la ligne 130

V) LIST – 130 visualise le début du programme BASIC jusqu'à la ligne 130.

2.3.4. NEW

Efface de la mémoire le programme BASIC et réinitialise les variables numériques à 0 et toutes les chaînes de caractères ne contiennent aucun élément. La longueur de la zone réservée par l'instruction «CLEAR» n'est pas réinitialisée à zéro.

2.3.5. RUN

I) RUN commence l'exécution à la première ligne du programme BASIC contenu en mémoire

II) RUN 100 commence l'exécution du programme BASIC à partir de la ligne 100. Si la ligne 100 n'existe pas, un message d'erreur apparaît. Notez que RUN 100 ne réinitialise pas les variables.

2.4. Les erreurs

retour

2.4.1. Les différentes erreurs

Lorsqu'une erreur est rencontrée lors de l'interprétation d'une ligne, un message est affiché. Dans certains cas, d'autres informations seront données.

I) Si c'est une erreur de commande BASIC (donc sans numéro de ligne) aucune autre information n'est donnée
ex. : la commande erronée RIUN provoque l'affichage du message SYNTAX ERROR.

II) Si l'erreur se trouve dans une ligne de programme (donc avec un numéro de ligne), trois étoiles sont insérées entre le numéro de ligne et la première

instruction, et un point d'interrogation indique la première erreur trouvée (cette ligne n'est pas perdue).

III) Si l'erreur se trouve dans une ligne de programme en cours d'exécution, un message d'erreur apparaît suivi de «IN LINE» (à la ligne) puis du numéro de la ligne considérée. Remarque : ce sont des erreurs imputables au programme.

Exemple :

```
*10 A = 0
*20 B = C/A
*RUN
DIVISION BY ZERO IN LINE 20.
```

2.4.2. Les différents messages d'erreur

- CAN'T CONT (l'exécution du programme ne peut pas être reprise).
  Le Programme ne peut pas continuer car il n'a pas été interrompu par un BREAK ni un «STOP».

exemple :

```
*
*NEW
*CONT
CAN'T CONT
```

- COLOR NOT AVAILABLE (couleur non disponible).
  La couleur demandée n'a pas été définie dans l'instruction «COLORG» (seulement pour les modes graphiques pairs et différents de zéro)

- COMMAND INVALID (commande non valable).
  Cette commande ne peut pas être utilisée dans un programme BASIC

exemple :

```
*
*10 RUN
*10 ***RUN?
COMMAND INVALID
*
```

- DIVISION BY ZERO (division par zéro).
  Un nombre est divisé par zéro.

- ERROR LINE RUN (ligne erronée non corrigée).
  Apparaît lorsqu'une ligne de programme non corrigée est exécutée.

exemple :

```
10 A%27
10   *** A%?27
SYNTAX ERROR
*RUN
ERROR LINE RUN IN LINE 10
*
*
```

- INVALID NUMBER (nombre non valable).
  Le paramètre donné pour une expression numérique n'est pas un nombre.

exemple :

```
*
*10 A$ = «BON»
*20 B = VAL(A$)
*RUN
INVALID NUMBER IN LINE 20
*
```

A$ aurait dû être une chaine de caractères numériques.

- LINE NUMBER OUT OF RANGE (numéro de ligne hors format)
  Les numéros de ligne doivent être compris entre 1 et 65 535.

- LINE TOO COMPLEX (ligne trop compliquée)
  La ligne est trop compliquée pour être codée sur 128 octets de mémoire.

- LOADING ERROR 0, 1, 2, ou 3 (erreur de lecture cassette).
  Le fichier n'a pu être lu.

0 trouve une erreur de «cheksum» sur le nom du programme
1 mémoire insuffisante
2 trouve une erreur de «checksum» sur le programme
3 trouve une coupure dans la lecture de la cassette (due à la bande ou à l'arrêt du minicassette).

- NEXT WITHOUT FOR (NEXT sans FOR).
  L'instruction NEXT n'est pas précédée par l'instruction FOR. Cette erreur apparaît lors de l'exécution d'un programme.

- NUMBER OUT OF RANGE (nombre hors format).
  Un nombre est utilisé dans un contexte où il est trop grand ou trop petit.

Exemple :

```
*
*10 MODE 9
10   *** MODE 9?
NUMBER OUT OF RANGE
*
```

Les différents modes vont de 0 à 6, pas plus !

- OFF SCREEN (hors de l'écran).
  Les coordonnées du point de l'écran considéré n'existent pas dans ce mode d'affichage.

- OUT OF DATA (dépassement des données).
  Toutes les données ont été lues, pourtant une instruction READ est rencontrée.

- OUT OF MEMORY (dépassement mémoire).
  Il n'y a plus assez de place rnémoire pour continuer le programme.

- OUT OF SPACE FOR MODE (pas assez de mémoire pour ce mode).
  Lorsqu'un programme est exécuté en mode 1 ou 2 et qu'il ne reste plus assez de place mémoire pour passer en mode 0, 1A ou 2A ; (instruction PRINT ou mode 0, 1A, 2A) le programme est détruit et le message d'erreur apparaît.

- OUT OF STRING SPACE (dépassement de mémoire réservée).
  Pas assez de place mémoire réservée pour un dimensionnement de variable (solution CLEAR X).

- OVERFLOW (dépassement de capacité).
  Dépassement de capacité par un nombre.

Exemple :

```
*
*10 A = 99999999999999999999
10   *** A = 999999999999999999?999
OVERFLOW
```

- RETURN WITHOUT GOSUB (RETURN sans GOSUB).
  Une instruction RETURN est rencontrée sans qu'une instruction GOSUB l'ait précédée.
- STACK OVERFLOW (dépassement du pointeur de pile).
  Le stack pointeur a été trop utilisé pendant le programme.

Exemple : plus de 36 GOSUB imbriqués.

- STRING TOO LONG (valeur d'une variable alphanumérique trop grande).
  Une chaîne de caractères ne peut contenir que 255 éléments au maximum.
- SUBSCRIPT ERROR (erreur d'affiliation).
  Nom de variable dimensionnée, supérieur à la dimension.

Exemple : DIM A (6,5)
PRINT A (7,7) provoquera l'affichage du message d'erreur.

- SYNTAX ERROR (erreur de syntaxe).
  Erreur détectée sur une commande BASIC ou sur une lecture de donnée via l'instruction READ ou INPUT.

Exemple :

```
10 READ A%
*20 DATA BON
*RUN
SYNTAX ERROR IN LINE 20

* EDIY
SYNTAX ERROR
*
```

- TYPE MISMATCH. Erreur sur une opération entre variables.

Exemple :

A$ = «BONJOUR» + B% provoquera l'affichage du message d'erreur.

- UNDEFINED ARRAY (variable dimensionnée indéfinie).
  Erreur détectée pour une variable dimensionnée non définie.

Exemple :

10 H (4,7) = 86.

S'il n'y avait pas avant cette ligne l'instruction DIM H (X, Y) avec X ⩾ 4 et Y ⩾ 7, la ligne 10 provoquerait l'affichage du message d'erreur.

- UNDEFINED LINE NUMBER (numéro de ligne inexistant).
  Les instructions GOTO X, GOSUB X et IF... THEN X provoqueront l'affichage de ce message d'erreur si la ligne X n'existe pas dans le programme.

2.5. Les différents types de valeurs

retour

2.5.1. Les variables et les nombres

Le BASIC du D.A.I. reconnaît deux types de valeurs numériques, les nombres entiers (Integer) et les nombres réels (floating point). Les entiers sont définis de $-2\uparrow31$ à $+(2\uparrow31)-1$ (soit de − 2147483647 à + 2147483646) et sont exacts. Ils ne sont jamais arrondis.

Les nombres réels sont définis de $\pm 10^{-18}$ à $\pm 10^{18}$ avec une précision de 6 chiffres significatifs. Les nombres, qu'ils soient entiers ou réels, sont codés en mémoire sur 4 octets (32 bits). La virgule est représentée par un point (2,2→2.2). Certaines commandes BASIC acceptent soit les entiers, soit les réels comme paramètres.

Par exemple :

A DRAW A,B C,D X tous les nombres A,B,C,D,X sont considérés comme des entiers. Si un nombre contient des décimales, seule la partie entière sera considérée. DRAW 0,11.8 18.7,9 5 est identique à
DRAW 0,11 18,9 5.

B A = SQR (B) le paramètre B doit être un nombre réel positif

Le BASIC du D.A.I. obéit aux règles suivantes :

1) lorsqu'un nombre réel est trouvé à la place d'un nombre entier, le système ne tient pas compte des chiffres après la virgule.
ex. : 2.3 → 2 − 1,7 → − 1

2) lorsqu'un nombre entier est trouvé à la place d'un nombre réel, l'entier est automatiquement converti.
ex. : 4 → 4.0 − 16 → − 16.0

3) lorsqu'un nombre entier (ex. : 3, et non pas 3.0) est tapé, il est codé en mémoire, soit en tant que réel, soit en tant qu'entier, suivant le contexte défini par la commande «IMP»

Le nom d'une variable BASIC peut avoir de 1 à 14 caractères mais le premier doit être une lettre, les suivants peuvent être numériques ou alphabétiques. Les caractères après le 14e sont ignorés. Si aucun symbole (**$**, %,!) n'est trouvé après le nom de la variable, le type de celle-ci (entière ou réelle) dépendra du contexte défini par la commande «IMP». A la mise sous tension, toutes les variables sont définies en tant que réelles.

Remarque : les noms des variables entières sont terminées par %
les noms des variables réelles sont terminés par !

Les noms des chaines de caractères sont terminés par **$** mais regardons sur les exemples suivants, l'influence de la commande «IMP».

exemple 1 : A la mise sous tension

- les variables I,A,S sont réelles car elles sont les abréviations de I !, A !, S !
- les variables I%, A%, S% sont entières et sont différentes de I,A,S
- les variables I!, A!, S! sont réelles et sont identiques à I,A,S
- les variables **I$**, **A$**, **S$** sont des chaines de caractères et sont différentes de I,A,S et de I%, A%, S%

Donc, si la commande «IMP» n'est pas utilisée, les variables représentant des nombres réels n'auront pas besoin d'un caractère supplémentaire pour être définies. Par contre, on devra utiliser le «%» pour les variables entières et le «**$**» pour les chaines de caractères.

IMPORTANT : A la mise sous tension, après un «Reset» ou une commande «IMP FPT» tous les noms de variables sans caractère supplémentaire sont considérés comme des abréviations de noms suivis de «!» donc comme des variables réelles.

Après la commande «IMP INT», tous les noms de variables sans caractère supplémentaire sont considérés comme des abréviations de noms suivis de «%» donc comme des variables entières.

exemple 2

```
*IMP FPT
*LIST
10  A = 45.8:S = 78.7:REM variables réelles
20  F% = 36:X% = 27:REM variables entières
30  D$ = «BONJOUR»:REM chaine de caractères.
```

maintenant si nous tapons la commande

```
*
*IMP INT : LIST
10  A! = 45.8:S! = 78.7:REM variables réelles
20  F = 36:X = 27:REM variables entières
30  D$ = «BONJOUR»:REM chaine de caractères
*
```

Après la commande IMP STR A-Z (A-Z indiquant que l'on change toutes les variables de A à Z) tous les noms de variables sans caractère supplémentaire sont considérés comme des abréviations de noms suivis de «$», donc comme des chaînes de caractères.

```
*
*IMP STR A-Z :LIST
10  A! = 45.8:S! = 78.7:REM variables réelles
20  F% = 36:X% = 27:REM variables entières
30  D = «BONJOUR»:REM chaine de caractères
*
```

Après INT ou FPT, on peut aussi préciser la limite des variables concernées par ces commandes.

ex : IMP INT A-C affecte aux variables A,B,C des valeurs entières.

IMP est surtout utilisé lors de la programmation :
si vous travaillez principalement avec des chaînes de caractères, la commande «IMP STR A-Z» vous évitera de taper à chaque fois le «$» derrière les noms de chaînes de caractères.

Même remarque pour les nombres entiers et les nombres réels.

ATTENTION : IMP ne peut pas être écrit dans un programme. IMP est une commande, pas une instruction.

2.5.2. Les chaînes de caractères

1) une chaîne de caractère peut contenir de 0 à 255 caractères.
2) les chaînes de caractères, comme les variables numériques, peuvent être dimensionnées. Donc DIM A$ (10, 10) crée 121 éléments de 255 caractères chacun. DIM A$ (10, 10) correspond à un matriçage de la mémoire en 11 lignes de 11 cases (0 à 10).
3) Le total des caractères utilisés dans une chaîne de caractères et les mots de contrôle associés ne doivent pas dépasser en mémoire la fin de la zone réservée, sinon un message d'erreur sera affiché (solution : CLEAR X, X étant plus grand que précédemment)
4) les chaînes de caractères ne peuvent pas contenir de guillemets. Mais toutefois, vous pouvez les afficher à l'aide de l'instruction CHR$ (#22) (22 hexa est le code ASCII des»)
5) Pendant une instruction «INPUT», la virgule sépare les différentes chaînes de caractères, à moins que la virgule ne soit dans une chaîne entre guillemets.

exemple :

```
*
10 INPUT A$, B$, C$
*RUN
?BONJOUR, «IL FAIT, SOMME TOUTE,», ASSEZ BEAU

A$ contient BONJOUR
B$ contient IL FAIT, SOMME TOUTE,
C$ contient ASSEZ BEAU
```

Exemples d'utilisation de chaîne de caractères.

– DIM A$ (10, 10) alloue une place mémoire pour chaque élément de la matrice.

Attention, n'oubliez pas de réserver la mémoire avant : instruction «CLEAR».

– A$ = «FRED» + V$. Le signe = assigne à la variable A$ une chaîne de caractères composée de la chaîne «FRED» suivie de la valeur de la chaîne V$. Ceci demande une place mémoire égale à la longueur de la chaîne de caractères plus un octet (chaque caractère est codé sur 8 bits : code ASCII). Si V$ contenait «ERIC», A$ contient la chaîne «FREDERIC».

ATTENTiON : pour les chaînes de caractères, l'opérateur « + » désigne une réunion de chaînes de caractères et non l'opération arithmétique

```
A$ = «12» + «95»
PRINT A$
```

le résultat est : 1295 et non pas 107.

6) IF A$ = B$ THEN STOP : Comparaison entre deux chaînes de caractères.
Pour que deux chaînes soient égales, il faut et il suffit :
1) qu'elles aient le même nombre de caractères
2) qu'elles aient les mêmes caractères dans le même ordre
ex : «ABC» est différent de «ACB»
«ABC» est différent de «A» (l'espace est un caractère)
«ABC» est égal à «ABC»

Si une des deux chaînes a moins de caractères que l'autre, elle est considérée comme plus petite, inférieure.
La longueur (ou le nombre de caractères) de deux chaînes peut être comparée par l'intermédiaire de la fonction LEN.

Si A$ = «17B!Zb» LEN (A$) = 6
A$ contient 6 caractères

Vous pouvez donc écrire
– IF LEN (A$) < LEN (B$) THEN STOP
– IF «FRED» > «FREDERIC» THEN STOP

> signifie : est supérieur à
< signifie : est inférieur à
IF signifie : si
THEN signifie : alors

7) INPUT X$, Y$: Entrée de deux chaînes de caractères via le clavier.
Dans ce cas, il n'est pas obligatoire d'écrire les chaînes de caractères entre guillemets mais les espaces commençant les chaînes seront ignorés. Si plusieurs chaînes doivent être écrites sur la même ligne, une virgule doit séparer les différentes chaînes de caractères.

Exemple :

```
INPUT X$, Y$
?␣FRED, ␣OUI ↵      X$ = «FRED» Y$ = «OUI»
?«␣FRED», «␣OUI, ␣↵      X$ = «␣FRED» Y$ = «␣OUI».
```

8) READ X$: lecture d'une chaine de caractères sous forme de DATA.
Transfert, au fur et à mesure de l'exécution des instructions READ, dans la variable spécifiée des chaînes de caractères se trouvant derrière une instruction DATA. Les chaînes ne doivent pas obligatoirement être entre guillemets, mais dans ce cas, les espaces se trouvant devant chaque chaîne seront ignorés.
Une virgule doit séparer chaque chaîne.
INPUT X$ charge la variable X$ à partir de données du clavier
READ X$ charge la variable X$ à partir de données du programme

9) PRINT X$
affiche le contenu de la variable X$

PRINT «FRED» + X$
affiche la chaîne «FRED» suivie de la chaîne contenue dans la variable X$.

2.6. Les opérateurs BASIC :

retour

Les opérateurs agissent entre deux opérandes et renvoient un résultat. Les opérandes pourront être soit arithmétiques soit alphanumériques (chaîne de caractères). Le résultat sera arithmétique, alphanumérique ou logique. Les résultats logiques ne peuvent pas être assignés à des variables, ils sont seulement utilisés pour conditionner des branchements (IF... THEN).

Les opérateurs arithmétiques ou logiques requièrent des «données type» comme opérandes. Si une donnée arithmétique n'est pas valable, elle est automatiquement transformée pour devenir correctement utilisable. Par exemple, lorsque le système additionne un nombre réel à un nombre entier, ce dernier est traduit en nombre réel pour donner un résultat réel cohérent. Si ce résultat est assigné à une variable entière, le résultat sera tronqué de sa partie fractionnaire :
ex : A% = 32,9999 A% contient le nombre 32.

Les opérateurs comme >, =, < donnent un résultat booléen (vrai ou faux). Ces résultats peuvent être manipulés par les fonctions AND, OR pour conditionner un branchement.

2.6.1. Les opérateurs arithmétiques :

Opérateurs	*type des deux opérandes*		*type du résultat*
+, −, *, /	entier	entier	entier
	réel	entier	réel
	entier	réel	réel
	réel	réel	réel
↑ (élever à la puissance) ∧ sur le clavier	Idem		toujours réel
MOD (modulo)	Idem		toujours entier

Il est à noter que si, pour une division, le dénominateur et le numérateur sont des entiers, le résultat sera un entier et il n'y aura aucun moyen de récupérer le reste de la division.
exemple : A% = 11
B% = 4
PRINT %/B% donne 2

2.6.1.1. MOD

exemple : N% = X MOD Y

«MOD» provoque une division sans virgule de X par Y et donne la valeur entière du reste. Les opérandes réelles sont automatiquement tronquées de leur partie fractionnaire.

MOD est un opérateur très utilisé lorsque l'on travaille avec des nombres dont la base est différente de 10.

exemple de programme : Conversion de minutes en heures, minutes.

```
*LIST
10  INPUT TOTALMINUTES%
20  HEURES% = TOTALMINUTES%/60
30  MINUTES% = TOTALMINUTES%MOD 60
40  PRINT :PRINT HEURES%,MINUTES%
*
```

2.6.2. Les opérateurs logiques

L'ensemble des opérateurs logiques demande deux opérandes entières et donne toujours un résultat entier. Les opérandes à virgule sont tronquées de leur partie fractionnaire avant d'être traitées. Les nombres entiers sont codés en mémoire sur 32 bits et les opérateurs permettent de réaliser certaines fonctions booléennes. Les opérandes doivent être sous la forme de variables lorsque la valeur est négative, sinon «SYNTAX ERROR».

PROBLEME : 1 - qu'est-ce qu'un bit ?
2 - Comment code-t-on un nombre ENTIER sur 32 bits en base 2 ?
3 - Comment traduire un nombre BINAIRE (base 2) en un nombre entier en base 10 ?

1) un bit est une case mémoire qui a autant de chance d'être à 0comme d'être à 1. Nous pouvons donc utiliser des bits pour écrire des nombres en base 2 (binaire).

base 2 → 2 chiffres 0 ou 1
base 10 → 10 chiffres 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Nous ne pouvons pas écrire directement un nombre en base 10 en mémoire car un bit ne peut avoir comme valeur 2 ou 3 ou 4 etc. c'est pourquoi chaque nombre décimal (base 10) est traduit en un nombre binaire pouvant être écrit en mémoire.

2) Comment un nombre entier en (base 10) s'écrit-il en base 2 + ?

Si nous divisons un nombre décimal, par exemple 2876, par 10 (sa base) nous obtenons comme reste de la division le chiffre des unités (Rang 0). Si nous recommençons avec le résultat, le reste sera égal aux dizaines (rang 1), puis aux centaines (rang 2) etc. jusqu'à ce que le résultat soit inférieur à 10.

exemple :

2876	10		
876	287	10	
76	87	28	10
⑥	⑦	⑧	②

Nous retrouvons, en lisant le dernier résultat puis les restes, le nombre 2876. Nous venons de convertir le nombre décimal 2876 dans la base du diviseur (ici 10).

• De même, pour convertir un nombre décimal en base X, nous diviserons ce nombre par X jusqu'à ce que le résultat soit inférieur à la base. Puis nous lirons le nombre converti en commençant par le dernier résultat, suivi des différents restes.

exemple : comment convertir le nombre décimal 77 en base 2 ?

77	2						
17	38	2					
①	18	19	2				
	⓪	①	9	2			
			①	4	2		
				⓪	2	2	
					⓪	①	

77 décimal s'écrit 100 1101 en binaire. 77 est converti en base 2 sur 7 chiffres. Si ce nombre est rangé en mémoire, on dira qu'il est codé sur 7 bits. Un nombre plus grand serait codé sur plus de bits, si 77 doit être codé sur 32 bits, les 25 suivants seront positionnés à zéro.

00000000000000000000000001001101 représente 77 codé en binaire sur 32 bits.

3) Comment traduire un nombre binaire en un nombre entier en base 10. Dans la base 10, le nombre 2876 est le résultat de l'addition :

$2876 = 2000 + 800 + 70 + 6$
ou encore $2876 = (2 \times 1000) + (8 + 100) + (7 \times 10) + (6 \times 1)$
ou encore $2876 = (2 \times 10^3) + (8 + 10^2) + (7 \times 10^1) + (6 \times 10^0)$

Nous nous apercevons qu'un nombre décimal est une suite d'additions dont chaque membre correspond à un chiffre du nombre décimal, multiplié par 10, élevé à la puissance du rang du chiffre (les unités étant le rang 0, les dizaines le rang 1, etc.).

D'une manière générale :
Un nombre décimal est une suite d'additions dont chaque membre correspond à un chiffre du nombre en base X à convertir, multiplié par la base élevée à la puissance du rang du chiffre.

Exemple : Nous voulons convertir le nombre Binaire 1001101 en base 10.

Le nombre binaire 1001101 correspond au chiffre décimal A tel que :

$A = (1 \times 2^6) + (0 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$
$A = 2^6 + 2^3 + 2^2 + 2^0$
$A = 64 + 8 + 4 + 1$
$A = 77$

donc le nombre binaire 1001101 est égal au nombre décimal 77

REMARQUE : en réalité le DAI Code, les nombres entiers décimaux sur 31 bits, le 32e indiquant si le nombre est positif ou négatif
si 32e bit = 0 : nombre positif
si 32e bit = 1 : nombre négatif

▶ Mais ATTENTION, dans ce cas, les nombres sont codés en complément à deux ; pour retrouver la codification précédente il faut :
positionner tous les bits à 1, à 0 et tous les bits à 0, à 1 (y compris le 32e) puis ajouter 1. Le code ainsi ajusté correspond à la valeur absolue du nombre précédemment codé.

Exemple :
111111111111...111111 (32 bits à 1) représente un nombre négatif
inversion : 0000000000...00000 : on obtient 32 bits à 0
incrémentation : 0000000...0000001 : ce nombre binaire correspond à la valeur absolue du nombre représenté par les 32 bits à 1 donc 1111...111 représente le nombre décimal − 1.

Remarque : (− 1) + 1 = 0 en base 10, de la même façon
1111111...1111 + 000000000...0001 = 0000...00000 en base 2

2.6.2.1. IAND

exemple : N% = I IAND J

Cet opérateur permet de réaliser la fonction logique «et» sur les 32 bits représentant chaque nombre.
Table de vérité de la fonction «ET»

bit n de I	«ET»	bit n de J	donne	bit n du résultat
0		0		0
0		1		0
1		0		0
1		1		1

le bit n du résultat est positionné à 1 si le bit n de I ET le bit n de J sont à 1.

Remarque : 0 «ET» X = 0
1 «ET» X = X X a pour valeur soit 0 soit 1
Le résultat est un nombre entier codé sur 32 bits.

Cet opérateur est surtout utilisé pour positionner à 0 (masquer) certains bits dans une donnée.

63 IAND 16 = 16
63 = 111111
16 = 010000
résultat = 010000 soit 16

4 IAND 2 = 0
4 = 100
2 = 010
résultat = 000 soit 0

quelque soit le contenu de la variable A%

A% IAND 0 = 0

(B% = -1) A% IAND B% = A%

#7F3E IAND #00FF = #3E (ou #003E).

2.6.2.2. IOR

exemple : N% = I IOR J.

Cet opérateur permet de réaliser la fonction logique «ou» sur les 32 bits représentant chaque nombre.

Table de vérité de la fonction «OU»

bit n de I	«OU»	bit n de J	donne	bit n du résultat
0		0		0
0		1		1
1		0		1
1		1		1

Le bit n du résultat est positionné à 1 si le bit n de I OU le bit n de J est à 1 :

Remarque : 0 «ou» X = X
1 «ou» X = 1
X a pour valeur soit 0, soit 1

Le résultat est un nombre entier codé sur 32 bits.

Cet opérateur est surtout utilisé pour mettre à 1 certains bits dans une donnée.

exemples : 4 IOR 2 = 6
4 = 100
2 = 00
résultat = 110 soit 6

10 IOR 10 = 10
10 = 1010
10 = 1010
résultat = 1010 soit 10

quelque soit le contenu de A%

A% IOR 0 = A%

(B% = -1) A% IOR B% = -1

#3E3E IOR #00FF = #3EFF

2.6.2.3. IXOR

exemple : N% = I IXOR J

Cet opérateur permet de réaliser la fonction logique «OU exclusif» sur les 32 bits représentant chaque nombre.

Table de vérité de la fonction «IXOR»

bit n de I	IXOR	bit n de J	donne	bit n du résultat
0		0		0
0		1		1
1		0		1
1		1		0

Le bit n du résultat est positionné à 1 si le bit n de J est différent du bit n de I

Remarque : 0 IXOR X = X
1 IXOR X = X
X IXOR X = 0

X a pour valeur soit 0, soit 1
|| X a pour valeur 0 quand X = 1
|| X a pour valeur 1 quand X = 0

Le résultat est un nombre entier codé sur 32 bits.

Cet opérateur est surtout utilisé pour comparer deux nombres entiers.
Si A% IXOR B% = 0 alors A% = B%.

exemples : 7 IXOR 5 = 2

7 = 111
5 = 101
résultat = 010 soit 2

8 IXOR 0 = 8

8 = 1000
0 = 0000
résultat = 1000 soit 8

quelque soit le contenu de A%
0 IXOR A% = A%
A% IXOR A% = 0
#F0 IXOR #FF = #0F.

2.6.2.4. INOT

exemple : N% = INOT (J%)

Cet opérateur permet de réaliser la fonction logique «NON» (appelé aussi «barre» ou «complémentaire») sur les 32 bits représentant le nombre entre parenthèses.

Table de vérité de la fonction «NON»

bit n de J	bit n du résultat
0	1
1	0

le bit n du résultat est positionné à 1 si le bit n de J est à 0. Si le bit n de J est à 1, le bit n du résultat est positionné à 0

Le résultat est un nombre entier codé sur 32 bits.

exemples : A% = 674 : INOT (A%) donne − 675
A% = 2645 : INOT (A%) donne − 2646
A% = − 27 : INOT (A%) donne 26
A% = − 535 : INOT (A%) donne 534

Remarque : INOT d'un nombre quelconque X donne − (X + 1)

2.6.2.5. SHR

exemple : N% = I SHR J

Cet opérateur décale, J fois, vers la droite les 32 bits représentant le nombre I. A chaque décalage, le bit le moins significatif (de rang 0) est perdu et le bit le plus significatif (rang 31) est remplacé par un zéro.

exemples : 77 SHR 1 = 38

77 = 1001101

38 SHR 1 = 19

1er décalage = 0100110 soit 38

77 SHR 2 = 19

2e décalage = 0010011 soit 19

quelque soit A%
A% SHR 0 = A%
A% SHR 32 = 0

Remarque : Si J est négatif, un message d'erreur en résulte : SYNTAX ERROR

2.6.2.6. SHL

exemple : N% = I SHL J

Cet opérateur décale, J fois, vers la gauche les 32 bits représentant le nombre I. A chaque décalage, le plus significatif (rang 31) est perdu et le bit le moins significatif (rang 0) est remplacé par 0.

exemples : 19 SHL 1 = 38
38 SHL 1 = 76
19 SHL 2 = 76

19 = 0010011
1er décalage = 0100110 soit 38
2e décalage = 1001100 soit 76

quelque soit A% :
A% SHL 0 = A%
A% SHL 32 = 0

▶ Remarque : Si J est négatif, un message d'erreur apparaît : SYNTAX ERROR

2.6.2.7. L'utilisation des opérateurs logiques

L'utilisation typique de ce genre d'opérateur est le test de bit sur des mots provenant d'un Port en entrée.

Sur le DAI les ports sont considérés comme des cases mémoires de 8 bits, le bit 7 étant le plus significatif et le bit 0 étant le moins significatif.

case mémoire

Par exemple, supposons que le bit 1 du port n° 5 soit à 0 lorsque la porte d'une pièce est fermée, et à 1 lorsqu'elle est ouverte. Le programme qui suit affichera un message dans le cas où la porte s'ouvre.

10 IF INP (5) IAND 2 = 0 THEN 10.
Le système ne passera à la ligne suivante que lorsque le bit 1 du port n° 5 sera égal à 1 :
INP (5) = XXXXXX0X (porte fermée. X = soit 0, soit 1)
2 = 00000010
Le résultat de la fonction «ET» = 00000000 = 0 le système réexécute donc la ligne 10.
par contre, si le bit 1 passe à 1 :
INP (5) = XXXXXX1X
2 = 00000010
Le résultat de la fonction «ET» = 00000010 = 2 ce résultat n'est pas égal à zéro, le système passe donc à la ligne suivante.
20 PRINT «ALERTE INTRUSION !!»

On peut aussi remplacer la ligne 10 par l'instruction «WAIT» :

10 WAIT 5,2

Cette instruction attend que le bit 1 du port n° 5 passe à 1.

Dans notre programme, «WAIT» est plus simple que l'instruction «IF» et prend moins de place mémoire pour être stockée.

L'exemple suivant déclenche une alarme si, sur le port n° 5, le bit 3 = 1 ou si le bit 1 = 0 ou si le bit 0 = 0.

Il faut, tout d'abord, masquer les bits qui ne nous intéressent pas à l'aide de l'opérateur IAND. Rappelons que, quelque soit X :
X «ET» 0 donne 0
X «ET» 1 donne X

Le masque sera, en binaire : 00001011 donc 11 en décimal.

L'alerte n'est pas déclenchée si bit 3 = 0 et bit 1 = 1 et bit 0 = 1.

Dans ce cas, après l'instruction INP (5) IAND 11, nous devons avoir :

INP (5) XXXX0X11
«ET» 11 00001011
résultat : 00000011 soit 3 en décimal.

Si le résultat, après IAND, est différent de 3, il faudra alors déclencher l'alarme. Le programme peut s'écrire comme suit :

```
*
LIST
100  IF INP(5) IAND 11 = 3 THEN 100
110  PRINT «ETAT D'ALERTE»
*
```

2.6.3. Les opérateurs de comparaison

L'ensemble de ces opérateurs demande deux paramètres de même type (soit 2 opérandes numériques, soit 2 opérandes alphanumériques) et délivre un résultat Booléen (vrai ou faux). Ces opérateurs ne sont utilisés qu'à l'intérieur d'une instruction IF. Une variable ne peut pas être assignée au résultat d'une comparaison.

Opérateurs de comparaison :

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

On peut comparer :

un nombre réel à un nombre réel
un nombre entier à un nombre réel
un nombre réel à un nombre entier
une chaîne de caractères à une chaîne de caractères.

▶Remarque : - lorsqu'un entier est comparé à un nombre réel, l'entier est converti en réel, 3 devient 3,0

- vous ne pouvez pas comparer une chaîne de caractères à un nombre quelqu'il soit.

exemple :
```
IF A$ = «OUI» THEN 90
IF A% >B THEN 100
```

Nous pouvons associer différentes conditions grâce aux fonctions Booléennes AND (ET) et OR (OU). L'utilisation des parenthèses est alors obligatoire.

Rappel : fonction AND – , il faut que les deux conditions soient vraies pour que le résultat soit vrai
fonction OR – il faut que l'une des deux conditions soit vraie pour que le résultat soit vrai.

exemple :
```
IF (A >B OR A = C) AND C>D THEN 1000
```
pour que le programme passe à la ligne 1000 il faut que A soit supérieur à B OU A égal à C ET, de plus, il faut que C soit supérieur à D.

2.6.4. *Exemples d'opérations*

(Les nombres sans virgule représentent des nombres entiers)

Opérations	Résultats	types de résultat	
1 + 2	3	entier	
1.0 + 2.0	3.0	réel	
1.0 + 2	3.0	réel	
3 *4	12	entier	
3 ↑4	81.0	réel	ATTENTION
12.0/4.0	3.0	réel	
12.0/4	3.0	réel	
12/4	3	entier	
11/4	2	entier	ATTENTION
3 IAND 2	2	entier	
3.0 IAND 6.0	2	entier	
3.14 IAND 6.72	2	entier	
3 SHL 2	12	entier	
3.2 SHL 2.8	12	entier	
7 = 4	Faux	logique	
3.0 > 2.1	Vrai	logique	
«FREA» < «FREDA»	Vrai	logique	
«A» = «A»	Vrai	logique	
7.1 = 7	Faux	logique	
7.0 = 7	Vrai	logique	ATTENTION
3 < 4 OR 7 = 8	Vrai	logique	
4 < 3 OR 7 = 8	Faux	logique	
3 = 7 AND 9 <10	Faux	logique	
3 < 4 AND 10 > 9	Vrai	logique	

2.6.5. *Les priorités d'exécution des différents opérateurs*

Si une expression contient seulement des variables entières et des opérateurs travaillant sur les entiers, à aucun moment les nombres réels ne sont utilisés.

Ordre de priorités :

expression entre parenthèses
↑ puissance
*, /, MOD
+, −
SHL, SHR
IOR, IAND, IXOR, INOT
>, <, =, <>, <=, >=
AND, OR.

Lorsqu'une expression contient plusieurs opérateurs de même priorité, les calculs se font de gauche à droite.

exemples :

1) 3*5 MOD 2
   15 MOD 2
   1

2) 3 + 5 MOD 2 — MOD est prioritaire sur +
   3 + 1
   4

3) 3↑3 + 10/5 = 29 — ATTENTION

4) 3↑3 + 10/2 + 3 = 35

2.7. Les instructions numériques et alphanumériques

Voici la liste alphabétique des fonctions numériques et alphanumériques. Ces instructions demandent un argument (entre parenthèses) et renvoient un résultat pouvant être assigné à une variable ou utilisé dans une expression numérique.

retour

2.7.1. ABS (X)

Renvoie la partie absolue de l'argument X :

si X positif ABS (X) = X
si X négatif ABS (X) = − X

exemples : ABS (− 253.7) = 253.7
ABS (186,5) = 186.5
ABS (0) = 0.0
ABS (0.001) = 1E-3
le résultat est un nombre réel.

2.7.2. ACOS (X)

Renvoie l'arcosinus en radians de l'argument X. ACOS (X) demande quel est l'angle qui a pour cosinus l'argument X.

La valeur de X doit être comprise entre − 1 et + 1.

Le résultat est un réel positif variant de 0 à PI (π)

exemples : ACOS (0) = 1.5708 (PI/2)
ACOS (1.0) = 0.0
ACOS (- 1.0) = 3.14159 (PI)

2.7.3. ALOG (X)

Renvoie le nombre qui a pour logarythme décimal l'argument X.

exemples : ALOG (1) = 10.0
ALOG (2) = 100.0
ALOG (0.3020) = 2.0047
ALOG (- 1) = 0.1

2.7.4. ASC (X$)

Renvoie le code ASCII du premier caractère de la chaîne X$; le résultat est un entier.

exemples : ASC («ABC») = 65 car le code ASCII de A est 65 (# 41)
ASC («») = 0 si le nombre de caractère est nul.
ASC («␣») = 32. Espace a pour code ASCII 32.

Remarque : le code ASCII des guillemets (") est 34.

2.7.5. ASIN (X)

Renvoie l'arcsinus en radians de l'argument X.ASIN (X) demande quel est l'angle qui a pour sinus l'argument X.

La valeur de X doit être comprise entre − 1 et + 1

le résultat est un réel variant de $-\pi/2$ à $+\pi/2$

exemples : ASIN (0) = 0.0
ASIN (1) = 1,5708 ($\pi/2$)
ASIN (- 1) = − 1,5708 ($-\pi/2$)

2.7.6. ATN (X)

Renvoie l'arctangente en radians de l'argument X.ATN (X), demande quel est l'angle qui a pour tangente l'argument X.

Le résultat est un réel variant de $-\pi/2$ à $+\pi/2$

Exemples :
ATN (0) = 0.0
ATN (− 1.0) = − 0.785398 (- PI/4)
ATN (10000) = 1.5707.

2.7.7. CHR$ (I)

Renvoie le caractère qui a pour code ASCII l'argument I (fonction inverse de ASC).

La valeur de I doit être comprise entre 0 et 255.

Exemples :
CHR$ (65) renvoie comme résultant la chaîne «A»
CHR$ (42) renvoie comme résultat la chaîne «*»
CHR$ (#42) renvoie comme résultat la chaine «B».

Remarque :
le # signale que le chiffre qui suit est hexadécimal (base 16).

2.7.8. COS (X)

Renvoie le cosinus de l'argument X.
La valeur de X représente un angle en radians.

Note : 1 radian = $180/\pi$degrés = 57,2958 degrés.

Cosinus de X en degrés = COS (X/57,2958).

exemples :
COS (3.14159) = − 1.0
COS (180/57.2958) = − 1.0
COS (90/57.2958) = 0.0.

2.7.9. EXP (X)

Renvoie le nombre réel «e» (2.71828) élevé à la puissance de l'argument X (e^X).

Suivant les versions, l'argument X doit être compris :
- exactement, entre − 31 et + 31 inclus, avec le processeur arithmétique
- approximativement entre − 42 et + 42, sans le processeur arithmétique.

exemples :
EXP (− 1) = 0.367879
EXP (1) = 2.71828.

2.7.10. FRAC (X)

Renvoie la partie de l'argument X.

exemples :
FRAC (2.7) = 0.7
FRAC (− 1.2) = − 0.2.

2.7.11. HEX$ (I)

Renvoie une chaîne de caractères représentant la valeur hexadécimale de l'argument I.

Exemples :
HEX$ (12) = «C»
HEX$ (255) = «FF».

2.7.12. INT (X)

Renvoie la valeur entière de l'argument X. Le résultat est un nombre *réel toujours inférieur ou égal* à X.

Si X < 0 alors − (1 + INT(-X)) est renvoyé.

exemples : INT (0.23) = 0.0
INT (7.0) = 7.0
INT (2.9) = 2.0
INT (− 2.1) = − 3.0

2.7.13. LEFT$ (X$, I)

Renvoie une chaîne de caractères comprenant les I premiers caractères de X$. (I⩾0).

Exemples : LEFT $ («BONJOUR», 3) renvoie «BON»
LEFT $ («BONJOUR», 1) renvoie «B»
LEFT $ («BONJOUR», 0) renvoie une chaîne nulle
LEFT $ («BONJOUR», − 1) provoque une erreur
LEFT $ («BONJOUR», 8) provoque une erreur.

2.7.14. LEN (X$)

Renvoie un nombre entier positif représentant le nombre de caractères de l'argument X$.

Exemples : LEN («BONJOUR») = 7
LEN («») = 0.

2.7.15. LOG (X)

Renvoie le logarythme naturel (base e) de l'argument X. (X > 0).

Exemples : LOG (1) = 0.0
LOG (2) = 0.693147.

2.7.16. LOGT (X)

Renvoie le logarythme décimal (base 10) de l'argument X. (X > 0).

Exemples : LOGT (100) = 2.0
LOGT (10) = 1.0.

2.7.17. MID$ (X$, I, J)

Renvoie une chaîne de J caractères commençant à la position I de la chaîne X$. Le premier caractère a la position 0.

exemple : MID$ («BONJOUR», 1, 3) crée la chaîne «ONJ»
MID$ («BONJOUR», 1, 0) crée une chaîne nulle.

Remarque : I et J doivent être positifs et I + J doit être inférieur ou égal à LEN (X$), sinon une erreur est détectée.

2.7.18. PI

Renvoie le nombre réel dont la valeur est 3.14159.

Exemple : COS (PI) = − 1.0.

2.7.19. RIGHT$ (X$, I)

Renvoie une chaîne de caractères formée des I derniers caractères de X$.

ATTENTION : 0⩽ <I ⩽ LEN (X$) sinon ERREUR

exemples : RIGHT$ («BONJOUR», 4) crée «JOUR»
RIGHT$ («BONJOUR», 0) crée une chaîne nulle
RIGHT$ («BONJOUR», − 1) provoque une erreur.

2.7.20 RND (X)

si X < 0 RND (X) déclenche une séquence de génération d'un nombre aléatoire par PROGRAMME. Le même nombre négatif X provoque la MÊME séquence, donc la même suite de résultats. Les résultats sont compris entre 0 et X exclus.

si X > 0 RND (X) déclenche une NOUVELLE séquence de génération d'un nombre aléatoire par PROGRAMME. Les résultats sont compris entre 0 et X exclus.

si X = 0 RND (0) renvoie un nombre aléatoire, provenant d'un CIRCUIT générateur de nombres, compris entre 0 et 1 mais ce résultat peut être ramené à une valeur utile à l'aide d'une multiplication.

Le programme qui suit est un jeu de dés. Le générateur du nombre aléatoire est écrit à la ligne 30.

Ce nombre est compris entre 1 et 6 inclus et représente les 6 faces d'un dé à jouer.

```
*LIST
10  INPUT «VOTRE PARI:»;A%PRINT
20  IF A%>6 OR A%<1 THEN 10
30  B% = RND(6) + 1
40  IF B% = A% THEN PRINT «Vous avez gagné»:GOTO 10
50  PRINT «Vous avez Perdu, c'était :»;B%:GOTO 10
*
```

2.7.21 SGN (X)

Renvoie : 1.0 si x > 0 positif
0.0 si x = 0 nul
− 1.0 si x < 0 négatif

2.7.22 SIN (X)

Renvoie le sinus de l'argument X. La valeur de X représente un angle en radians.

NOTE : 1 Radian = 180/π degrés = 57.2958 degrés.
sinus de X en degrés = SIN (X/57.2958)

exemples : SIN (90/57.2958) = 1.0
SIN (3.14159) = 0.0

2.7.23. SPC (I)

Renvoie une chaîne de caractères où le nombre d'espaces est égal à I.

ATTENTION I ≤ 255

2.7.24. SQR (X)

Renvoie la racine carrée de l'argument X.
Le résultat est un nombre réel.

L'argument doit être supérieur ou égal à zéro.

Exemples : SQR (9) = 3.0
SQR (0) = 0.0

2.7.25. STR$ (X)

Renvoie une chaîne dont les caractères représentent le nombre X.

exemples : STR $ (9.2) crée la chaîne «␣9.2» (␣ signifie espace)
STR $ (− 20.88) crée la chaîne « − 20.88»
LEN (STR $ (9.2) = 4 (3 digits + 1 espace)
ATTENTION : STR $ (I%) avec I% = 7 crée la chaîne «␣ 7.0»

2.7.26. TAB (I)

Renvoie une chaîne d'espaces nécessaires pour déplacer le curseur vers la droite jusqu'à la colonne I. Le curseur ne peut être déplacé que vers la droite avec cette instruction. Tenter de le déplacer vers la gauche ne provoque aucun effet.

2.7.27. TAN (X)

Renvoie la tangente de l'argument X. La valeur de X représente un angle en radians.

NOTE : 1 Radian = 180/π degrés = 57.2958 degrés.
Tangente de x en degrés = TAN (X/57.2958)

2.7.28. VAL (X$)

Renvoie le nombre réel représenté par la chaine de caractères X$ (fonction inverse de STR $).

Exemples : VAL («9.2») = 9.2
VAL («7») = 7.0 ATTENTION
VAL («A») provoque une ERREUR
VAL (STR $ (9.2)) provoque une erreur (à cause du «space» provoqué par STR $).
VAL (RIGHT $ (STR $ (9.2),3)) = 9.2

Remarque : X $ doit représenter un nombre.

2.8. Les instructions de contrôle du programme

2.8.1. END

retour

Termine l'exécution d'un programme BASIC. Seule l'instruction RUN peut réexécuter ce programme. Le message «END PROGRAM» est affiché à l'exécution de l'instruction END.

2.8.2. FOR... NEXT

Syntaxe : FOR <variable> = <expression>TO<expression> STEP <expression> (éventuellement).

exemples :

I) FOR X = 1.0 TO 9.3 STEP 0.6
II) FOR P = 0.0 TO 9.3.
III) FOR Z = 10 ↑ N TO 3.4/Q STEP SQR (X)
IV) FOR I% = 9 TO 0 STEP − 1
V) FOR I = 1 TO 9 : FOR J = 1 TO 9 : PRINT I,J : NEXT : NEXT

La variable de l'instruction FOR est initialisée à la première expression donnée. Les instructions suivantes sont exécutées jusqu'à l'instruction NEXT. A l'exécution du NEXT, la variable est incrémentée de la valeur de l'expression du STEP puis le programme «saute» à l'instruction qui suit l'expression FOR. Les instructions FOR... NEXT provoquent donc une boucle de programme jusqu'à ce que la variable ait atteint la valeur de la deuxième expression. Si le STEP n'est pas précisé, l'incrément sera, par défaut, égal à 1. A la fin de chaque boucle la valeur de la variable est incrémentée de la valeur du STEP.

Il est à noter que toutes les expressions de l'instruction FOR sont définies avant que la boucle ne commence. Nous ne pouvons pas, par exemple, changer le STEP en cours de bouche.

exemple :

```
10  B = 1.0
20  FOR I = 1.0 TO 10.0 STEP B
30  PRINT I
40  B = B + 1.0
50  NEXT:REM NEXT Provoque un saut à la ligne 30
*RUN
 1.0
 2.0
 3.0
 4.0
 5.0
 6.0
 7.0
 8.0
 9.0
10.0
*
```

Le fait d'incrémenter la variable B n'a rien changé à la valeur du STEP dans la boucle.

Le nombre de fois N que la boucle est effectuée est donné par la formule :

N = INT (((Val d'arrivée − Val départ)/val STEP) + 1)

avec FOR <variable> = <val départ> TO <val d'arrivée> STEP <val STEP>. Nous ne remarquons que si «val STEP» égale zéro, la boucle est impossible, et le message d'erreur «DIVISION BY ZERO» apparaîtra.

Cas particuliers :

A) l'interpréteur terminera une boucle non achevée si une instruction NEXT d'une autre boucle est rencontrée.

exemple :

```
FOR A = 1 TO 10:FOR B = 1 TO 10 : PRINT A,B : NEXTA
```

est admis

B) Le fait d'exécuter une instruction FOR avant que sa *propre boucle* ne soit finie, réinitialise la boucle au début.

exemple :

```
*LIST
10  FOR A = 1.0 TO 10.0
20  FOR B = 0.0 TO 3.0
30  PRINT A,B:GOTO 10
*
```

est admis.

C) Une boucle est arrêtée à l'exécution d'une instruction RETURN.

exemple :

```
*LIST
10  GOSUB 30
20  STOP
30  FOR A = 1.0 TO 10.0
40  PRINT A
50  RETURN
*
```

est admis.

D) Après la fin d'une boucle, la variable a la valeur d'arrivée de la boucle plus 1.

exemple :

```
*LIST
10  FOR I% = 0 TO 10:NEXT
20  PRINT I%
*RUN
 11
*
```

2.8.3. GOSUB

exemple : GOSUB 9000

provoque un branchement à la ligne 9000. Lorsque l'instruction RETURN sera exécutée, la prochaine instruction exécutée sera celle qui se trouve après le GOSUB. Les GOSUB imbriqués sont limités à la mémoire du «Stack» pointeur. Un programme peut avoir 10 niveaux de GOSUB ou 15 niveaux de FOR sans poser de problèmes. On dit que GOSUB provoque l'appel d'un sous-programme.

2.8.4. GOTO

exemple : GOTO 100

provoque un saut de programme à la ligne spécifiée (ici 100)

2.8.5. IF... GOTO

Le programme «saute» à la ligne spécifiée si la condition est VRAIE.

Syntaxe : IF <condition> GOTO <n° de ligne>
exemple : IF A = B + 25 GOTO 70 : Z = A.

Si l'égalité est vraie, le programme saute à la ligne 70 sinon l'instruction Z = A est exécutée.

2.8.6. IF... THEN

Cette instruction a deux syntaxes possibles.

I) IF <CONDITION> THEN <n° de ligne>

dans cette forme, elle est exactement identique à l'instruction IF... GOTO.

II) IF <condition> THEN <instruction>

La première partie est identique aux autres exemples : c'est la condition.

Après le THEN, une ou plusieurs instructions peuvent être données sur la MEME LIGNE (avec : pour les séparer). Leur exécution dépendra de la condition. Si la condition est vraie, elles seront exécutées. Si la condition est fausse, elles seront ignorées et le programme *passera à la ligne suivante*

exemple :

```
10  FOR I% = 1 TO 10
20  X = 0.0
30  IF I% = 5.0 THEN X = 1.0:X = X + 1.0:X = SQR(X)
40  PRINT I%,X
50  NEXT
*RUN
 1          0.0
 2          0.0
 3          0.0
 4          0.0
 5          1.41421
 6          0.0
 7          0.0
 8          0.0
 9          0.0
10          0.0
*
```

Les instructions de la ligne 30 (après le THEN) seront exécutées lorsque I sera égal à 5. Si I est différent de 5, elles sont ignorées et le programme passe directement à la ligne 40.

ATTENTION : Si la forme IF... THEN GOTO est utilisée, les instructions derrière le GOTO et sur la même ligne ne seront jamais exécutées.

exemple :

```
*LIST
100  IF A = B THEN GOTO 110:A = A + B
110  PRINT A,B
*
```

si A = B le programme se branche à la ligne 110

si A ≠ B l'instruction A = A + B est ignorée puisqu'elle se trouve sur la même ligne et derrière le GOTO.

Donc, que la condition soit vraie ou fausse, la deuxième partie de la ligne 100 est toujours ignorée.

Ce programme peut être corrigé de la façon suivante.

```
*LIST
100  IF A = B THEN GOTO 110
105  A = A + B
110  PRINT A,B
*
```

2.8.7. ON... GOTO

exemple : 1) ON I GOTO 10, 20, 30, 40

le programme se branche à la ligne indiquée par le I^ème^ nombre après GOTO.
Donc, si

I = 1, saut en 10
I = 2, saut en 20
I = 3, saut en 30
I = 4, saut en 40

Le BASIC ne tient compte que de la partie entière de I.
Si 0 ≤ I < 1 ou si I> au nombre de saut possible, alors, l'instruction suivant le ON... GOTO est exécutée.

Si I négatif ou si une ligne n'existe pas un message d'erreur apparaît.

exemple : 2) ON SGN (X) + 2 GOTO 100, 200, 300

le programme se branche :
en 100 si X négatif
en 200 si X nul
en 300 si x positif

2.8.8. ON... GOSUB

exemple : ON I GOSUB 100, 200
«ON... GOTO», excepté qu'ici on exécute un GOSUB à la place d'un GOTO.
Identique à :
L'instruction RETURN, après le GOSUB provoquera un retour à l'instruction qui suit le «ON... GOSUB» sur la même ligne ou sur la ligne suivante.

2.8.9. RETURN

Provoque le retour à l'instruction suivant le DERNIER GOSUB exécuté.

2.8.10 STOP

exemple : 200 STOP

Le BASIC suspend l'exécution du programme et rend le contrôle à l'utilisateur en affichant «STOPPED IN LINE 200».

Pour continuer le programme à l'instruction qui suit STOP, taper «CONT».

2.8.11. WAIT

Cette instruction possède trois fonctions différentes suivant la syntaxe employée.

I) WAIT <expression>, <expression>, <expression>
(facultatif.)

exemples : WAIT I, J, K
WAIT A(N), P.

Considérons l'instruction WAIT I, J, K.

Cette instruction lit le mot de 8 bits (octet) se trouvant sur le port I, fait un IXOR entre l'octet lu et K, puis un IAND entre le résultat et J jusqu'à obtenir un résultat égal à J. A ce moment, l'instruction suivant WAIT est exécutée. Si l'instruction WAIT ne comprend que 2 arguments, le 3e est considéré comme NUL.

I, J, et K doivent être compris entre 0 et 255 inclus.

Considérons l'exemple suivant :
Vous voulez attendre jusqu'à ce que, sur le port 1, les bits 5 et 7 soient à 1 et les bits 3 et 1 soient à zéro.

• Pour cela, mettre tous les bits de J qui ne sont pas à considérer à zéro (ici les bits 6,4,2,0) et mettre à 1 tous les bits concernés par le test (ici les bits 7,5,3,1). C'est ainsi que l'on détermine la valeur de J. J est appelé un «MASQUE». Il masque les bits qui ne nous intéressent pas.

• Tous les bits initialisés à zéro dans J, soit initialisés à zéro dans K. Tous les bits, considérés par le test comme devant être à 1, sont positionnés à 0 dans K (ici bits 7 et 5). Tous les bits, considérés par le test comme devant être à 0 sont positionnés à 1 dans K (ici bits 3 et 1).

Ceci nous donne :

bit n°	7	6	5	4	3	2	1	0
État du port 1 désiré	1	X	1	X	0	X	0	X
J	1	0	1	0	1	0	1	0
K	0	0	0	0	1	0	1	0

X = aucune importance : soit égal à 0 soit à 1

dans ce cas précis, l'instruction s'écrit :

WAIT 1, 170, 10 (en décimal)
ou WAIT 1, #AA, #A (en hexadécimal)

exemple :

Etat du port 1 à l'instant t	00110001
IXOR	
K	00001010
1er résultat	00111011

Rappel IXOR

0	0	=	0
1	0	=	1
0	1	=	1
1	1	=	0

puis un IAND avec J

1er résultat	00111011
IAND	
J	10101010
résultat final	00101010

Rappel : IAND

0	0	=	0
0	1	=	0
1	0	=	0
1	1	=	1

Le programme continue d'attendre dans la mesure où le résultat est différent de J.

deux remarques :

1) les bits 6,4,2,0 n'ont aucune importance dans cet exemple car :
X IAND 0 = 0

2) dans l'exemple où J = 10101010 et où le résultat final = 00101010 ; nous remarquons que c'est le bit n° 7 qui n'est pas dans l'état voulu.

II) WAIT MEM <expression>, <expression>, <expression>
(facultatif.)

exemples : WAIT MEM I, J, K,
WAIT MEM PORT%, A + B

Cette forme de syntaxe est similaire à la précédente, mais le premier paramètre correspond ici, à une case mémoire. Rappelons que la structure du DAI est en mémoire banalisée. Chaque entrée/sortie est adressable comme une case mémoire.

Exemple d'utilisation de cette instruction : attente d'un contact sur le bouton du paddle 1.

```
*LIST
10  WAIT MEM #FD00, #20
20  PRINT «BOUTON APPUYE»
*
```

20 égal en binaire à 00100000. L'action ur le bouton du paddle 1 provoque la mise à 1 du bit 5 de la case mémoire # FD00.

III)
WAIT TIME I

Cette instruction suspend l'exécution du programme pendant un temps égal à I *20 millisecondes.
I peut varier de 0 à 65535
Le temps varie entre 0 et 21 mm 50 s.
exemple :

```
10 WAIT TIME 50 : REM ATTENTE 1 s.
```

2.9. Les instructions d'accès aux cases mémoires.

retour

2.9.1. CALLM

Exemple :

I) CALLM 1234

provoque l'exécution du programme en langage machine se trouvant en mémoire à l'adresse décimale 1234. Le retour à l'instruction BASIC suivant le CALLM s'effectuera par l'intermédiaire du code machine # C9 (ret.).

II) CALLM I, V

provoque l'exécution du programme en langage machine se trouvant en mémoire à l'adresse spécifiée par I. Mais le double registre HL du 8080A est chargé avec l'adresse mémoire de la variable V.

ATTENTION : le programme machine doit préserver TOUS LES registres utilisés, même HL, et les restituer avant le RET (# C9).

– Si V est une variable numérique, HL contient l'adresse où est stockée la valeur de V. HL contient HEX$ (VARPTR(V)).

Remarque : une valeur entière est stockée sur quatre octets, le plus à gauche étant le plus significatif.

exemples : si X% = 256 codification 00.00.10.00
si X% = 257 codification 00.00.10.01.

– Si V est une chaîne de caractères, HL pointe la case mémoire où se trouve l'adresse basse (l'adresse haute étant en HL + 1) où est stockée le PREMIER OCTET DEFINISSANT V.

exemple : si X$ = «BA»
codification : 02. 42. 41

nombre de caractères dans X$ (en hexa)

codes ASCII hexa de la chaîne X$

Remarque : le premier octet définissant une chaine est le NOMBRE DE CARACTERES.

2.9.2. INP (I)

exemple A = INP (# 31)

Cette instruction lit l'octet se trouvant sur la carte DCE-BUS n° 3 port 1 et assigne cette valeur à la variable A.

2.9.3. OUT I, J

exemple : OUT # 91, A

sort la valeur de la variable A sur la carte DCE-BUS n° 9 port 1.
I et J doivent être compris entre 0 et 255.

2.9.4. PDL (I)

Exemple : A = PDL (I)

positionne la valeur de la variable A à un nombre compris entre 0 et 255 représentant la position physique d'un des potentiomètres d'un paddle.

I doit être compris entre 0 et 5 inclus.

I = 0 correspond au potentiomètre X paddle n° 1
I = 1 correspond au potentiomètre Y paddle n° 1
I = 2 correspond au potentiomètre Z paddle n° 1
I = 3 correspond au potentiomètre X paddle n° 2
I = 4 correspond au potentiomètre Z paddle n° 2
I = 5 correspond au potentiomètre Y paddle n° 2

Si aucun paddle n'est branché, A contient une valeur entre 0 et 255

2.9.5. PEEK (I)

exemples : I) A = PEEK (# 13C2)
le contenu décimal de l'adresse mémoire #13C2 (en hexa) est assigné à la variable A. Si I est supérieur à 65536 ou négatif, une erreur est détectée.

II) PRINT PEEK (258)
affiche en décimal la valeur de la case mémoire dont l'adresse décimale est 258.

III) PRINT HEX$ (PEEK(258))
affiche en hexadécimal la valeur de la case mémoire dont l'adresse décimale est 258.

2.9.6. POKE

exemple : POKE I, J

Cette instruction range l'octet spécifié par J dans la case mémoire dont l'adresse est spécifiée par I.

J doit être compris entre 0 et 255, sinon : erreur
I doit être compris entre 0 et 65535 sinon : Erreur

l'utilisation insouciante de l'instruction POKE peut provoquer l'arrêt complet de la machine et la perte du programme qui y est stocké ; le «Reset» restant la seule solution pour récupérer le contrôle. Un POKE dans une case mémoire dont on ne connaît pas l'usage est très dangereux. Toutefois, un mauvais POKE ne peut, en aucun cas, détériorer la machine.

Exemples de POKE :

POKE # 131,0 sortie sur écran et RS 232
POKE # 131,1 sortie sur écran uniquement
POKE # 131,2 sortie vers buffer d'édition
POKE #135,2 lecture du buffer d'édition
POKE # 13D, #10 sélection cassette n° 1
POKE #13D, #20 sélection cassette n° 2
POKE # 40, #28 mise en marche moteur cassette 1
POKE # 40, # 18, mise en marche moteur cassette 2
POKE # 40, #30, arrêt des moteurs cassette 1 et 2
POKE # 730, #30 disquette drive 0 activé
POKE # 730, # 31 disquette drive 1 activé.

2.9.7. UT

exemple : UT
appelle les utilitaires du moniteur

2.10. Instruction sur des données ou des entrées/sorties

retour

2.10.1. DATA

exemples : I) DATA 1,3, − 1,E3, − 0.4, #F3
cette instruction spécifie des données qui seront lues par le programme dans l'ordre où ils apparaissent dans l'instruction DATA (de la gauche vers la droite).

II) DATA «F00», «ZO0», «B,A»
des chaînes de caractères peuvent être lues par l'intermédiaire de l'instruction DATA. Si une chaîne contient un espace ou une virgule, cette donnée doit être écrite entre guillemets.

Remarque : Sans les guillemets, les espaces entre la dernière virgule et la première lettre de la chaîne sont ignorés.

Exemples :
```
5 FOR I% = 0TO 2
10 READ A$ : PRINT «a» + A$ + «a» : NEXT : STOP
20 DATA ␣OUT␣, ␣␣NON␣␣, ␣␣␣X␣Y␣␣␣
RUN
aOUI␣a
aNON␣␣a
aX␣Y␣␣␣a
Si les données étaient entre guillemets
RUN
a␣OUI␣a
a␣␣NON␣␣a
a␣␣␣X␣Y␣␣␣a
```
Note : ␣ signifie : espace

2.10.2. GETC

exemple : A = GETC

le code ASCII de la touche appuyée est assigné à la variable A. Si aucune touche n'est actionnée : GETC = 0.

ATTENTION : GETC n'attend pas l'enfoncement d'une touche

exemple :
```
*LIST
1010  PRINT «FIN.....Taper F»
1020  PRINT «POUR REJOUER.....Taper C»
1030  PRINT «POUR UN AUTRE JEU. Taper A»
1040  A% = GETC:IF A% = 0 THEN 1040:REM boucle d'attente
1050  IF A% = 70 THEN PRINT «C'est fini»:END
1060  IF A% = 67.0 THEN PRINT «On rejoue»:GOTO 10
1070  IF A% = 65.0 THEN PRINT «Autre jeu»:GOTO 3000
1080  GOTO 1040
*
```

ligne 10 : début du jeu
ligne 3000 : début d'un autre jeu
70 : code ASCII de F
67 : code ASCII de C
65 : code ASCII de A

2.10.3. INPUT

exemples : I) INPUT V, W, W2

demande de données au clavier : ici trois. La première sera assignée à la variable V, la deuxième à W, la troisième à W2. Chaque donnée doit être séparée de la suivante par une virgule. La dernière valeur tapée doit se terminer par un RETURN. Un point d'interrogation apparaît sur l'écran lorsqu'une instruction INPUT est exécutée.

Seules des constantes peuvent être tapées comme réponse à un INPUT ; telles que 4.5, 2E-3 ou «CHAT».

– Si plus de données étaient demandées dans l'instruction INPUT, un autre point d'interrogation apparaitrait et le reste des données pourrait être tapé.

– Si plus de valeurs demandées étaient tapées au clavier, les valeurs supplémentaires seraient ignorées et le système afficherait le message «SOME INPUT IGNORED». Toutefois l'exécution, dans ce cas, n'est pas arrêtée.

II) INPUT «VALEUR» ; V

affiche VALEUR : avant de demander la valeur de la variable V.

«CONT», après qu'une instruction INPUT a été interrompue par la touche BREAK, relance complètement l'instruction INPUT.

NOTE : l'instruction INPUT ne provoque pas de saut de ligne lors de l'exécution,

exemple :
```
10 INPUT A%
20 PRINT A%
```
La valeur entrée et la valeur affichée seront inscrites sur la même ligne d'écran.

2.10.4 PRINT (peut-être remplacé par «?»)

exemples :
```
I) PRINT X, Y, Z$
II) PRINT
III) PRINT X ; Y
IV) PRINT «la valeur est» ; A
V) ? A, B
VI) ? HEX $ (A + VARPTR (C) * 8)
```

Cette instruction affiche une expression numérique ou alphanumérique sur l'écran (ou autre périphérique suivant le cas). Si la liste des valeurs à afficher n'est pas terminée par une virgule ou un point-virgule, un saut de ligne (C.R.) est exécuté après l'affichage des valeurs.

exemples : 1

```
*LIST
10  PRINT «A», Z%
20  PRINT «B», Z%
*RUN
A          0
B          0
*
```

2

```
*LIST
10  PRINT «A», Z%,
20  PRINT «B», Z%
*RUN
A          0          B          0
```

Si un point-virgule sépare deux expressions dans un PRINT, leurs valeurs seront affichées :

– l'une à côté de l'autre si la deuxième expression est alphanumérique

– avec un espace entre les deux valeurs si la deuxième expression est numérique positive.

Exemple : 10 A% = 5 : B% = 7 : C% – 3 (l'espace correspond au signe +)
```
20 PRINT A% ; B% ; «OUI» ; C%
RUN
␣5␣7OUI – 3
```

Si une virgule apparaît après une expression dans un PRINT, le curseur se positionne à la prochaine colonne d'édition. Il existe 5 colonnes qui sont : position 0, 12, 24, 36 et 48. S'il n'y a pas d'expression derrière un PRINT (ex.II), cette expression provoque juste un saut de ligne (C.R.)

Remarque : l'affichage d'une expression numérique positive est TOUJOURS précédé d'un ESPACE.

2.10.5. READ

exemple : READ V

Cette instruction lit les données spécifiées dans une instruction DATA et assigne la donnée pointée à la variable V. A la suite d'un RUN le pointeur de DATA se place sur la première donnée du premier DATA. A chaque fois qu'une instruction READ est exécutée, le pointeur passe à la donnée suivante. L'exécution d'une instruction READ après que toutes les données ont été lues, provoque un message d'erreur «OUT OF DATA IN LINE X» où X précise la ligne de l'instruction READ.

La commande RUN positionne le pointeur de DATA sur la première donnée du premier DATA. L'instruction RESTORE positionne, EN COURS d'EXECUTION d'un programme, le pointeur de DATA au début.

exemple :

```
*LIST
10  READ A%
20  PRINT A%
30  GOTO 10
40  DATA 23,4,5,3
50  DATA 6,103, 55
*RUN
 23
 4
 5
 3
 6
103
 55
OUT OF DATA IN LINE 10
*
```

2.10.6. RESTORE

Repositionne le pointeur de DATA à la première donnée du premier DATA.

exemple :

```
*LIST
10  READ A%
20  IF A% = 55.0 THEN RESTORE
30  PRINT A%;
40  GOTO 10
50  DATA 23,4,5,3
60  DATA 6,103,55

*RUN
 23 4 5 3 6 103 55 23 4 5 3 6 103 55 23 4 5 3 6 103 55 23 4 5
3 6 103 55 23 4 5 3 6 103 5 etc.
***BREAK
```

2.11. Instructions d'entrée/sortie sur cassette ou disquette

2.11.1. CHECK

retour

La commande CHECK examine tous les fichiers se trouvant sur une cassette. Le type et le nom du fichier sont affichés, suivis du mot «OK» ou «BAD»

- «OK» signifie que le fichier est bon
- «BAD» signifie que le fichier est mal enregistré, un LOAD «ce fichier» provoquerait une erreur.

Pour les minicassettes, les commandes ne peuvent pas s'arrêter d'elles-mêmes. Pour récupérer le contrôle du DAI, il suffit d'appuyer un instant sur la touche BREAK .

2.11.2. LOAD

exemple : I) LOAD «FRED»
charge dans le micro-ordinateur, à partir d'une cassette ou d'une disquette, le programme nommé FRED. Lorsque le chargement est terminé, le système rend le contrôle à l'utilisateur en affichant l'étoile-BASIC.

II) LOAD
charge le premier programme rencontré sur une cassette.

Si la commande du moteur du minicassette est sous contrôle automatique, il sera démarré. Sinon le magnétophone doit être déclenché manuellement.

Lorsque la commande LOAD «FRED» est exécutée, le type (0 pour les programmes BASIC) et le nom de chaque fichier rencontré sont affichés. Lorsque le chargement est réussi, l'étoile est visualisée sur l'écran. Si le chargement est mauvais, le message «LOADING ERROR» est inscrit, suivi d'un code indiquant la nature de l'erreur.

0 : erreur de «checksum» dans le nom
1 : mémoire insuffisante
2 : erreur de «checksum» dans le programme
3 : coupure dans l'enregistrement.

Le clignotement du curseur cesse lorsque les données de la cassette commencent à être lues.

Lors de l'exécution de l'instruction LOAD (ou LOAD «nom) dans un programme, ce système démarre le moteur du magnétophone, efface le programme qui était en mémoire, charge le premier fichier rencontré (ou le fichier «nom») puis l'exécute à partir de la première ligne.

2.11.3. LOAD A

Charge à partir d'une cassette (ou d'une disquette) une donnée (ou un tableau de données) assignée à la variable qui suit LOADA.

exemple : LOADA X «FRED»
LOADA V$ «TEST»

exemple de programme :

```
*LIST
10     CLEAR 2000
20     DIM A$(5.0,5.0)
30     FOR T = 0.0 TO 5.0:FOR T1 = 0.0 TO 5.0:A$(T,T1) = «OUI»:NEXT:NEXT
40     PRINT «METTRE LA CASSETTE SUR ENREGISTREMENT, PUIS TAPER SPACE»
50     IF GETC<>32.0 THEN 50
60     SAVEA A$ «TITRE»
70     REM effacement de toutes les variables A$
80     FOR T = 0.0 TO 5.0:FOR T1 = 0.0 TO 5.0:A$(T,T1) = « »:NEXT:NEXT
90     PRINT «REVENIR EN ARRIERE, METTRE EN LECTURE, TAPER SPACE»
```

```
00 IF GETC<>32.0 THEN 100
10 LOADA A$ «TITRE»
20 FOR T = 0.0 TO 5.0:FOR T1 = 0.0 TO 5.0:PRINT A$(T,T1):NEXT:NEXT
*
```

Remarque : Lors de l'exécution de l'instruction LOADA A$ «TITRE», le nom et le type des autres programmes ou fichiers rencontrés sur la cassette ne sont pas affichés. Les fichiers lus par LOADA sont du type 2.

2.11.4. SAVE

exemple :

I) SAVE «FRED»

II) SAVE A$

Cette commande préserve sur cassette ou sur disquette le programme présent dans la mémoire du DAI. Le programme n'est pas modifié. Plusieurs programmes peuvent être stockés sur une même cassette (ou disquette) grâce à cette commande. Le programme est écrit sur la cassette après le nom spécifié (ex I) FRED. II) A$).

Un nom de fichier peut comporter jusqu'à 60 caractères.

III) SAVE

Le programme sera écrit sur la cassette après un nom «nul».

Le système accepte ces différentes commandes en affichant le message : «SET RECORD, START TAPE, TYPE SPACE». Placez la bonne face de la cassette dans le magnétophone, positionnez le minicassette en séquence d'enregistrement (noter que si le contrôle du moteur est relié au micro-ordinateur, le moteur ne tournera pas) puis appuyer sur SPACE . Le moteur démarre et le programme est transféré de la mémoire centrale sur la bande de la cassette. A la fin de l'enregistrement le moteur s'arrête (s'il est relié au DAI) et l'étoile apparaît sur l'écran. Si le magnétophone fonctionne manuellement, il peut être arrêté lorsqu'apparaît l'étoile.

Les fichiers créés par SAVE sont du type 0.

2.11.5. SAVEA

exemples :

I) SAVEA G% «GEORGES»

II) SAVEA A$ B$

Sauve sur cassette (ou disquette) les données qui sont assignées à la variable qui suit SAVEA. La chaîne de caractères qui termine cette instruction est le nom de ce fichier. Lors de l'exécution de cette instruction, le magnétophone doit être en mode enregistrement et la cassette doit être POSITIONNEE.

exemple de programme :

```
*LIST
10  CLEAR 2000
20  DIM A$(5.0,5.0)
30  LOADA A$ «TITRE»
40  POKE #13D, #20:REM Selection sortie cassette 2
50  SAVER A$ «COPIE TITRE»
60  POKE #13D, #10:REM Selection sortie cassette 1
70  END
*
```

Ce programme lit les données «TITRE» sur le magnétophone n° 1 et recopie ces données sur le minicassette n° 2 sous le nom «COPIE TITRE»

Les fichiers créés par SAVEA sont du type 2.

2.12. Instructions de contrôle et de mise au point.

2.12.1. CONT

retour

exemple : CONT

demande de CONTinuer l'exécution du programme BASIC à l'instruction qui suit l'instruction STOP ou qui suit la dernière instruction exécutée avant d'appuyer sur BREAK .

exemple :

```
*10  PRINT«BONJOUR»:STOP
*20  PRINT«OK»:STOP
*30  GOTO 10
*RUN
BONJOUR
STOPPED IN LINE 10
*CONT
OK
STOPPED IN LINE 20
*CONT
BONJOUR
STOPPED IN LINE 10
*
```

Messages indiquant à quelle ligne se trouve l'instruction «STOP»

Remarque : CONT n'est pas une instruction que l'on peut trouver dans un programme.

2.12.2. REM

exemples :

I) 10 REM maintenant V = 0

Cette instruction permet d'avoir des commentaires à l'intérieur d'un programme BASIC. Les instructions REM ne sont pas exécutées, mais on peut faire un GOTO à une ligne comportant un REM.

IMPORTANT : l'instruction REM se termine par la fin de la ligne et non pas par le caractère «:»

II) 10 REM mettre l'entrée à 0 : ENT = 0

l'instruction ENT = 0 ne sera jamais exécutée puisqu'elle se trouve derrière un REM.

2.12.3. STEP

Cette commande permet l'exécution d'un programme LIGNE PAR LIGNE. Si, au cours de l'exécution d'un programme, l'enfoncement de la touche BREAK est détecté ou si l'instruction STOP est exécutée, le programme s'arrête et l'étoile apparaît. L'utilisateur peut alors taper CONT (voir ci-dessus) ou STEP. Dans le dernier cas, chaque ligne de programme sera affichée AVANT d'attendre un espace pour être exécutée BREAK provoquera la fin du mode STEP.

Exemple de programme :

```
*LIST
10    PRINT «OK»
20    STOP
30    PRINT «BONJOUR»:PRINT «MESSIEURS»
40    GOTO 100
50    PRINT A
60    END
100   A = 90.0
110   GOTO 50
*RUN
OK
STOPPED IN LINE 20
*STEP
30    PRINT «BONJOUR»:PRINT «MESSIEURS» •
BONJOUR
MESSIEURS
40    GOTO 100 •
100   A = 90.0 •
110    GOTO 50 •
50    PRINT A •
 90.0
60    END
END   PROGRAM
*
```

• signifie que l'ordinateur attend un SPACE pour exécuter la ligne qu'il vient d'afficher et pour inscrire la suivante.

2.12.4. TRON

exemple :

```
LIST
10    PRINT «OK»
20    TRON
30    PRINT «BONJOUR»:PRINT «MESSIEURS»
40    GOTO 100
50    PRINT A
55    TROFF
60    END
100   A = 90.0
110   GOTO 50
*
```

L'exécution de l'instruction TRON provoque exactement le même affichage que la commande STEP ; mais le passage à l'instruction suivante se fait automatiquement (en mode STEP. Nous devons appuyer sur SPACE). Après l'exécution de TRON chaque ligne de programme est affichée avant d'être exécutée. Exemple d'exécution d'un programme (voir exemple ci-dessus).

```
*
*RUN
OK
30    PRINT «BONJOUR»:PRINT «MESSIEURS»
BONJOUR
MESSIEURS
40    GOTO 100
100   A = 90.0
110   GOTO 50
50    PRINT A
 90.0
55    TROFF
END PROGRAM
*
```

2.12.5. TROFF

annule l'instruction «TRON». Si TROFF est oubliée, l'instruction TRON sera annulée à la fin du programme

2.13. Instructions de dimensionnement

2.13.1. CLEAR

retour

exemple : CLEAR 2000

Remet à zéro toutes les variables numériques et à « » les variables alphanumériques. Le nombre d'octets à initialiser est à spécifier derrière CLEAR. Ce nombre peut varier entre 4 et 32 767 inclus.

2.13.2. DIM

exemple : I) DIM A (3), B (10)
II) DIM R3 (5,5), D$(2,2,2)

Cette instruction alloue de la place mémoire pour les différentes valeurs d'une variable. Le dimensionnement s'effectue sur une, deux ou trois dimensions. Les valeurs des dimensions commencent à zéro, donc :
DIM A (3) signifie qu'il existera 4 valeurs possibles pour A. Afin de les distinguer, elles sont appelées respectivement A (0), A (1) A (2) et A (3). La notation A (I%) avec la variable I% comprise entre 0 et 3 est autorisée, voire même fortement conseillée.

La longueur maximum pour une dimension est de 255. Les dimensions peuvent être spécifiées par une variable ou une expression.

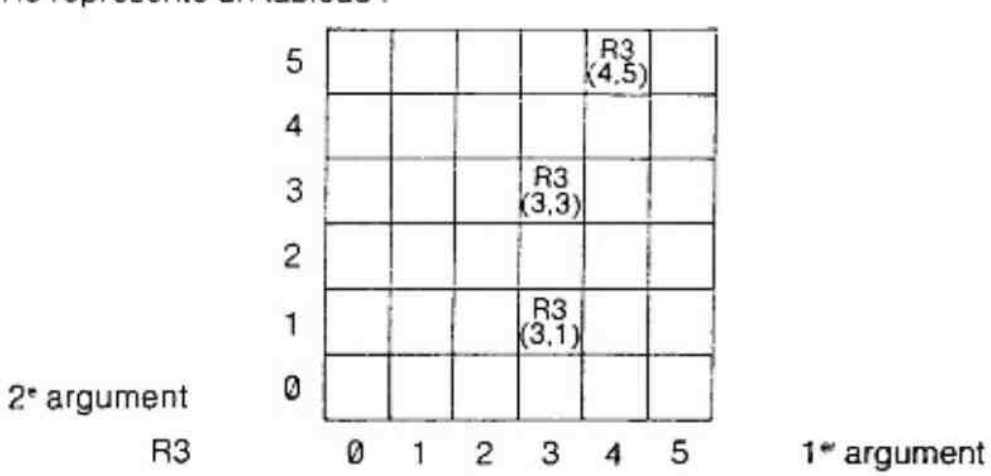
Pour des valeurs supérieures à 10, DIM est obligatoirement précédé de CLEAR.

Nous pouvons changer le dimensionnement d'une variable pendant l'exécution d'un programme mais les précédentes valeurs sont réinitialisées.

exemple :

```
*LIST
10  DIM A%(2.0)
20  A%(0.0) = 10:A%(1.0) = 25:A%(2.0) = 78
30  PRINT A%)0.0);A%(1.0);A%(2.0)
40  DIM A%(3.0)
50  PRINT A%(0.0);A%(1.0);A%(2.0);A%(3.0)
*RUN
 10 25 78
 0 0 0 0
*
```

DIM R3 (5,5) signifie qu'il existera 36 (6 × 6) valeurs possibles pour R3. On dit que R3 représente un tableau :



2.13.3. FRE

exemples :

I) A = FRE

La variable A contient le nombre d'octets laissés libres par le PROGRAMME BASIC.

La place mémoire allouée pour les dimensionnements n'est pas incluse dans ce nombre.

II) PRINT FRE

Le total de la place mémoire restante est affichée. A la mise sous tension FRE = 44898.

2.13.4. LET

exemples : I) LET W = X
II) V = 5.1
III) A = B + A

assigne une valeur à une variable. Le mot LET n'est pas obligatoire.

ATTENTION, le signe « = » n'a ici aucun sens mathématique. Il signifie un TRANSFERT de valeur de la DROITE vers la GAUCHE :

```
*LIST
10  A% = 3
20  B% = 6
30  A% = B%
40  PRINT A%,B%
*RUN
 6          6
*
```

2.13.5. VARPTR (V)

I) A = VARPTR (B)
assigne à la variable A l'adresse mémoire décimale du premier des quatre octets définissant la valeur de la variable B.

II) A = VARPTR (B(3,4))
assigne à la variable A l'adresse mémoire décimale du premier des quatre octets définissant la valeur de l'élément de la matrice B (X, Y) ici B (3,4).

2.14. Instructions graphiques

retour

2.14.1. Instructions d'initialisation

2.14.1.1.MODE

I) MODE 0 définit l'écran en mode Texte

II) MODE 1A définit l'écran en mode graphique basse résolution avec quatre lignes de texte en bas.

Le DAI possède trois définitions différentes pour l'affichage en graphique et dans chacune d'elle, il existe quatre configurations possibles de l'écran. Deux sont entièrement graphiques, les deux autres configurations sont identiques aux deux premières, mis à part la zone graphique décalée vers le haut pour permettre l'écriture de quatre lignes de texte en bas de l'écran.

La vidéo-RAM a deux configurations possibles.
Dans le 1er cas chaque point de l'écran peut être allumé suivant une des 16 couleurs. Toutefois un ensemble de 8 points horizontaux (position 0 à 7, 8 à 15 etc.) ne peut avoir que 2, parfois 3, couleurs différentes. Pour plus de renseignements au niveau de cette restriction, se rapporter au paragraphe 5.2.2.
Dans le 2e cas, chaque point de l'écran ne peut avoir qu'une des quatre couleurs définies. Ces quatres couleurs peuvent être changées à tout moment ce qui permet, au niveau de l'animation, des effets de couleurs très intéressants.

TABLE DES DIFFERENTS MODE.

MODE	dimensions graphique X × Y	dimensions texte	couleurs
0		24 × 60	2 parmi 16
1	72 × 65		16
1A	72 × 65	4 × 60	16
2	72 × 65		4 parmi 16
2A	72 × 65	4 × 60	4 parmi 16
3	160 × 130		16
3A	160 × 130	4 × 60	16
4	160 × 130		4 parmi 16
4A	160 × 130	4 × 60	4 parmi 16
5	336 × 256		16
5A	336 × 256	4 × 60	16
6	336 × 256		4 parmi 16
6A	337 × 256	4 × 60	4 parmi 16

Possibilité de 528 × 240

2.14.1.2. COLORG

exemple : COLORG 0 13 8 9

Cette instruction définit les quatre couleurs disponibles pour les modes graphiques pairs (sauf 0). Si COLORG est utilisée comme une commande BASIC, le changement de couleurs du dessin est immédiat.
Par exemple, chaque point ayant pour couleur le premier «code couleur» du précédent COLORG est maintenant de la couleur 0. Si l'écran est dans un mode impair, la commande COLORG ne change rien à l'écran. Mais si plus tard l'écran est défini dans un mode pair, (non nul), les quatre couleurs auront été spécifiées par la dernière commande COLORG enregistrée.

A la mise sous tension COLORG est initialisée à 0 5 10 15.
COLORG sert aux modes 2, 2A, 4, 4A, 6 et 6A.

Les codes couleurs sont :

0	NOIR	8	GRIS
1	BLEU VIF	9	BLEU MARINE
2	PARME	10	ORANGE
3	ROUGE VIF	11	ROSE SAUMON
4	VERT KAKI	12	BLEU CLAIR
5	VERT PRE	13	VERT CLAIR
6	ROUGE FONCE	14	JAUNE
7	VIOLET	15	BLANC

Il est à noter que ces couleurs peuvent varier suivant le réglage du téléviseur.

2.14.1.3. COLORT

exemple : COLORT 8 15 0 0

Cette instruction permet de changer de couleur en mode texte.
Le premier «code couleur» définit la couleur du fond, le deuxième définit la couleur ces caractères alphanumériques du texte.
Les deux autres codes couleur ne sont normalement pas utilisés. On peut changer ces couleurs à l'aide de POKE ou d'un programme en langage machine.

A la mise sous tension COLORT est initialisée à 8000.

2.14.2. Instructions de dessins

En graphique, les points de l'écran sont définis par deux coordonnées (X, Y), (0, 0) étant le point en bas à gauche de l'écran. Les valeurs maximums de X et de Y sont variables suivant le mode graphique défini ; si un dessin dépasse ces valeurs, le message «OFF SCREEN» sera affiché.

Il est cependant possible de dessiner dans la partie invisible de la zone graphique lorsque l'écran est défini avec quatre lignes de texte.

Grâce aux instructions de dessin, vous pouvez directement allumer un point, tracer une droite ou tracer un rectangle.

2.14.2.1. DOT

exemple : DOT 10, 20 15

allume le point de couleur 15 à la position X = 10 et Y = 20. Les dimensions du point dépendent directement de la résolution graphique sélectionnée.

2.14.2.2. DRAW

exemple : DRAW 91,73 42,77 15

Trace une ligne de couleur 15 entre les positions (91,73) et (42,77), l'ordre des deux positions n'a pas d'importance et la grosseur de la ligne dépend de la résolution graphique sélectionnée.

2.14.2.3. FILL

exemple : FILL 91,73 42,77 15

dessine un rectangle de la «couleur 15», ayant pour sommets opposés les positions (91,73) et (42,77). L'ordre des deux points n'a pas d'importance et les dimensions du rectangle dépendent de la résolution choisie.

2.14.3. Instructions de contrôle d'écran.

2.14.3.1. CURSOR

exemple : CURSOR 40,20

Affiche le curseur en position 40 (la position du premier caractère est 0) à la ligne 20 (la ligne 20 se trouve en bas de l'écran). En mode 0les positions supérieures à 59 et les numéros de ligne supérieurs à 23 provoquent un message d'erreur «OFF SCREEN».

Dans les modes mixtes (graphique + 4 lignes de texte), une position supérieure à 59 ou un numéro de ligne supérieur à 3 provoquent l'affichage du message «OFF SCREEN».

2.14.3.2. CURX

exemple : A = CURX

Met dans la variable A la valeur de la position en X du curseur (numéro du caractère). Cette valeur ne peut pas être supérieure à 59.

2.14.3.3. CURY

exemple : A = CURY

Met dans la variable A la valeur de la position en Y du curseur (n° de ligne). Cette valeur ne peut pas être supérieure à 23.

exemple de programme :

```
*LIST
10  CURSOR 20,10
20  A = CURX:B = CURY
30  PRINT A;B
*RUN
 20.0 10.0
*
```

2.14.3.4. SCRN

exemple : A = SCRN (31,20)

Met dans la variable A le code-couleur du point dont les coordonnées sont (31,20)

2.14.3.5. XMAX

exemple : A = XMAX

Met dans la variable A la valeur maximum autorisée par la coordonnée X dans le mode graphique défini.

en mode 1	XMAX = 71
en mode 3	XMAX = 159
en mode 5	XMAX = 335

2.14.3.6. YMAX

exemple : A = YMAX

Met dans la variable A la valeur maximum autorisée pour la coordonnée Y dans le mode graphique défini.

en mode 1	YMAX = 64
en mode 3	YMAX = 129
en mode 5	YMAX = 255.

exemple de programme :

```
*LIST
10  MODE 2
20  COLORG 0 13 3 8
30  FOR I = 1.0 TO XMAX
40  DOT I,30 13
50  DOT I-1,30 0:NEXT
60  FOR I = XMAX-1.0 TO 0.0 STEP -1.0
70  DOT 1,30 3
80  DOT I + 1,30 0:NEXT
90  GOTO 30
*
```

2.14.4. Les codes couleur de 16 à 23.

Lorsque l'affichage est défini dans un mode 4 couleurs (2, 2A, 4, 4A, 6, 6A) chaque point de l'écran est codé sur 2 bits. Cette valeur binaire sélectionne une des quatre couleurs définies par COLORG. Par exemple, l'instruction COLORG 0 5 10 15 définit comme 1ère couleur (0), le noir, comme 2e couleur le vert pré (5), comme 3e couleur l'orange (10) et comme 4e couleur le blanc (15).

bit d'un point de l'écran		Valeurs binaires	Codes couleurs	Couleurs autorisées
Rang 1	Rang 0			
0	0	0	0	NOIR
0	1	1	5	VERT PRE
1	0	2	10	ORANGE
1	1	3	15	BLANC

il existe 4 pseudo «codes-couleur» permettant de changer les bits définissant un point et 4 autres permettant de définir la couleur d'un point sans connaître les 4 couleurs autorisées par COLORG. Ces pseudo «codes-couleurs» peuvent être utilisés avec DOT, DRAW, FILL.

2.14.4.1. Codes couleur de 16 à 19

Lorsque l'affichage est défini dans un mode 4 couleurs, chaque point de l'écran est codé sur deux bits. Ces couleurs 16,17 18 et 19 permettent de changer un des 2 bits.

en utilisant la couleur :	Vous obtenez :
– 17	la mise à 1 du bit rang 0
– 19	la mise à 1 du bit rang 1
– 16	la mise à 0 du bit rang 0
– 18	la mise à 0 du bit rang 1

exemple de programme :

```
*LIST
10  MODE 2
20  COLORG 0 9 15 12
30  FILL 0,0 30,30 17
40  WAIT TIME 50
60  FILL 10,10 71,64 19
70  IF GETC = 0 GOTO 70
80  END
*
```

- Les lignes 10 et 20 définissent le mode graphique n° 2 (72 × 65 points) et les quatre couleurs utilisées sont : en premier le noir, en second le bleu marine, en troisième le blanc et en quatrième le bleu clair. De plus, le changement de MODE (le programme est écrit en mode 0) positionne tous les bits des points de l'écran à la valeur binaire 0
- La ligne 30 trace un carré de 31 points de côté avec la pseudo couleur 17. Toutes les valeurs binaires des points définis par le carré, auront leur bit de rang 0 mis à 1.

  Valeur binaire initiale 00 → couleur 17 → 01.

  La valeur binaire 01 code la 2e couleur. Le carré est donc en bleu foncé. Dans ce cas, l'instruction FILL 0,0 30,30 9 aurait donné le même résultat.
- la ligne 40 suspend l'exécution du programme pendant une seconde (50 × 0,02 seconde).
- La ligne 60 trace un rectangle de 72 points sur 65 points avec la pseudo-couleur 19. Toutes les valeurs binaires des points définis par ce rectangle auront leur bit de rang 1 mis à 1. Une zone recouverte par le rectangle était précédemment NOIRE ; cette zone sera maintenant BLANCHE

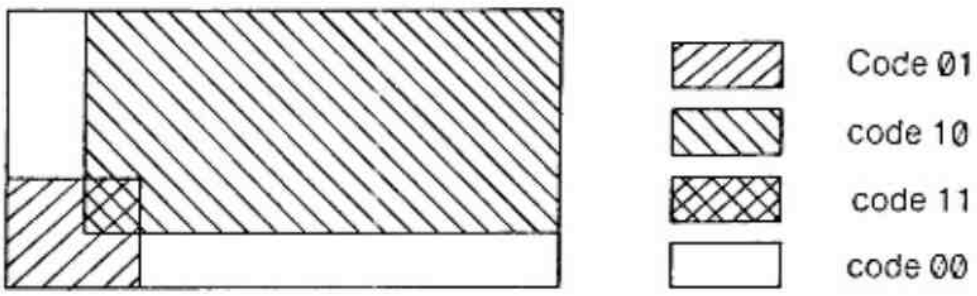
  valeur binaire initiale 00 → couleur 19 → 10.

La valeur binaire 10 code la 3e couleur, le blanc. Une zone récouverte par le rectangle était précédemment BLEU MARINE, cette zone sera maintenant BLEU CLAIR

Valeur binaire bleu marine 01 → couleur 19 →11.

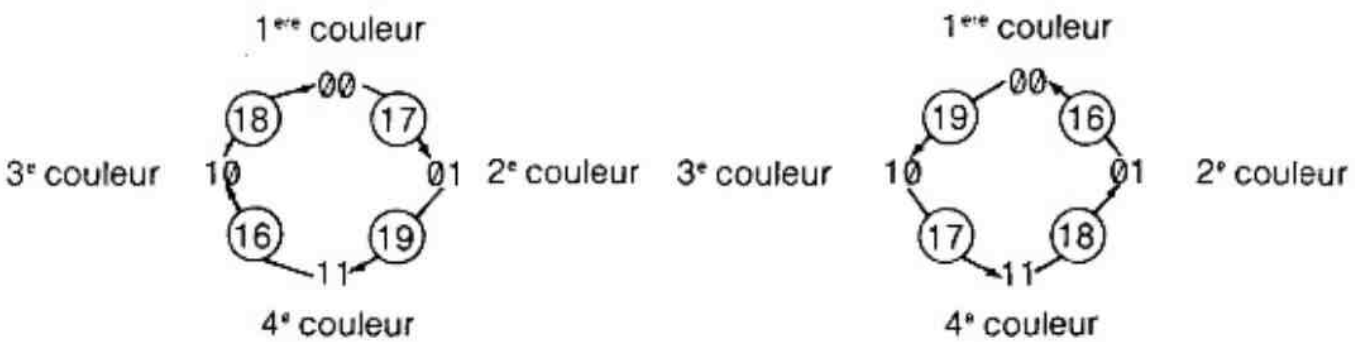
La valeur binaire 11 code la 4e couleur, le bleu «clair».

Nous avons sur l'écran



– La ligne 70 attend qu'une touche soit appuyée pour terminer le programme.

NOTE : Pour passer d'une couleur à une autre :



exemple de programme :

```
*LIST
10  MODE 2
20  Y% = YMAX/3:W% = Y% + R
30  D% = RND(7.0) + 1.0:B% = RND(7.0) + 1.0:C% = D% + B%
40  FOR A% = 0 TO XMAX-10
50  COLORG 0 D% B% C%
60  E% = XMAX-10-A%
70  FILL A%,Y% A% + 10,W% + 10 17
80  FILL E%,W%E% + 10,W% + 10 19
90  WAIT TIME 10:MODE 2:NEXT:GOTO 30
*
```

A chaque fin de boucle, l'écran est effacé par l'instruction MODE 2 ; donc à la ligne 70, la couleur 0 devient la couleur D% (00→01) et à la ligne 80, la couleur 0 devient la couleur B% (00→10).

Lorsque les deux parallélogrammes se croisent, la couleur C% apparaît. La ligne 70 colore avec D% (01) puis la ligne 80 provoque la transition de 01 à 11. 11 est la valeur binaire de la quatrième couleur, ici C%.

2.14.4.2. Les «codes couleurs» de 20 à 23.

Lorsque l'affichage est défini dans un mode 4 couleurs, les pseudo-couleurs 20, 21 22 et 23 définissent respectivement la 1ère, la 2e la 3e et la 4e couleur autorisées.

pseudo-couleur	couleur	valeur binaire d'un point écran
20	1ère couleur	00
21	2e couleur	01
22	3e couleur	10
23	4e couleur	11

exemple : COLORG 0 10 15 5

La pseudo-couleur 20 correspond au noir (0)
21 correspond à l'orange (10)
22 correspond au blanc (15)
23 correspond au vert pré (5).

FILL 0,0 XMAX, YMAX 22 : tout l'écran sera de couleur blanche (15) ;

Ces 4 couleurs permettent de définir des commandes de dessin sans avoir connaissance des couleurs autorisées par la dernière instruction COLORG.

Exemple de programme :

```
*LIST
10    MODE 2:REM efface l'écran
20    COLORG 0 0 0 0
30    FILL 0,0 XMAX, YMAX 0
40    COLORG 5 14 0 0
40    FILL 0, 30 10,40 14
60    P = 0.0:REM P indique la colonne
70    A = 0.0:REM A indique le changement de couleur
80    REM début animation
90    COLORG   5 + 4*A   14-A   14-A   5 + 4*A
100   WAIT TIME 5
110   FILL P,40 P,30 22
120   Z = (P + 11.0) MOD XMAX:FILL Z,30 Z.40 23
130   COLORG   5 + 4*A   14-A   14-A   5 + 4*A
140   FILL P,30P, 40 20:FILL Z, 30 Z,40 23
150   P = (P + 1.0) MOD XMAX
160   IF P = 0.0 THEN A = (A + 1.0) MOD 3.0
170   GOTO 80
*
```

en ligne 90 et 130 :
si A = 0 COLORG 5 14 14 5
si A = 1 COLORG 9 13 13 9
si A = 2 COLORG 13 12 12 13

2.15. Générateur de son :

retour

2.15.1. Programmation des sons

Le générateur de son du micro-ordinateur DAI composé de trois canaux oscillateurs et d'un canal de bruits blancs, est commandé par des commandes BASIC admises par le programme de contrôle du système sonore.

La commande SOUND est une des méthodes de contrôle du son. Cette commande spécifie le canal concerné, l'enveloppe utilisée, le volume et la fréquence désirée.

Exemple de commande :

SOUND 0 1 15 0 FREQ (1000)

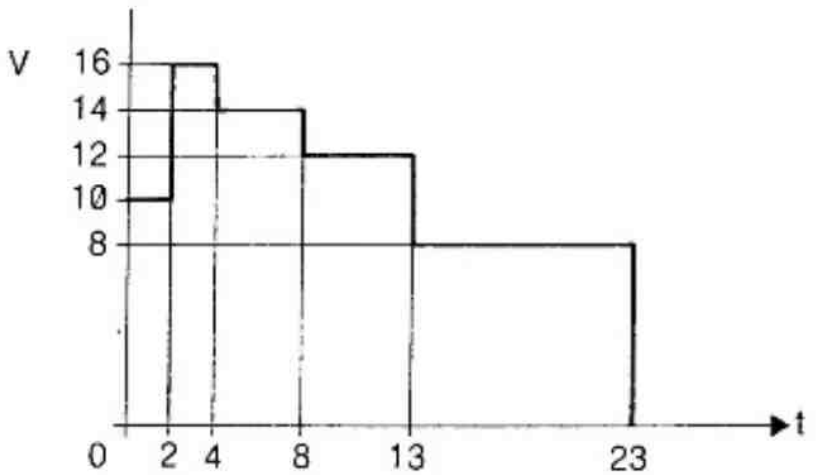
Cette commande concerne le canal 0 utilisé avec l'enveloppe n°1, un volume de 15 et une fréquence de 1000 Hz.

L'instruction ENVELOPE permet de changer rapidement le volume d'une note dans la même proportion qu'un instrument de musique. Ainsi la montée et la descente du volume pour une note peuvent être spécifiées. Cette commande définit un volume en fonction du temps. L'unité de volume est comprise entre 0 et 16 et le temps a pour unité 3,2 unité-seconde ($3{,}2.10^{-1}$ s.).

Par exemple la commande :

ENVELOPE 0 10,2 ; 16,2 ; 14,4 ; 12,5 ; 8,10 ; 0

définit le volume représenté ci-dessous :



Ainsi à chaque fois qu'une commande SOUND est donnée, cela produit un «court éclat» de son.

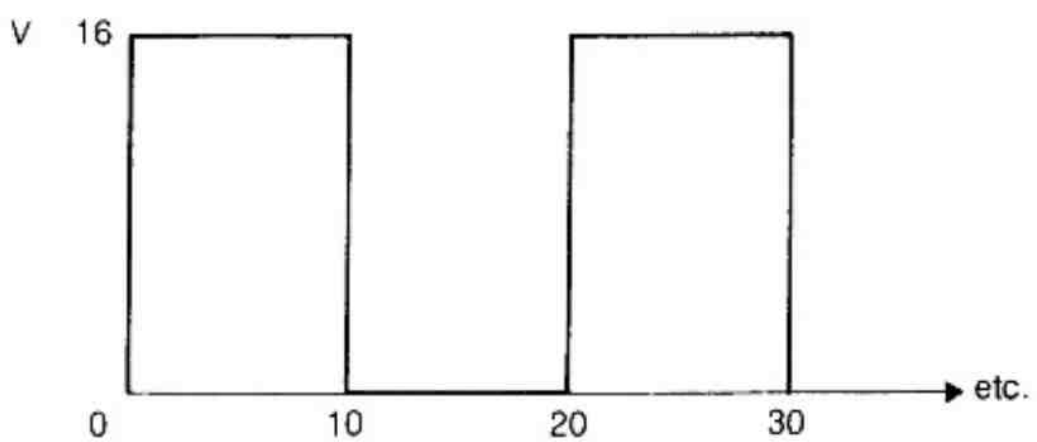
En variant l'enveloppe, vous faites varier les qualités du son produit. Le volume spécifié dans la commande SOUND est multiplié par le contenu de l'enveloppe. Ainsi, si une commanee SOUND demande un volume de 8 unités (8/15 du volume total) et que l'enveloppe demande 4 unités (4/16 du volume maximum) le volume utilisé sera de 2/15 du maximum.

$$(4/16 \times 8/15 = 8/60 = 2/15)$$

Dans certains cas, le volume du son doit rester constant ou varier périodiquement en fonction du temps. Dans ce cas, l'enveloppe est répétée indéfinitivement. Par exemple :

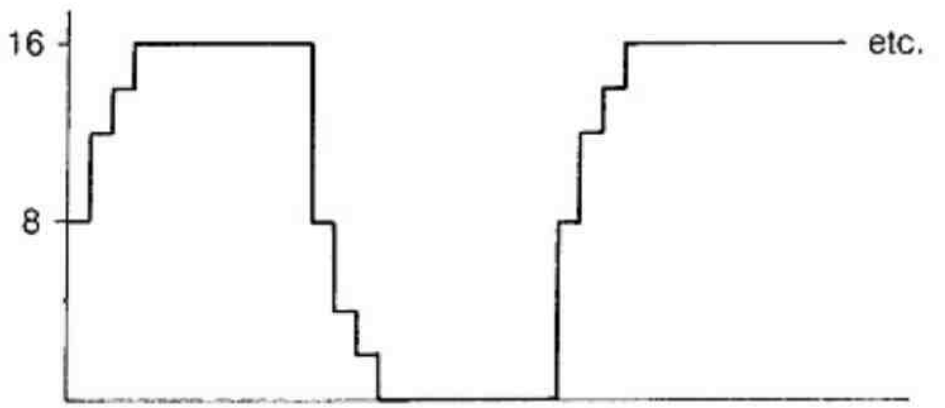
ENVELOPE 0 16,10 ; 0, 10 ;

définit un volume comme suit :



format du volume après une commande SOUND. Cela donnera une série de «BIP-BIP» l'instruction ENVELOPE 0 16 n'aura aucun effet sur une commande SOUND puisque tous les volumes seront multipliés par 16/16 soit 1.

NOTE : L'interpréteur BASIC limite la rapidité avec laquelle le volume est admis à changer (quelque soit le canal). Ce phénomène d'amortissement se produit à partir de D/2 + 1, où D est la différence entre le volume demandé et le volume précédent. Aussi, la forme du volume réel pour une commande ENVELOPE 0 16,10 ; 0, 10 ; sera :



Le changement de volume provoque une déformation du son.

Le générateur de bruit est contrôlé par la commande NOISE, seule l'enveloppe et le volume sont à préciser :
Par exemple : NOISE 0 15
cette commande signifie : mettre en marche le générateur de bruit en utilisant l'enveloppe 0 et le volume maximum, soit 15.

En plus des possibilités déjà décrites, la commande SOUND contrôle deux autres paramètres.
Il s'agit du TREMOLO et du GLISSANDO. Le trémolo est simplement une variation très rapide du volume de ± 2 unités. Le trémolo donne l'impression que le son «vibre». Par l'effet du glissando, la nouvelle note présentée sur le canal ne va pas démarrer à la fréquence spécifiée mais «glisser» de la fréquence précédente à la nouvelle fréquence. Cet effet ressemble à la guitare Hawaïenne ou au stylophone. Le glissando et le trémolo sont contrôlés par un seul argument dans la commande SOUND.

Par exemple : I) SOUND 00 13 1 FREQ (1000)
II) SOUND 00 15 2 FREQ (5000)

Le premier exemple valide le canal 0,utilise l'enveloppe 0 à un volume de 13 et avec le trémolo. (Le volume variera rapidement de 11 à 15).

Le dernier exemple monte le niveau du volume à 15 (le maximum) et fait «glisser» la fréquence jusqu'à 5000 Hz, grâce au glissando.

Remarque : en sortie stéréo :
le canal 0 est à droite
le canal 1 est à gauche
le canal 2 et le générateur de bruit sortent à droite et à gauche.

2.15.2.1. SOUND

I) SOUND <CANAL> <ENV> <T.G.> <PERIODE>
II) SOUND <CANAL> OFF
III) SOUND OFF.

<CANAL> est une expression dont la valeur est égale à 0,1 ou 2.
Cet argument sélectionne l'oscillateur 0,1 ou 2.

<ENV> est une expression dont la valeur est soit 0, soit 1 ; cet argument sélectionne une des deux enveloppes préalablement définies.

<VOL> est une expression dont la valeur est comprise entre 0 et 15. Cet argument sélectionne le volume pour un son en particulier. Le volume sera multiplié par les différents niveaux de volume spécifiés dans l'enveloppe choisie.

<T.G> est une expression dont la valeur est comprise entre 0 et 3.

0 ne sélectionne ni le trémolo ni le glissando
1 sélectionne le trémolo
2 sélectionne le glissando
3 sélectionne le trémolo et le glissando.

<PERIODE> est une expression dont la valeur A est comprise entre 2 et 65535. Cet argument positionne la période du son. La période est égale à A × 0,5 microseconde.

Normalement cette valeur A sera définie par la fonction FREQ (fréquence).

2.15.2.2. ENVELOPE

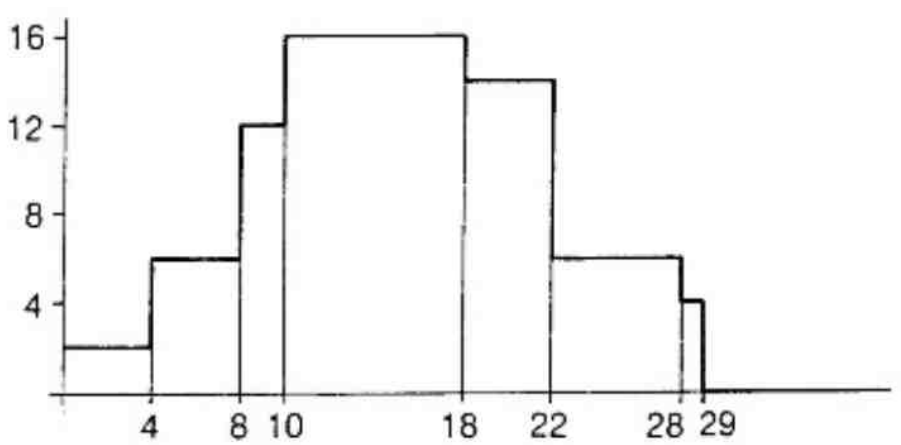
I) ENVELOPE <ENV> (<V>, <T> ;) <V>, <T> ;
II) ENVELOPE <ENV> (<V>, <T> ;) <V>

<ENV> est une expression dont la valeur est soit 0, soit 1. Cet argument sélectionne l'enveloppe à modifier.

<V> est une expression dont la valeur est comprise entre 0 et 16. Cet argument sélectionne un niveau de volume.

<T> est une expression dont la valeur est comprise entre 0 et 255. Cet argument définit le nombre d'unités de temps pendant lesquelles le niveau de volume V sera validé.

Exemple : ENVELOPE 0 2,4 ; 6,4 ; 12,2 ; 16,8 ; 14,4 ; 6,6 ; 4,1 ; 0 ceci signifie que l'enveloppe 0 aura un volume de 2 pendant 4 unités puis de 6 pendant 4 unités puis de 12 pendant 2 unités etc.



Si l'instruction ENVELOPE se termine par un chiffre seul, celui-ci indique le niveau du son jusqu'à la prochaine note.

Si l'instruction ENVELOPE se termine par un « ; » l'enveloppe est répétée.

Les notes de DO à DO représentent une octave. Une octave est un ensemble de notes allant de la fréquence de la note de base jusqu'au double de cette fréquence. Ainsi, pour produire une note (ex :DO) trois octaves plus haut, il suffit de multiplier 3 fois par 2 la fréquence de base.

Exemple : la fréquence du DO grave égale 65 Hz

la fréquence du DO suivant est de 65 × 2 soit 130 Hz

La différence de fréquence entre DO et DO# est appelée un DEMI-TON. Un demi-ton représente l'incrémentation de la fréquence dans un rapport de $\sqrt[12]{2}$ pour 1.

Donc freq (DO #) = freq (DO) * $\sqrt[12]{2}$
si DO = 65 Hz→DO # = 69 Hz

Remarque : $\sqrt[12]{2} = 2^{1/12} = 1{,}05946$

La différence de fréquence entre DO et RE est appelé un TON. Un ton représente l'incrémentation de la fréquence dans un rapport de $\sqrt[6]{2}$ pour 1

donc freq (RE) = freq (DO) * $\sqrt[6]{2}$
si DO = 65 Hz → RE = 73 Hz

Remarque : $\sqrt[6]{2} = 2^{1/6} = 1{,}12246$

$\sqrt[6]{2} = \sqrt[12]{2} * \sqrt[12]{2}$

1 ton = 2 demi-tons.

2.15.3.2. Génération des gammes

Il est possible de générer des notes dans une ou plusieurs gammes en incrémentant ou en décrémentant une fréquence de base pour le nombre désiré de demi-tons. Ainsi pour passer de DO à FA, l'incrément est de 2 tons plus 1 demi-ton soit 5 demi-tons.

freq (FA) = freq (DO) * $\sqrt[12]{2}$ * $\sqrt[12]{2}$ * $\sqrt[12]{2}$ * $\sqrt[12]{2}$

= freq (DO) * $(\sqrt[12]{2})^5$

Il convient de remarquer que 12 demi-tons séparent deux mêmes notes de deux gammes différentes donc :

freq (LA) sup. = freq (LA) * $(\sqrt[12]{2})^{12}$

= freq (LA) * $2^{12/12}$

= freq (LA) * 2

La note conventionnelle de base est le «LA» pour une fréquence égale à 440 Hz. En descendant de 9 demi-tons, nous pouvons générer un DO plus grave.

Exemple de programme généraux la séquence DO, RE, MI, FA, SOL, LA, SI, DO.

```
*LIST
100  CLEAR 1000:SOUND OFF
110  DEMI = 2.0↑(1.0/12.0):REM calcul de la valeur d'un demi-ton
120  DOINF = 440.0/(DEMI↑9.0):REM calcul de la fréquence du DO
130  ENVELOPE 0 16
140  FOR N% = 1 TO 8
150  READ D%
160  FRQ = DOINF*(DEMI↑D%)
170  SOUND 0 0 15 0 FREQ(FRQ)
180  WAIT TIME 20:SOUND OFF
190  NEXT N%
200  STOP
210  DATA 0,2,4,5,7,9,11,12
*
```

Le programme suivant génère une montée chromatique sur 5 octaves.

Remarque : Une montée chromatique est une suite de notes dont l'écart est d'un demi-ton.

```
*LIST
100  CLEAR 1000:SOUND OFF
110  DEMI = 2.0↑(1.0/12.0)
120  DOINF = 440.0/(DEMI↑33.0)
130  ENVELOPE 0 16
140  FOR N% = 0 TO 59
150  IF N% MOD 12 = 0 THEN RESTORE
160  READ A$:FRQ = DOING*(DEMI↑N%)
170  SOUND 0 0 15 0 FREQ(FRQ)
180  PRINT A$, «FREQ = » ; FRQ
190  WAIT TIME 20:SOUND OFF
200  NEXT N%
210  STOP
220  DATA D0, D0#, RE,RE#,MI,FA,FA#,SOL,SOL#,LA,LA#,SI
```

*Tableau des neufs octaves correspondant aux possibilités de l'audition humaine

	DOo à SIo	DO à SI	DO, à SI,	DO, à SI,	DO, à SI,	DO, à SI,	DO, à SI,	DO, à SI,	DO, à SI,
DO	32.69 Hz	65.39 Hz	130.79 Hz	261.59 Hz	523.19 Hz	1 046.37 Hz	2 092.75 Hz	4 185.50 Hz	8 371.00 Hz
DO #	34.62 Hz	69.25 Hz	138.50 Hz	277.02 Hz	554.05 Hz	1 108.10 Hz	2 216.22 Hz	4 432.44 Hz	8 864.88 Hz
RE	36.68 Hz	73.37 Hz	146.78 Hz	293.56 Hz	587.01 Hz	1 174.02 Hz	2 348.05 Hz	4 696.11 Hz	9 392.22 Hz
RE #	38.84 Hz	77.70 Hz	155.44 Hz	310.88 Hz	621.66 Hz	1 243.28 Hz	2 486.58 Hz	4 973.18 Hz	9 946.36 Hz
MI	41.20 Hz	82.39 Hz	164.80 Hz	329.60 Hz	659.21 Hz	1 318.42 Hz	2 636.56 Hz	5 273.12 Hz	10 546.24 Hz
FA	43.64 Hz	87.30 Hz	174.61 Hz	349.22 Hz	698.44 Hz	1 396.88 Hz	2 793.76 Hz	5 587.52 Hz	11 175.04 Hz
FA #	46.21 Hz	92.45 Hz	184.91 Hz	369.82 Hz	739.64 Hz	1 479.29 Hz	2 958.59 Hz	5 917.18 Hz	11 834.36 Hz
SOL	48.98 Hz	97.96 Hz	195.93 Hz	391.86 Hz	783.73 Hz	1 567.46 Hz	3 134.92 Hz	6 269.84 Hz	12 539.68 Hz
SOL #	51.87 Hz	103.74 Hz	207.48 Hz	414.97 Hz	829.97 Hz	1 659.94 Hz	3 319.88 Hz	6 639.77 Hz	13 279.54 Hz
LA	55.00 Hz	110.00 Hz	220.00 Hz	440.00 Hz	880.00 Hz	1 760.00 Hz	3 520.00 Hz	7 040.00 Hz	14 080.00 Hz
LA #	58.24 Hz	116.49 Hz	232.98 Hz	465.96 Hz	931.92 Hz	1 863.85 Hz	3 727.70 Hz	7 455.40 Hz	14 910.80 Hz
SI	61.73 Hz	123.46 Hz	246.94 Hz	493.88 Hz	987.57 Hz	1 975.13 Hz	3 950.27 Hz	7 900.54 Hz	15 801.08 Hz

2.15.4. Synthèse vocale

Grâce à la grande souplesse des commandes de volume et de fréquence, il devient possible d'explorer les possibilités d'une synthèse vocale. Le BASIC du DAI possède tous les contrôles pour permettre au programmeur de développer, expérimentalement, une «explosion» de sons pouvant aboutir à une synthèse vocale. La commande TALK offre davantage de possibilités pour permettre l'expérimentation.

2.15.4.1. TALK

exemple :
```
TALK # B000
TALK VARPTR (X(0))
```

TALK est a la fois une commande BASIC et un appel à un sous-programme écrit en langage machine. Les commandes TALK de l'exemple déclenchent l'exécution de «l'INTERPRETEUR CODE TALK» qui va décoder les codes stockés en mémoire à partir de l'adresse mémoire spécifiée. Les codes sont :

code instruction	nombre d'octets de données	Fonction
# 0	2	fréquence sur canal 0
# 2	2	fréquence sur canal 1
# 4	2	fréquence sur canal 2
# 8	1	contrôle volume canal 0
# 9	1	contrôle volume canal 1
# A	1	contrôle volume canal 2

# B	1	contrôle volume bruit
# C	2	durée (unité 13,5 μs)
# D	2	appel sous programme en machine
# FF	0	fin de TALK, retour au BASIC

Le contrôle du volume d'un canal particulier est permis en utilisant le «code instruction talk» approprié (# 0, # 2 ou # 4), suivi par deux octets définissant une fréquence dont l'unité est la demi-microseconde ($0{,}5.10^{-6}$ seconde).

Problème : comment déterminer le nombre hexadécimal donnant une fréquence de 800 Hz

PRINT HEX$ (FREQ (800)) affiche 9C4

L'ordre des deux octets suivant le «code fréquence», est :
<«code fréquence»> <partie basse de la freq.> <partie haute de la freq.>

ex. : # 0, # C4, # 9 définit sur le canal 0 une fréquence de 800 Hz.

L'unité de durée est de 13,5 μs ($13{,}5.10^{-6}$ seconde) et les deux octets suivants définissent le nombre d'unités requis pour la durée désirée. Une durée maximum de 0,9 seconde est possible ($65535 \times 13{,}5.10^{-6}$)

Remarque : l'ordre des deux octets suivant le «code durée», est :
<«code durée»> <partie basse de la durée> <partie haute de la durée>

ex. : # C, # 10, # 8 définit une durée égale à :
0810 $\times$ $13{,}5.10^{-6}$ seconde
soit 2064 $\times$ $13{,}5.10^{-6}$ seconde
soit 0,028 seconde

Les «codes instruction talk» sont interprétés pour la génération des sons.

Ils sont plus restrictifs que les codes du langage machine. Vous pouvez donc avoir besoin de fonctions plus sophistiquées. TALK offre la possibilité d'appeler un sous-programme écrit en code langage machine 8080. Ainsi les boucles, les modifications de données deviennent possibles. Les deux octets suivant le «code instruction talk» # 0 D définissent l'adresse de destination.

NOTE : Pour toutes les données codées sur deux octets, la partie basse doit être stockée avant la partie haute.

ex. : # D, # BC, # 50
Appelle le sous-programme se trouvant en mémoire à l'adresse hexadécimale 50BC.

Considérons le programme TALK suivant :

Adresse	Code talk	Donnée	Fonction
# A 0 10	0	# C4, 9	fréq. 800 Hz sur canal 0 (# 9 C4)
# A 0 13	8	# F	volume maxi sur canal 0
# A 0 15	# C	# FF, # FF	durée maximum
# A 0 18	8	0	volume minimum sur canal 0
# A 0 1A	2	# A, # 1A	fréq. 300 Hz sur canal 1 (# 1A0A)
# A 0 1D	9	# F	volume maximum sur canal 1
# A 0 1F	# C	# FF, # FF	durée maximum
# A 0 22	9	0	volume minimum sur canal 1
# A 0 24	4	# 20, # 4E	freq. 100 Hz sur canal 2 (# 4E20)
# A 0 27	# A	# F	volume maximum sur canal 2
# A 0 29	# C	# FF, # FF	durée maximum
# A 0 2C	# A	0	volume minimum
# A 0 2E	# D	# 31, # A0	appel du sous programme se trouvant en # A031

Pour cet exemple, nous allons écrire un sous-programme en langage machine qui permettra de réexécuter cette séquence à partir du «code instruction talk» stocké à l'adresse A010.

Adresse	Code machine	
A031	# 21, # 10, # A0	LXI H, A010
A034	# C9	RET

Le double registre HL charge l'adresse du prochain «code instruction talk» pour «l'interpréteur TALK».

Le programme suivant stocke et exécute les «codes instruction talk» donnés plus haut.

```
*LIST
10    CLEAR 1000:B% = #A010
20    READ A%:IF A%<0 GOTO 40
30    POKE B%,A%:B% = B% + 1:GOTO 20
40    TALK #A010
100   DATA 0,#C4,9,8,#F,#C,#FF,#FF,8,0
110   DATA 2,#A,#1A,9,#F,#C,#FF,#FF,9,0
120   DATA 4,#20,#4E,#A,#F,#C,#FF,#FF,#A,0
130   DATA #0D,#31,#A0,#21,#10,#A0,#C9,-1
*
```

On obtient le même résultat en écrivant les «codes talk» dans la zone mémoire réservée pour une variable dimensionée.

L'intérêt de ce programme est d'utiliser une zone réservée et non pas un emplacement mémoire arbitraire ; ce qui présente des inconvénients majeurs.

```
*LIST
10    CLEAR 1000:DIM Q%(100.0):B% = VARPTR(Q%(0.0))
20    READ A%:IF A%<0.0 GOTO 40
30    POKE B%,A%:B% = B% + 1:GOTO 20
35    REM la boucle 20,30 a charge les CODES-TALK en mémoire
40    READ F%:IF F%<0 GOTO 50:READ S%:GOSUB 1000:GOTO 40
50    TALK VARPTR(Q%(0.0))
60    STOP
100   DATA 0,#C4,9,8,#F,#C,#FF,#FF,8,0
110   DATA 2,#A,#1A,9,#F,#C,#FF,#FF,9,0
120   DATA 4,#20,#4E,#A,#F,#C,#FF,#FF,#A,0
130   DATA #0D,0,0,#21,0,0,#C9,-1
140   DATA 32,34,35,1,-1
150   REM le sous-programme en 1000 permet de calculer et de stocker
      les valeurs
160   REM après le CODE-TALK #D et après le code machine
      #21(LXI H)
1000  J% = 4:UF% = VARPTR(Q%(F%/J%)) + (F% MOD 4.0)-1.0
1010  US% = VARPTR(Q%(S%/J%)) + (S% MOD 4.0)-1.0
1020  POKE UF%,US% IAND #FF:REM IAND #FF annule la partie
      haute de l'adresse
1030  POKE UF% + 1,US% SHR 8:REM SHR 8 annule la partie basse
      de l'adresse
1040  RETURN
*
```

3. Les commandes «utilitaires» du moniteur

3.1. Introduction

retour

Ces commandes facilitent le développement et la mise au point des programmes en langage machine 8080. Elles permettent par exemple de visualiser le contenu des registres du microprocesseur lors de l'exécution d'un programme. Le contenu des mémoires peut être visualisé, changé, stocké ou chargé à partir d'une disquette ou d'une cassette. Ainsi, l'utilisateur peut facilement retrouver ses erreurs de programmation.

3.2. Structure des commandes «utilitaires» sous moniteur

Lorsque le contrôle est passé à un programme en code machine à l'aide des commandes GO ou LOOK, l'état des registres du microprocesseur est échangé avec l'image des registres utilisateur (cf. la commande «X»). Lors de la fin du programme ou d'une commande BREAK, le microprocesseur récupère l'état initial de ses registres et sauve les nouvelles valeurs des registres utilisateur.

Les vecteurs et les masques d'interruptions sont susceptibles d'être changé immédiatement par la commande «Vv». Par contre, les registres de contrôle G et T ne seront chargés qu'après une commande GO ou LOOK.

3.3. Présentation et format des commandes :

Pour avoir accès au moniteur, la commande BASIC est : *UT ↵

Le message suivant s'affiche : P.C. UTILITY V3.3. Le moniteur inscrit le caractère «>» qui indique l'attente d'une commande (comme «*» sous BASIC). Le format des commandes est toujours une seule lettre, suivie ou non, d'un ou plusieurs nombres.

Le premier nombre doit s'écrire directement derrière la lettre. Les nombres rentrés sont toujours en hexadécimal (Base 16) et sont validés par un SPACE ou un RETURN. Les commandes «utilitaires» ne tiennent compte que des derniers chiffres rentrés, les deux ou quatre derniers suivant le format de la commande.

Exemple :

- G12345678 équivaut à G5678
  car le format de cette commande est de 4 chiffres.
- F0000 ␣ 1048 ␣ 5566 équivaut à F0000 ␣1048 ␣66
  car le troisième nombre ne doit avoir que deux chiffres pour cette commande.

Si un nombre comporte moins de caractères que le format prévu, les caractères manquants seront remplacés par des zéros.

Exemple :

G3 G03 G003 G0003	Ces commandes sont identiques 3 équivaut à 0003

Lorsqu'une commande est erronée, le moniteur affiche «?». C'est le seul message d'erreur possible.

Lorsque le moniteur «LOOK» un programme ou affiche le contenu d'une zone mémoire la touche BREAK arrête l'affichage et rend le contrôle sous moniteur (affichage «>»).

La touche ← (flèche à gauche) est aussi utilisée pour sortie de certaines commandes (ESCAPE est le nom donné à cette touche).

Lors de la description des commandes, quelques symboles seront utilisés.

␣ pour space

↵ pour return

← pour escape (flèche à gauche).

Dans les exemples, les caractères soulignés sont rentrés au clavier.

Pour retourner sous BASIC, taper **B**.

3.4. Les commandes :

Ce paragraphe décrit trois classes de commandes qui sont :

Les commandes sur la mémoire :
elles aident l'utilisateur à vérifier son programme pendant que celui-ci s'exécute (ou en temps réel ou en mode «LOOK»). Elles peuvent aussi visualiser en Hexadécimal une partie de la mémoire, écrire ou déplacer des données en mémoire.

Les commandes sur les registres :
ces commandes permettent l'initialisation, la visualisation et la modification
- des images des registres 8080 de l'utilisateur
- des vecteurs d'interruptions
- du masque d'interruptions
- des registres de contrôle.

Les commandes sur les entrées/sortie :
Avec ces fonctions, l'utilisateur peut lire ou écrire des fichiers sur cassette ou disquette.

Les abréviations utilisées ci-dessous sont définies comme suit :

adr : adresse mémoire
ladr : adresse mémoire de départ
hadr : adresse mémoire de fin
dadr : adresse mémoire de début
byte : donnée (octet).

- Le format des adresses est de 4 chiffres hexadécimaux (base 16).

Remarque : si le format comporte plus de 4 chiffres, le moniteur ne tient compte que des 4 derniers. Si l'utilisateur se trompe de touche en écrivant une adresse, il lui suffit de la retaper en laissant les chiffres erronés.
(**DELCHAR** provoque un message d'erreur).

- Le format des données est de 2 chiffres hexadécimaux. Si le format comporte plus de 2 chiffres, le moniteur ne tient compte que des 2 derniers.

1ère classe : *les commandes sur la mémoire*

3.4.1. LOOK : Ldadr ␣ *ladr* ␣ *hadr* ↵

La fonction LOOK charge les registres du 8080 avec l'image des registres utilisateur, puis exécute le programme (adresse de début défini par dadr), instruction, par instruction et visualise éventuellement le contenu des registres du 8080 après chaque instruction.

Après **RETURN**, le moniteur charge PC (pointeur de programme du 8080) avec la première adresse tapée après L ; les deux autres adresses indiquent le début et la fin du mode de visualisation des registres.

Cette fonction est très utilisée pour cerner les erreurs de programmation. Une partie du programme sera exécutée en temps réel, sans visualisation des registres, jusqu'à l'adresse de début de mode-visualisation. Dans cette partie, l'exécution se fera en pas à pas avec affichage des registres à chaque pas jusqu'à l'adresse de fin de mode-visualisation.

Ensuite l'exécution reprend normalement sans visualisation.

La zone à visualiser est spécifiée par l'adresse de départ et l'adresse de fin incluse (de ladr jusqu'à hadr).

Les registres sont affichés de la façon suivante :

I = 1043 A = 02 F = 12 B = 00 C = 0A D = 00 E = 10 H = 7B L = 00 S = F8FC P = 1044

I représentant l'adresse de l'instruction exécutée, P : l'adresse de la prochaine instruction à exécuter.

Attention : avant de taper la commande LOOK, réinitialisez les vecteurs d'interruptions, les masques d'interruptions et SP par la commande Z3 (LOOK se sert de l'interruption 0).

Pendant l'exécution d'un programme par LOOK la touche **BREAK** est autorisée. Après **BREAK**, le contrôle est rendu au moniteur (affichage « > »), laissant le choix à l'utilisateur de modifier des registres, des cases mémoires, tout en pouvant ensuite relancer le programme sous mode LOOK. La commande serait : L↵.

Le programme en langage machine serait de nouveau exécuté à partir de l'adresse contenue dans P (l'utilisateur a pu la modifier) et avec l'image des registres utilisateur (qui ont pu être modifiés également), si l'on n'a pas modifié P le programme s'exécutera à l'adresse où le **BREAK** l'avait arrêté.

L'adresse de départ et l'adresse de fin de visualisation sont les mêmes qu'avant le **BREAK**. Pour les changer, la commande est :

LPadr ␣ hadr ↵ (ne change pas P).

Sous mode «LOOK», la touche **BREAK** est toujours active, que le moniteur affiche ou non les registres. Ceci permet de sortir d'une boucle de programme sans perdre le compteur de programme P.

3.4.2. DISPLAY : D ladr ␣ hadr ↵

Cette commande visualise en hexadécimal le contenu des cases mémoires de l'adresse de départ «ladr» jusqu'à l'adresse de fin «hadr» incluse. L'affichage est formaté pour ne pas visualiser plus de 16 données hexa par ligne. Au début de chaque ligne, l'adresse de la première donnée est affichée.

La touche BREAK arrête l'affichage et rend le contrôle au moniteur (affichage «>»).

Ex. : on veut visualiser les données de l'adresse A000 jusqu'à A005.

Commande : D A000 ␣A005.

Réponse A000 00 02 00 00 20 30

Remarque : si l'adresse de départ est supérieure à l'adresse de fin, seule la donnée à l'adresse de départ est affichée.

3.4.3. FILL : F ladr ␣ hadr ␣ byte ↵

La zone mémoire définie par l'adresse de départ (ladr) et l'adresse de fin (hadr) incluse est remplie avec la donnée (byte). Si byte n'est pas précisé, la mémoire sera remplie avec des zéros. Dans ce cas, on doit taper un SPACE avant RETURN sinon, une erreur sera détectée et la commande ne sera pas exécutée.

Exemple :
- on veut écrire FF de l'adresse 1010 à 101A inclus.
  Commande : F 1010 ␣ 101A ␣ FF ↵
- On veut écrire 00 aux mêmes adresses.
  Commande : F 1010 ␣ 101A ␣↵ ou même F 1010 ␣ 101A␣␣

Si l'adresse de départ est supérieure à l'adresse de fin, la donnée n'est écrite qu'une seule fois à l'adresse de départ.

3.4.4. GO : G dadr ↵

La commande GO lance le programme à partir de l'adresse définie par «dadr». Le retour sous moniteur se fait lorsque le programme exécute l'instruction RET (code machine : C9). La commande GO provoque donc un «call» à l'adresse «dadr». Si celle-ci n'est pas précisée le «call» se fera à l'adresse pointée par P (le pointeur de programme). A n'importe quel moment l'exécution du programme peut être arrêtée par la touche BREAK.

Exemple :
G 1040 ↵ exécute le programme dont l'adresse est 1040

G ↵ exécute le programme dont l'adresse est dans P.

3.4.5. MOVE : M ladr ␣ hadr ␣ dadr ↵

Cette commande recopie le bloc mémoire compris entre l'adresse de départ «Padr» et l'adresse de fin «hadr» incluse dans la zone mémoire qui commence à l'adresse «dadr».

Exemple :
on veut recopier la zone mémoire de 1000 à 100A à partir de 1011.
Commande : M1000 ␣ 100A ␣ 1011↵
avant cette commande la mémoire contenait par exemple :

```
1000  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F  00
1010  FF  FF  FF  FF  FE  FF  FF  FF  FF  FD  FE  FE  FC  FD  12  FF
```

après cette commande elle aura :

```
1000  02  03  04  05  06  07  08  09  0A  0B] 0C  0D  0E  00
1010  FF [01  02  03  04  05  06  07  08  09  0A  0B] FC  FD  12  FF
```

Remarque :
la zone comprise entre 1000 et 100A n'est pas modifiée.

La commande MOVE est utilisée quand une instruction doit être insérée dans un programme machine stocké en mémoire.

Exemple : un programme (ou une table) commence en 1040 et finit en 1075.
On veut insérer 3 octets en 1046, 1047, 1048.
Il faut donc faire un MOVE pour déplacer la zone commençant en 1046 de 3 octets, donc à partir de l'adresse 1049.

Remarque : MOVE ne modifie aucune donnée (attention aux JMP et CALL dans les programmes) commande : M1046 ␣ 1075 ␣ 1049 ↵
Avant cette commande la mémoire avait par exemple :

```
1040 01 02 03 04 05 06 0A 0B 0C 0D 0E 0F 10 11 12 13
1050 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23
1060 24 25 26... etc.
```

Après cette commande elle aura :

```
1040 01 02 03 04 05 06 0A 0B 0C 0A 0B 0C 0D 0E 0F 10
1050 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20
1060 21 22 23 24... etc.
```

On pourra donc écrire
en 1046 à la place de 0A, 07
en 1047 à la place de 0B, 08
en 1048 à la place 0C, 09
grâce à la commande SUBSTITUTE :

3.4.6. SUBSTITUTE : Sadr ↵

Cette commande permet de modifier les octets commençant à l'adresse adr.

Après avoir tapé **RETURN**, apparaît la donnée contenue à l'adresse adr

1 - vous voulez la modifiez, écrivez la nouvelle valeur et validez par **SPACE** ou **RETURN**

2 - vous ne voulez pas la modifier. Taper **SPACE** ou **RETURN**

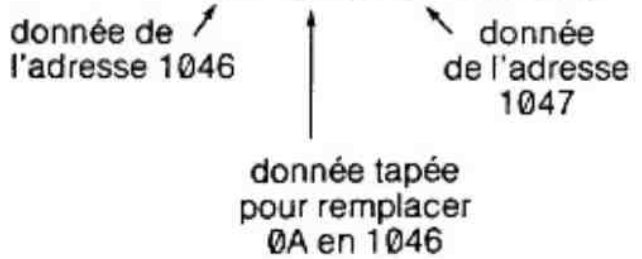
Après **RETURN** la donnée de l'adresse suivante apparaît en début de ligne (même procédure pour la modifier ou non). Après **SPACE**, la donnée de l'adresse suivante apparaît sur la même ligne, séparée de la donnée précédente par un **SPACE** (même procédure pour la modifier au non).

pour sortir du mode SUBTITUTE tapez **SPACE** (←)

Exemple : on veut écrire 07 à l'adresse 1046
08 à l'adresse 1047
09 à l'adresse 1048

Commande S 1046 ↵

0A – 07 ␣ 0B – 08 ␣ 0C – 09 ←



Remarque : ESCAPE ⬅ écrit aussi la donnée en mémoire puis rend le contrôle au moniteur.

ou identique :

S1046 ↵

0A-07 ↵

0B-08 ↵

0C-09 ←

Remarque : Si un seul chiffre est tapé, les dizaines seront considérées comme égales à zéro.

Exemple : S 1056 ↵

67-5 ← cette séquence à écrit 05 à l'adresse 1056.

2e classe : les commandes sur les registres.

3.4.7. EXAMINE : X↵

La commande «X» suivie de RETURN provoque l'affichage de l'image des registres du 8080 de l'utilisateur. Ce sont ces valeurs qui seront chargées dans les registres du 8080 lors d'une commande «GO» ou «LOOK».

Ces registres sont l'accumulateur A, les flags F, les registres de B à L, le pointeur de pile S et le pointeur de programme P.

Exemple : X↵

l'affichage apparaît de la façon suivante.

A = 00 F = 46 B = 01 C = 40 D = 80 E = C3 H = E6 L = 9E S = F8F8
P = EA04

Remarque : les flags sont rangés dans F comme suit :

S	Z	0	AC	0	P	1	C
D_7							D0

C : Carry : Indique si le résultat de la dernière opération arithmétique ou logique a provoqué un dépassement de capacité (une retenue)

1 :	bit toujours à 1
P : parité :	Indique si le résultat de la dernière opération arithmétique ou logique avait un nombre pair de bit à 1.
0 :	bit toujours à 0.
AC : carry auxiliaire	Indique s'il y a eu un dépassement de capacité lors d'un DAA (Décimal Ajust Accumulator)
2 : zéro	Indique si le résultat de la dernière opération arithmétique ou logique a provoqué un résultat nul.
S : Signe	Indique si le résultat de la dernière opération arithmétique ou logique a mis le bit 7 à 1 du registre concerné.

3.4.8. EXAMINE REGISTER : X reg

La commande «X reg» est similaire à la commande «S adr»

Avec S adr, on écrit, ou non, en mémoire

Avec X reg, on écrit, ou non, dans l'image des registres.

Exemple : On veut écrire 26 dans l'accumulateur A et le registre B. Taper :

```
XA 00-26 ␣ 46 - ␣ 01 - 26 ←
   ↑       ↑     ↑
affiche  affiche affiche
le       le     le
contenu  contenu contenu
de A     de F   de B
```

F n'est pas modifié.
Cette commande équivaut aux commandes suivantes :

XA 00 - 26 ←
XB 01 - 26 ←

Remarques : la procédure d'écriture, d'affichage de la donnée suivante et de sortie sont les mêmes que pour SUBSTITUTE.

– après avoir modifié (au non) le double registre P, le retour sous moniteur est automatique (c'est le dernier des registres)

3.4.9. VECTOR EXAMINE : V ↵

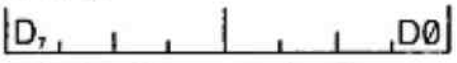
la commande «V» suivie de **RETURN** provoque l'affichage
- du registre des masques d'interruption M
- du registre de contrôle de l'interface parallèle G
- du registre T contrôlant la transmission Série et les interruptions
- des huits renvois d'interruption (il existe huit interruptions sur le DAI).

format de M : | D, | | | | | | | D0 |

Dx = 0 désactive l'interruption X

Si M = C5 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |

es interruptions 0, 2, 6 et 7 sont autorisées (interruption 6→clavier interruption 1→ est utilisé par LOOK).

format deG : 

D0 - 1 : PC (0→3) en entrée 0 : en sortie
D1 - 1 : PB (0→7) en entrée 0 : en sortie
D2 - 1 : mode 1 0 mode 0 : pour Pb et Pc (0→3)
D3 - 1 : Pc (4→7) en entrée : 0: en sortie
D4 - 1 : PA (0→7) en entrée 0 : en sortie
D5 - 00 mode 0 01 mode 1 10 ou 11 mode 2
D6 pour Pc (4→7) et PA
D7 - mode set flag (toujours à 1 en mode 0)

Format de T :

D0-1 pour Reset
D1-serial output break (1 to send break)
D2-0 interruption 7 sur timer 5, 1 : Interruption 7 sur extérieur
D3-0 interruptions dévalidées, 1 : interruptions validées
D4,5,6-0 : entrée/sortie série dévalidée.
1 : 110 baud
2 : 150 baud
3 : 300 baud
4 : 1200 baud
5 : 2400 baud
6 : 4800 baud
7 : 9600 baud
} vitesse de transmission
D7- 0 : 2 bits de stop, 1 : 1 bit de stop.

Les huit renvois d'interruption de 0 à 7 permettent de changer les branchements provoqués par les interruptions. La routine en début de chaque interruption est la suivante :

```
Push PSw
Push B
Push D
Push H
LHLD Vv (Vv est une adresse)
PCHL
```

Ce programme réalise un branchement à l'adresse contenue en Vv et Vv + 1

pour interruption 0 V_0 = 0062
pour interruption 1 V_1 = 0064
pour interruption 2 V_2 = 0066
pour interruption 3 V_3 = 0068
pour interruption 4 V_4 = 006A
pour interruption 5 V_5 = 006C
pour interruption 6 V_6 = 006E
pour interruption 7 V_7 = 0070

Exemple : V↵

M = C5 T = FC G = 1B 0 = EB5D 1 = C70E 2 = D9E2 3 = D755
4 = C6C0 5 = C6FD 6 = D578 7 = D9A9

3.4.10. VECTOR EXAMINE BYTES : Vv

v : M,T,G,0,1,2,3,4,5,6 ou 7

La commande «Vv» est similaire à la commande «X Reg»
Avec «X Reg», on écrit, ou non, dans l'image des registres
Avec «Vv», on écrit, ou non, dans les registres de contrôle ou dans les vecteurs d'interruptions.

Exemple : On veut changer le mot de contrôle de G par 9B

VG ␣ 1B − 9B ←

G contient maintenant 9B

Remarque : il est recommandé de ne pas changer les vecteurs d'interruptions.

3.4.11. INITIALIZE : Zn ↵

Ex. : Z1↵

Cette commande initialise l'accumulateur, les flags, les registres de B à L et le pointeur de programme à zéro. Le pointeur de pile S est initialisé en début de pile (F900)

Ex. : Z2↵

Cette commande initialise les registres de contrôle et les vecteurs d'interruptions.

Ex. : Z3↵

équivaut à Z1 + Z2.

3e classe : les commandes d'entrées/sorties

3.4.12. WRITE : W ladr ␣ hadr ↵

Les données hexadécimales contenues dans la zone mémoire définie par ladr et hadr sont enregistrées sur K7 ou sur disquette. Le format du transfert est défini ci-dessus :

Exemple : W 600 ␣ 60F ↵

format :

10 0600 00 B 7 2 D 000 8B C 7564440CFFFF20C5 H5 E5 CD 88

- 10 : nombre de données en hexa
- 00 : code 00:donnée 01: fin
- B 7 2 D 000 8B C 7564440CFFFF20C5 H5 E5 CD : données
- 88 : «check Sum» : c'est le nombre de 1 dans toutes les données

Exemple : W0 ␣ FFF ␣ GEORGE ↵

écrit la zone mémoire de 0000 à 0FFF sur disquette ou sur cassette derrière le nom GEORGE.

Ex. : W0 ␣ 1F ↵

écrit la zone mémoire de 0000 à 001F sans nom de fichier. (On ne doit pas écrire un fichier sans nom sur disquette).

3.4.13. READ : R↵

La fonction READ commencera à lire les fichiers hexa sur la cassette, lorsque l'utilisateur appuiera sur le START du mini-cassette. Si une erreur de «checksum» est détectée, le message d'erreur apparaît («?»), la lecture est arrêtée et le contrôle est rendu au moniteur («>»).

Les fichiers écris à l'aide de la commande «W» et lus par la commande «R» sont des fichiers de type 1.

Ils seront ignorés par les commandes BASIC : LOAD et LOADA. De la même façon, la commande «R» ne peut lire que des fichiers de type 1.

Le nom du fichier inclut tous les caractères entre l'espace et le Return.

ATTENTION : La touche DEL CHAR ne doit pas être utilisée.

Exemple : R ↵

Le prochain fichier hexa trouvé sur la cassette sera écrit en mémoire.
R ␣ FRED ↵

Quand le fichier hexa «FRED» sera trouvé, ce fichier sera écrit en mémoire. Si le nom du fichier n'est pas précisé, le moniteur charge le premier fichier rencontré.

R adr ↵

Le prochain fichier hexa trouvé sur la cassette sera écrit en mémoire à partir de son adresse d'origine plus adr (adr est considéré en hexa).
R adr ␣ FRED ↵

Lorsque le fichier hexa «FRED» sera trouvé, il sera écrit en mémoire à partir de son adresse d'origine plus adr.

- FRED était écrit en 1000
- R 10 ␣ FRED ↵ écrit le fichier FRED à partir de l'adresse 1010.

4. Le micro-ordinateur

4.1. Introduction

L'architecture de la carte du micro-ordinateur DAI est conçue autour du microprocesseur 8080 A. Cette carte se compose du microprocesseur avec ses circuits de décodage, de ROM (Read only memory), de RAM statique, d'un contrôleur d'interruptions avec une logique d'horloge, et de RAM (Random Accès memory) dynamique constituant les 48 KO de la mémoire utilisateur.

4.2. Organisation de la mémoire

L'espace mémoire du DAI est organisé de telle façon, que les Entrées/sortie soient adressables comme des cases mémoires. On dit que les Entrées/Sorties sont en mémoire banalisée.

Dans les explications suivantes, les adresses mémoires sont des nombres hexadécimaux. Les 64 KO adressables sont compris entre l'adresse 0000 et FFFF.

0000-003F	renvois d'interruptions
0040	contrôle sortie image
0041-0061	zone de travail du moniteur (images des registres)
0062-0071	vecteurs d'interruptions
0077-00CF	variables pour l'écran
00D0-00FF	zone de travail pour calcul mathématique
0100-02EB	variables BASIC
02EC	stockage des variables
jusqu'à la fin de la RAM	Programme Table des Symboles, Vidéo, RAM
F800-F8FF	zone pour la pile (stack)

Les doubles octets de variables BASIC qui suivent sont mis à jour par le système. En première adresse l'octet de poids faible, en deuxième adresse l'octet de poids fort.

Adresses :

#A2-#A3	adresse du début du Buffer d'édition
#A4-#A5	adresse de fin du buffer d'édition
#A6-#A6	adresse de début pour le «SAVE»
#029B-#029C	adresse du début du stockage des variables
#029D-#029E	longueur du stockage des variables
#029F-#02A0	adresse du début du buffer programme
#02A1-#02A2	adresse de fin du buffer programme et début de la table des symboles
#02A3-#02A4	adresse de fin de la table des symboles
#02A5-#02A6	adresse de fin de la vidéo-RAM

Exemple d'utilisation d'une de ces adresses :

```
*   REM PROGRAMME PERMETTANT D'OBTENIR EN DATA LES VALEURS
D'UN PROGRAMME ASSEMBLEUR
*0   CLEAR 20000 : INPUT «DEBUT DE LA MEMOIRE» ; DEB%:PRINT
20   INPUT «FIN DE LA MEMOIRE» ; FIN%:PRINT
30   INPUT «PREMIERE LIGNE» ; IP%:PRINT
40   POKE #131,2:POKE #A2,0:POKE #A4,0:POKE #A3,#A0:POKE
#A5,#A0
41   REM CHANGER #A0 EN UNE AUTRE VALEUR POUR UN GROS PRO-
GRAMME
50   FOR I% = DEB% TO FIN% STEP 16
60   PRINT IP%;«DATA»;
70   FOR J% = I% TO I% + 15
80   IF J%<>I% THEN PRINT «,»;
90   K% = PEEK(J%)
100  PRINT «#»;HEX$(K%);
110  NEXT
120  IP% = IP% + 1:PRINT
130  NEXT
140  PRINT CHR$(0)
150  POKE #131,0:POKE #135,2
160  END
```

4.3. Timers et contrôleur d'interruptions

Le micro-ordinateur DAI possède cinq timers programmables de 64 µs à 16 ms, deux interruptions externes et deux interruptions gérant les entrées/sorties série.

4.3.1. Contrôleur d'interruptions

Les huit adresses «renvoi d'interruptions» produites par le 8080A sont assignées aux fonctions suivantes :

Adresse hexa	fonction allouée
00	timer 1
08	timer 2
10	interruption externe
18	timer 3
20	réception-buffer plein
28	transmission-buffer vide
30	timer 4
38	timer 5/interruption auxiliaire

L'interruption externe est connectée à un signal qui indique qu'une écriture a été effectuée entre l'adresse F000 et l'adresse F7FF. Cette condition indique normalement un «STACK OVERFLOW».

L'interruption auxiliaire est connectée au signal «page» de la logique de l'image T.V. ; cela produit 20 µs d'horloge à l'intention du timing. La scrutation du clavier est déclenchée sur interruption.

Des caractéristiques plus complexes et précises vous sont expliquées tout au long de ce manuel (cf. para. 8.11), mais si vous avez besoin de plus d'informations, vous pouvez vous référer aux publications de DAI.
Par exemple : DCE MICROMPUTER SYSTEMS DESIGNER'S HANDBOOK»

4.4. *La mémoire RAM de 48 KO (master RAM)*

La mémoire «master» est divisée en trois parties séparées, appelées A,B,C. A une restriction près, chaque bloc mémoire contient 16 K octets de RAM dynamique.

La capacité mémoire est de 48 K octets (3 × 16).

L'adressage de la RAM dynamique est contrôlée par une simple ROM programmée pour valider la configuration RAM désirée.

La mémoire «Master» est vue par le programme comme un bloc mémoire continu, commençant à l'adresse 0000 et finissant en BFFF.

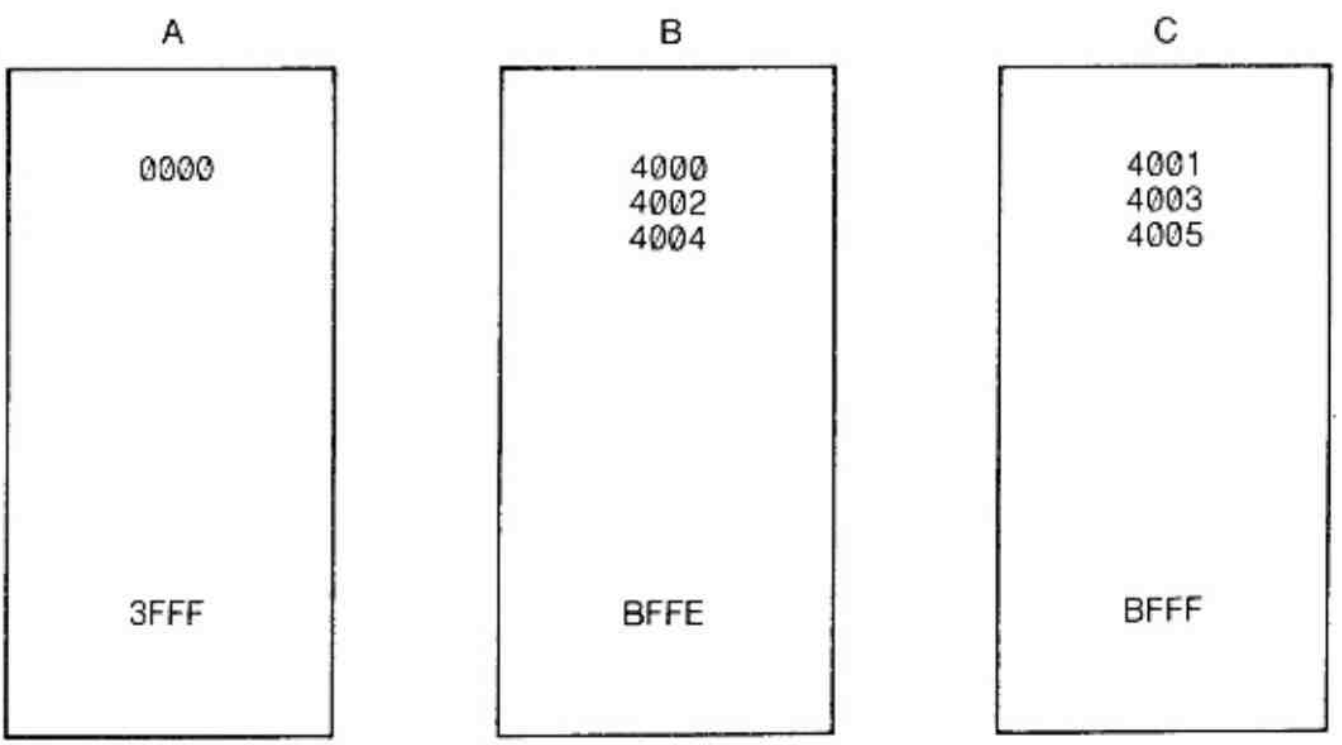
Le premier bloc mémoire commence à l'adresse 0000 et est disponible pour stocker un programme mais ne peut en aucun cas servir de vidéo-RAM. Les deux autres blocs sont configurés en 16 bits (2 octets) pour permettre l'accès de la logique de contrôle de l'écran.

Le bloc B contient les huit bits de poids faible, le bloc C contient les huit bits de poids fort du mot de 16 bits. Par contre, pour le microprocesseur, les adresses paires sont dans le bloc B les adresses impaires dans le bloc C.

Le bloc A contient donc 16 Koctets occupant les adresses 0000 à 3FFF puis l'adresse 4000 se trouve dans B, 4001 dans C, 4002 dans B etc. jusqu'à la fin de la RAM «master».

Le codage spécial des deux blocs B et C permet au microprocesseur de travailler sur des mots de 8 bits et permet à la logique de contrôle de l'écran d'avoir accès directement à des mots des 16 bits.

Nous avons donc 32 K maximum de disponible pour la vidéo RAM.



4.4.1. Organisation de la vidéo-RAM suivant les modes.

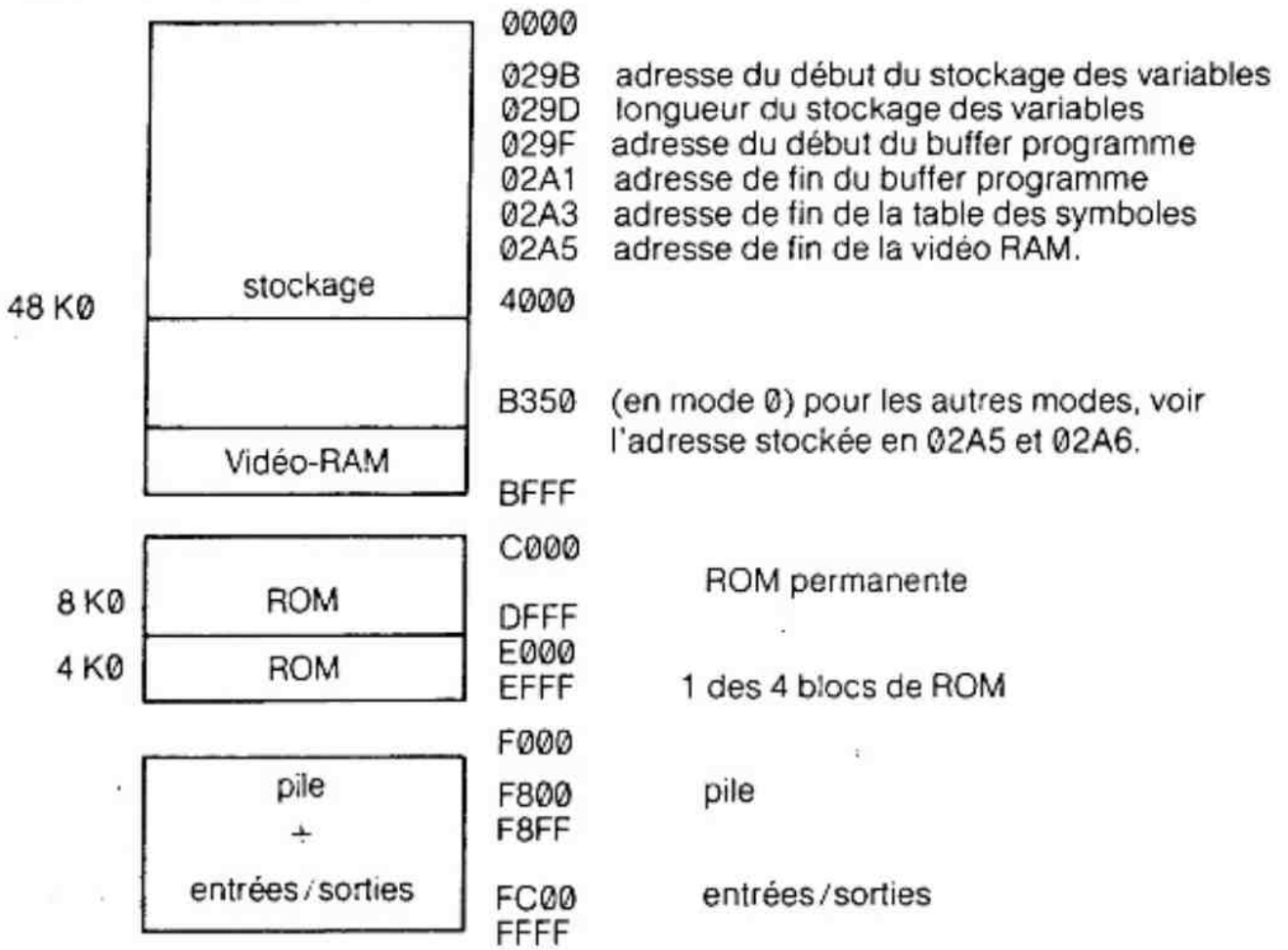
MODE	DIMENSIONS GRAPHIQUE	DIMENSIONS TEXTE	NOMBRE DE COULEURS	DIMENSIONS VIDEO-RAM	DIMENSIONS RESTANTES	NOTE
0	–	24 × 60	2	4,2 K	43,8 K	Texte
1	72 × 65	–	16	2,6 K	45,4 K	
1A	72 × 65	4 × 60	16	3,1 K	44,9 K	
2	72 × 65	–	4	2,6 K	45,4 K	
2A	72 × 65	4 × 60	4	3,1 K	44,9 K	
3	160 × 130	–	16	6,9 K	41,1 K	
3A	160 × 130	4 × 60	16	7,5 K	40,5 K	
4	160 × 130	–	4	6,9 K	41,1 K	
4A	160 × 130	4 × 60	4	7,5 K	40,5 K	
5	336 × 256	–	16	23,6 K	24,4 K	
5A	336 × 256	4 × 60	16	23,8 K	22,1 K	
6	336 × 256	–	4	23,6 K	24,4 K	non disponible mais possible cf. § 5.2.
6A	336 × 256	4 × 60	4	23,8 K	22,1 K	
–	528 × 240	–	4 ou 16	32 K	16 K→	

4.5. ROM et RAM statique

Le programme système réside dans trois ROM implantées de C000 à EFFF. Ce programme est continu, de l'adresse C000 à l'adresse DFFF (8K0), tandis que de E000 à EFFF (4K0) il existe quatre blocs de 4 K0 chacun. L'accès à une des quatre parties est validé lorsque le microprocesseur en a besoin. Le programme système stocké en ROM est de 24 koctets (8 + (4 × 4)) mais 12 K octets seulement sont présents sur le bus d'adresse.

1 K octet de RAM statique, compris entre F800 et F8FF, est utilisé pour le STACK POINTER du 8080 (pile) et pour l'émulation du système par un système MDS (intellec.).

4.5.1. La memory-map



5. Le graphisme programmable

5.1. Introduction

Le système vidéo-programmable (graphique ou caractères) utilise un arrangement mémoire, variable suivant le mode et l'image désirée.

Quelques définitions sont nécessaires avant de plus amples explications.

– *Le balayage :*
C'est la traversée de l'écran par le spot lumineux du canon à électrons de votre moniteur T.V. ou de votre téléviseur. 625 balayages définissent une image sur une télévision aux normes européennes.

– *La ligne :*
C'est l'ensemble des balayages contrôlés par une même information mémoire.

– *Le mode :*
C'est une des différentes sortes de codification des informations mémoire pouvant être affichées sur l'écran. Par exemple, les octets en vidéo-RAM représentent en mode texte les codes ASCII des caractères, et en mode «quatre couleurs graphiques», les couleurs des différents points de l'écran.

– *Le point :*
C'est la plus petite zone de l'écran où une couleur peut être définie. La dimension du point est différente suivant le mode choisi. C'est l'instruction DOT qui permet d'allumer un point.

– *Le champ :*
C'est un ensemble de huit points-écran dont les couleurs sont contrôlées par un octet en mémoire.

Une image est définie par un nombre de lignes. Chaque ligne est indépendante et peut être initialisée dans n'importe quel mode possible. En début de chaque ligne, les deux premiers octets sont appelés de la mémoire pour définir le mode de cette ligne et éventuellement, le changement d'un registre de couleur. Ils sont nommés respectivement «octet de contrôle de mode» et «octet de contrôle de couleur».

Le reste de la ligne est constitué d'informations de caractères ou de couleurs. Le nombre d'octets nécessaire pour coder une ligne est une caractéristique de chaque mode.

Le nombre de lignes définissant l'écran est différent suivant le mode choisi. Dans le mode haute résolution (mode 5 ou 6) il existe 336 points adressables par ligne. Les deux autres résolutions graphiques ont respectivement comme définition la moitié et le quart de 336 points par ligne.

Les caractères alphanumériques sont définis dans une grille utilisant huit colonnes par caractère. La position des points définissant chaque symbole est codée dans une ROM appelée «générateur de caractères».

En mode «texte» chaque ligne comporte 66 caractères mais le BASIC n'utilise que 60 caractères.

L'étoile du basic est le 4e caractère d'une ligne.

En mode haute résolution vous pouvez visualiser jusqu'à 40 caractères alphanumériques. En mode moyenne résolution, 18 caractères alphanumériques. En mode basse résolution, 7 caractères alphanumériques.

A l'aide de POKE, vous pouvez obtenir trois caractères de plus dans chaque mode.

Un total de seize couleurs, incluant le noir et le blanc, est généré par le système. Chaque couleur est codée sur quatre bits. Dans les différents modes «4 couleurs», un code de deux bits en mémoire suffit pour distinguer une des quatre couleurs initialisées par COLORG. Dans les différents modes «16 couleurs», les deux couleurs du champ précédent sont codées sur un octet (2 × 4 bits).

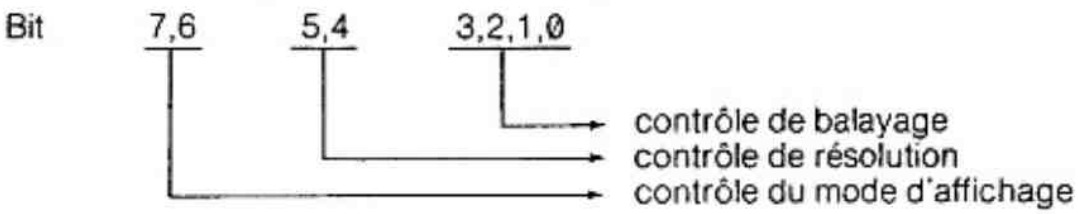
5.2. Format des données dans la vidéo-RAM

Au début de chaque ligne de donnée-écran, deux octets représentent le mot de contrôle de la ligne. Ce mot de contrôle définit le nombre de balayages («l'épaisseur»), la résolution, le mode de la ligne et éventuellement, le changement d'un registre de couleur.

A la suite de ce mot de contrôle de seize bits, sont stockés les octets représentant soit les couleurs de la ligne, soit les caractères ASCII et les codes couleurs selon le mode d'affichage choisi.

5.2.1. Format du mot de contrôle

5.2.1.1. L'octet de contrôle d'adresse haute (octet de mode)



Contrôle de balayage :

Ces quatre bits contrôlent le nombre de balayages pendant lesquels une seule ligne sera affichée. Le minimum est de deux balayages et correspond à un «contrôle de balayage» de zéro. Chaque incrémentation de ce mot de quatre bits ajoute deux balayages. Le nombre maximum de balayages programmable est donc de 32. Ces quatre bits définissent l'«épaisseur» de la ligne.

exemple :
```
MODE 0
POKE #BF69, #74    raccourcit la ligne n° 2 (moitié du caractère)
POKE #BF69, #7F    agrandit la ligne n° 2 (maximum)
POKE #BF69, #7A    état initial de la ligne n° 2.
```

Contrôle de résolution

Les deux bits de contrôle de résolution permettent de choisir pour chaque ligne une des quatre définitions horizontales possibles pour l'affichage des données sur l'écran.

Code (bit 5, bit 4)	*Définition* (nombre de points / ligne)
00	88 basse résolution
01	176 moyenne résolution
10	352 haute résolution
11	528 texte avec 66 caractères par ligne. Résolution pouvant être utilisée pour la très haute résolution graphique.

Exemple de modifications de définition en mode 0

```
*
*LIST
10  MODE 0:COLORT 0 10 0 0
20  CURSOR 0,12:PRINT «BONJOUR»
30  A% = #7A
40  POKE #BA2D,A%:REM Adresse de l'octet de mode de la ligne 12
50  WAIT TIME 50:A% = A%-#10:IF A% = #3A GOTO 30
60  GOTO 40
```

d'une manière générale

4A en octet de mode donne de très grandes lettres (-basse définition)
5A en octet de mode donne de grandes lettres (-moyenne définition)
6A en octet de mode donne des lettres moyennes(-haute définition)
7A état initial – mode 0.

Contrôle du mode d'affichage

Les deux bits du contrôle de mode d'affichage spécifient si les octets de cette ligne seront : soit des codes ASCII en quatre couleurs, soit des codes ASCII en seize couleurs, soit du graphique en quatre couleurs, soit du graphique en seize couleurs.

code (bit 7, bit 6)	*mode d'affichage*
00	4 couleurs graphique
01	4 couleurs caractère
10	16 couleurs graphique
11	16 couleurs caractère

5.2.1.2. L'octet de contrôle d'adresse basse (octet de couleur)

L'octet d'adresse basse est utilisé pour éventuellement changer une couleur d'un des quatre «registre couleur».

Un seul registre peut être changé au début de chaque ligne d'affichage de données. Seules les couleurs définies dans les registres peuvent être affichées en mode «4 couleurs». Ces couleurs sont à choisir librement parmi les seize couleurs définies dans la table des couleurs.

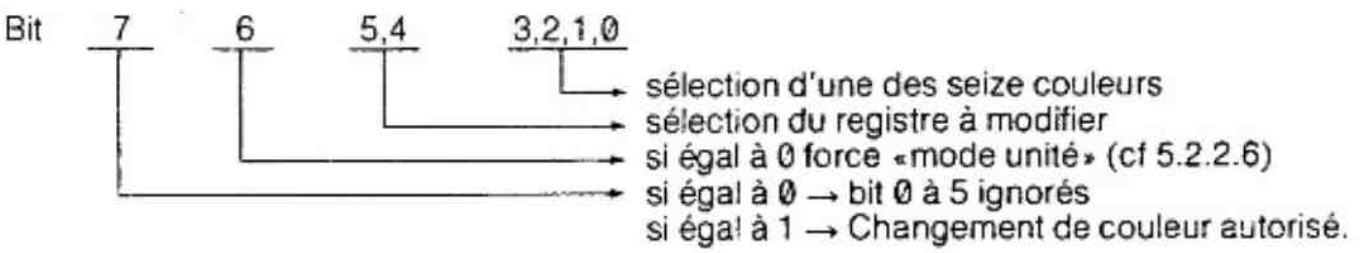


Table des couleurs

décimal	*code*	bit 3,2,1,0	*couleur*
0	0	0000	noir
1	1	0001	bleu vif
2	2	0010	parme
3	3	0011	rouge vif
4	4	0100	vert kaki
5	5	0101	vert pré
6	6	0110	rouge foncé
7	7	0111	violet
8	8	1000	gris
9	9	1001	bleu marine
10	A	1010	orange
11	B	1011	rose saumon
12	C	1100	bleu clair
13	D	1101	vert clair
14	E	1110	jaune
15	F	1111	blanc

code (bit 5, bit 4)	*Registre modifié*
00	Registre n° 1
01	Registre n° 2
10	Registre n° 3
11	Registre n° 4

avec COLORT ou COLORG <Reg 1> <Reg 2> <Reg 3> <Reg 4>

exemple : MODE 0 : COLORT 0 10 0 0 – Lettres orange sur fond noir
POKE # BA2C, #DF – change la couleur des lettres en blanc à partir de la ligne n° 12
BA2C est l'adresse basse du mot de contrôle pour la ligne 12

d'une manière générale :

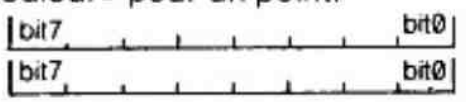
- POKE <adresse basse du mot de contrôle>, #DX ; avec X définissant une couleur entre 0 et F, change la couleur des lettres avec X (2e registre). # FX change le 4e registre définissant la couleur des lettres du 2e groupe
- POKE <adresse basse du mot de contrôle>, # CX ; avec X définissant une couleur entre 0 et F, change la couleur du fond avec X (1er registre). # EX change le 3e registre définissant la couleur de fond des lettres du 2e groupe.

5.2.2. Format des données suivant le mode d'affichage

5.2.2.1. mode graphique quatre couleurs

Dans ce mode, deux bits de données sont nécessaires pour définir la couleur d'un point. Ces deux bits sont en parallèle sur deux octets consécutifs. Chaque paire de bit sélectionne un des quatre «registres couleur» pour un point.

octet de donnée d'adresse haute
octet de donnée d'adresse basse



↑ point le plus à gauche ↑ point le plus à droite

bit n d'adresse haute	0	0	1	1	
bit n d'adresse basse	0	1	0	1	
	↓	↓	↓	↓	
	1er	2e	3e	4e	«registre couleur»

Remarque : les adresses basses sont toujours PAIRES.

exemple : MODE 2A
COLORG 0 1 3 5 (noir, bleu vif, rouge vif, vert pré)
POKE # BB20, 1 (point bleu, adresse H = 0 adresse B = 1)
POKE # BB21, 1 (point vert, adresse H = 1 adresse B = 1)
POKE # BB20, 0 (point rouge, adresse H = 1 adresse B = 0)

A l'initialisation, tous les octets de données sont positionnés à zéro donc à la première couleur. Les deux bits de chaque point sélectionnent une des quatre couleurs chargées dans les «registres couleur». Sur chaque ligne quatre couleurs sont autorisées, mais à la ligne suivante, une de ces couleurs peut-être changée grâce à son mot de contrôle d'adresse basse (cf. 5.2.1.2.).

5.2.2.2. Mode graphique seize couleurs

Ce mode graphique permet de définir une image en haute résolution multi-couleurs en utilisant moitié moins de mémoire que les autres systèmes. Huit points-écran sont codés sur seize bits.

L'octet d'adresse basse sélectionne deux des seize couleurs possibles :

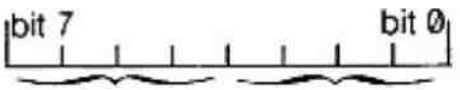
de Bit 0 → 3 couleur des zéros
de Bit 4 → 7 couleur des uns

l'octet d'adresse haute définit alors, pour chaque bit successif, la couleur choisie. (0 ou 1).

octet de donnée d'adresse haute :

point le plus à gauche ↑ point le plus à droite

octet de donnée d'adresse basse :



couleur des bits de l'adresse haute à 0. couleur des bits de l'adresse haute à 1.

A l'initialisation, toutes les adresses basses contiennent 0 et toutes les adresses hautes # FF (tous les bits à 1).

huit points d'écran sont codés sur seize bits.

chaque bit de chaque «champ de points» ne peut sélectionner que deux couleurs, soit celle des 0, soit celle des 1. Mais quelles que soient les deux couleurs choisies, elles sont totalement indépendantes des autres «champs de points». Ainsi, chaque ligne peut utiliser jusqu'à seize couleurs différentes. Les quatre «registres de couleur» ne sont pas utilisés dans les modes seize couleurs.

exemple : soit #BB00 l'adresse basse d'un champ en mode 1A
et #BB01 l'adresse haute de ce champ.

* MODE 1A

* POKE #BB00, # 9A choisit deux couleurs (bleu, orange).

Une barre bleue apparaît puisque #BB01 contient #FF

* POKE #BB01, #AA affiche un point bleu, un point orange, un point bleu, un point orange etc. car #AA = 10101010 en binaire

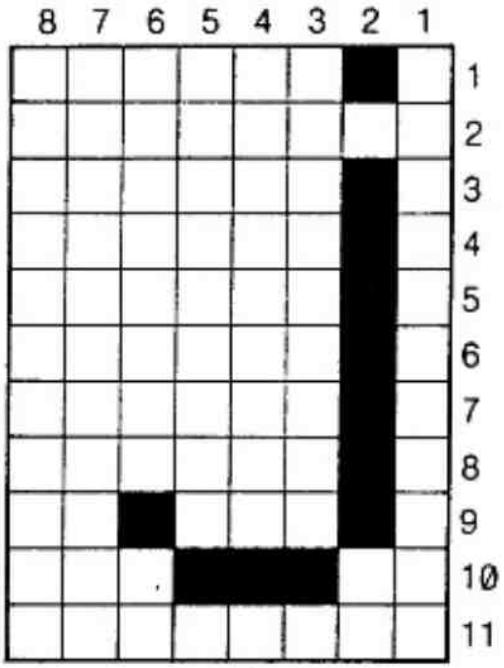
* POKE # BB01, # 55 ou POKE # BB00, # A9 décale les deux couleurs du champ défini.

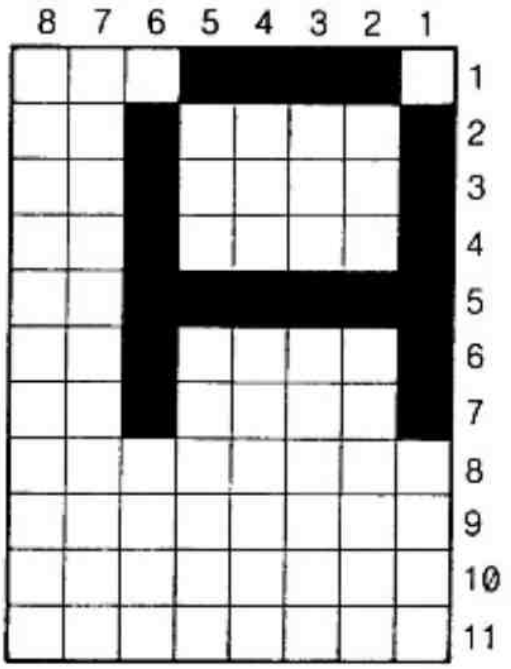
Remarque : un graphique en mode «4 couleurs» est plus vite défini que le même graphique en mode «16 couleurs».

5.2.2.3. Mode caractère quatre couleurs

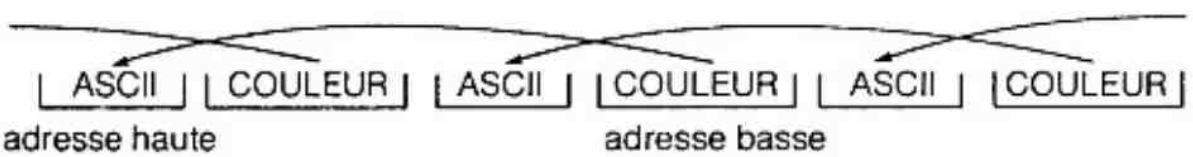
Dans ce mode, les différents caractères sont générés à partir d'une ROM appelée «générateur de caractères» et utilisent quatre registres de couleurs. La matrice moyenne d'un caractère est de 6 × 9 bits (6 × 7 «A», 6 × 10 pour «j») dans une matrice écran de 8 × 11 pouvant être modifiée à l'aide du contrôle de balayage jusqu'à 8 × 16 (voir chapitre 5.2.1.1.).

exemples de matrices de caractères :





L'arrangement mémoire en mode «4 couleurs caractère» (initialisé par MODE 0) est similaire au mode «4 couleurs graphique». Chaque caractère est défini sur deux octets. Le premier octet d'adresse haute contient le code ASCII. Le second octet d'adresse basse contient les indications pour la couleur de chacune des huit colonnes. «L'adresse haute» se situe à TROIS cases mémoire de «l'adresse basse» :



Les codes ASCII correspondent à différents dessins de caractères stockés dans le générateur de caractères. Les caractères sont affichés en utilisant les quatre couleurs définies dans les registres de couleur (initialisation par COLORT). Les quatre registres de couleur du mode texte sont différents des quatre registres de couleurs du mode graphique.

L'octet d'adresse basse (adresse haute − 3) définit une des deux combinaisons de couleurs pour chacune des huit colonnes du caractère. A chaque bit correspond une colonne ; la colonne 8 correspond au bit le plus significatif, la colonne 1 correspond au bit le moins significatif.

Chaque bit à zéro, sélectionne le premier registre couleur pour le fond et le deuxième pour le caractère (1ère combinaison).

Chaque bit à un, sélectionne le troisième registre couleur pour le fond et le quatrième pour le caractère (2e combinaison).

Exemple : soit #BA0F l'adresse haute contenant le code ASCII et soit # BA0C son «adresse basse»

- MODE 0
- COLORT 0 13 13 0 définit pour les registres 1 et 4, le noir et pour les registres 2 et 3, le vert clair
- POKE # BA0F, 106 affiche «j» sur la ligne 12, 12e position
- POKE # BA0C, #FF définit les huit colonnes avec le 3e et le 4e registre (soit caractère noir sur fond vert clair)

* POKE #BA0C, #F0 définit les colonnes 1 à 4 avec la 1ère combinaison et les colonnes 5 à 8 avec la 2e combinaison.

* POKE #BA0C, #0F définit les colonnes 1 à 4 avec la 2e combinaison et les colonnes 5 à 8 avec la 1ère combinaison.

exemple de programme :

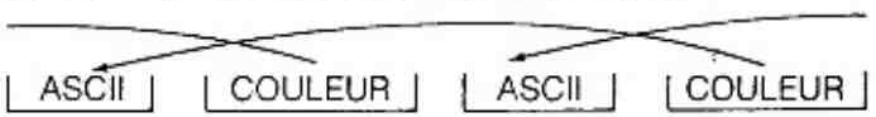
```
*LIST
5   COLORT 0 13 0 6
10  POKE #BA2D,#4A:POKE #BA1F,65
15  A% = 0
20  POKE #BA1C,A%:WAIT TIME 10:A% = A%*2 + 1
30  IF A%> #100 GOTO 15:GOTO 20
*
```

Remarque : – Après LIST, EDIT, MODE0, et PRINT CHR$(12) tous les octets d'adresse basse sont initialisés à zéro donc toutes les colonnes utilisent les deux couleurs de la première combinaison.

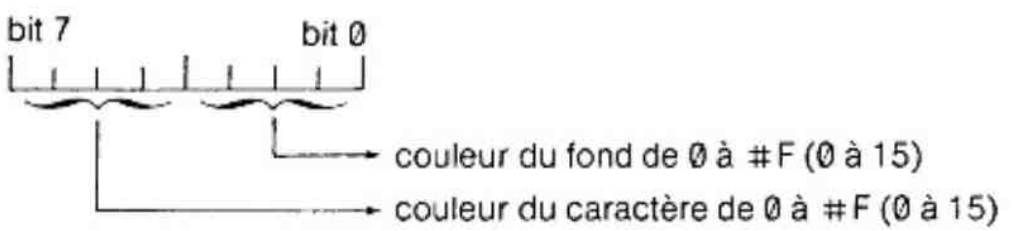
– Lors d'un RETURN sur la dernière ligne, les codes couleurs d'adresse basse gardent leur position physique sur l'écran.

5.2.2.4. Mode caractère seize couleurs

L'arrangement mémoire en mode «16 couleurs caractères» est similaire au mode «16 couleurs graphiques». Chaque caractère est affiché avec les deux couleurs définies dans l'octet d'adresse basse. Cet octet se trouve en mémoire, comme en mode «4 couleurs caractère» à – 3 adresses du code ASCII concerné.



format de l'octet de couleur d'adresse basse :



exemples :

```
*LIST
10  POKE #BDD7, #FA:REM octet de mode ligne 5
20  A% = #F
30  FOR I% = 30 TO 0 STEP -2
40  POKE (#BDA0 + I%),A%:A% = A% + #10:NEXT:REM chargement
des octets de couleur
50  CURSOR 7, 19:PRINT «CARACTERE*COULEUR»
*
```

```
*LIST
1   COLORT 0 10 0 0:INPUT T$:A% = #10:PRINT
2   POKE (PEEK(#79) SHL 8) + PEEK(#78), #E7
3   FOR T = 0.0 TO LEN(T$)-1.0:POKE #76,A%
4   A$ = MID$(T$,T,1):PRINT A$;:A% = A% + #10
5   IF A% = #100 THEN A% = #10
6   NEXT:PRINT:GOTO 1
*
```

Caractères ASCII spéciaux :

CR : termine une ligne de caractères et positionne le curseur à la première position de la ligne suivante. (13)

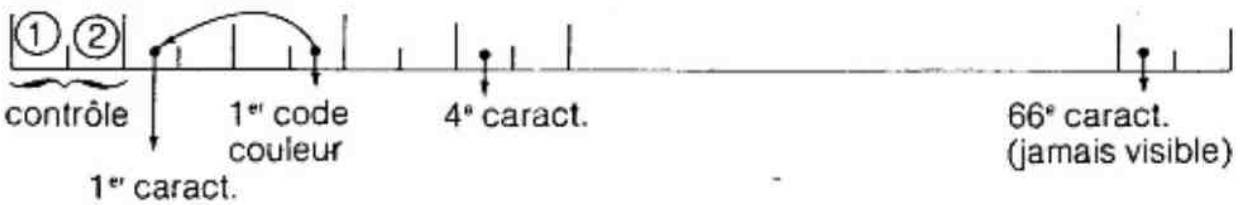
FF : efface l'écran et positionne le curseur en haut à gauche. (12)

BS : si la ligne comporte plusieurs caractères, le curseur retourne en arrière et efface le premier caractère rencontré. (127)

Remarques : – Si un programme définit l'écran en mode X graphique et si l'interpréteur BASIC exécute une instruction INPUT, l'écran passe automatiquement en mode XA.

– Pour effacer un écran défini dans un mode quelconque X, il suffit d'exécuter l'instruction MODE X. En mode 0, l'instruction mode 0 laisse 4 lignes d'écran en haut, c'est l'instruction PRINT CHR$ (12) qui efface entièrement l'écran en mode 0.

5.2.2.5. Format d'une ligne en mode zéro



contrôle 1 : contrôle de mode
contrôle 2 : contrôle de couleur
4e caractère : position de l'étoile sous BASIC.
63e caractère : position du dernier caractère d'une ligne visualisé sous BASIC.

contrôle de mode ligne 1	# BFEF
contrôle de mode ligne n	# BFEF – (#86*(n-1))
contrôle de mode ligne 5	# BFEF – (#86*(5-1)) = # BDD7
caractère n°X de la ligne 5	# BDD7 – X * 2
caractère n° 6 de la ligne 5	# BDD7 – 6 * 2 = # BDCB
octet de couleur du caractère 6 ligne 5	# BDCB – 3 = BDC8

Remarques : – lors d'un «PRINT» l'adresse mémoire #76 contient la valeur de l'octet de couleur à assigner au premier caractère du PRINT.

– Les cases mémoires # 79 et # 78 contiennent l'adresse de l'octet «contrôle de mode» de la ligne où se trouve le curseur.
79 contient la partie haute de l'adresse.
78 contient la partie basse de l'adresse.
Voir 2e exemple chapitre 5.2.2.4.

– La liste des adresses, de début de chaque ligne en mode 0 est donnée au chapitre 8.8.

5.2.2.6. Le mode unité.

Ce mode permet de définir une vidéo-RAM sur quarante octets seulement, laissant presque 48 Koctets pour le programme.

Chaque caractère est répété sur toute une ligne.

6. Le générateur de son programmable

6.1. Introduction :

Le générateur de son du micro-ordinateur DAI est extrêmement souple car chaque fréquence est générée par un oscillateur digital donnant des résultats très précis. Un générateur de bruits blancs et un contrôle de volume digital complètent le système.

6.2. Les oscillateurs programmables :

Le générateur de son programmable est réalisé avec trois oscillateurs programmables indépendants et un générateur de bruits blancs. Chaque oscillateur est connecté comme un dispositif d'entrée/sortie au microprocesseur et est programmable de 30 Hz à 1 MHz. Bien entendu les très hautes fréquences (> 20 KHz) ne sont pas utilisables pour des applications audios.

Les oscillateurs programmables sont utilisés pour la génération des sons et pour l'interface des paddles.

6.2.1. Sélection de la fréquence

Afin de programmer une fréquence sur un des canaux, un nombre de 16 bits doit être écrit dans une des adresses suivantes :

Canal oscillateur	Adresses
1	# FC00 ou # FC01
2	# FC02 ou # FC03
3	# FC04 ou # FC05

Avant d'écrire une fréquence sur un canal, l'adresse # FC06 doit être chargée avec un des octets qui suit :

canal 1	# 36
canal 2	# 76
canal 3	# B6

Les 16 bits définissant la fréquence doivent être écrits à l'adresse voulue via deux transferts de 8 bits, en commençant par les moins significatifs.

6.2.2. Contrôle du volume :

L'amplitude de la sortie de l'oscillateur ainsi que celle du générateur de bruits blancs est contrôlable digitalement en écrivant un mot de contrôle à l'une des adresses suivantes :

# FD04	bit 0	
	1	volume canal 1
	2	
	3	
	4	
	5	volume canal 2
	6	
	7	

# FD04	bit 0 1 2 3	volume canal 3
	4 5 6 7	volume générateur de bruits blancs.

6.3. *Le générateur de bruits blancs*

Un circuit générateur de bruits est inscrit dans le schéma du générateur de son. Le but de ce dispositif est de simuler un «bruit blanc» permettant la génération de sons complexes et de produire des séquences pour générer des nombres aléatoires. Les différents niveaux générés par ce circuit, fournissent au BASIC toutes les informations, pour générer de VRAIS nombres aléatoires.

6.4. *Le mélange des fréquences*

Tous les canaux de son, ainsi que la sortie du générateur de bruits sont mixés avant modulation. Les canaux 1 et 2 sortent sur le canal gauche de la sortie STEREO et les canaux 2 et 3 sortent sur le canal droit de la sortie STEREO. En stéréophonie, le bruit sort sur le canal droit et le canal gauche.

6.5. *Formule de calcul de fréquence :*

Pour sortir une fréquence de n Hz à partir d'un oscillateur donné, il faut le programmer avec un nombre entier égal à 2×10^6 divisé par n.

La fonction BASIC FREQ permet de calculer directement ce nombre.

exemple : ? HEX $ (FREQ(440)) donne # 11C1

? HEX $ (2 * 1000000/440) donne # 11C1

7. Les entrées/sortie

retour

7.1. Introduction

Toutes les entrées/sorties du micro-ordinateur DAI, correspondent à une ou plusieurs cases mémoire. Les entrées/sorties sont donc directement accessibles par un programme BASIC. Toutefois, on fera attention, à l'utilisation de l'instruction POKE pour éviter tout conflit au niveau de l'interpréteur BASIC.

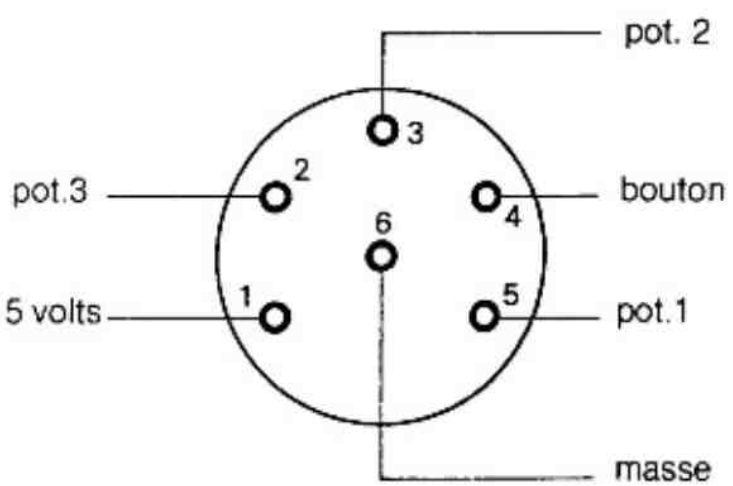
7.2. L'Interface des paddles

Le micro-ordinateur DAI est équipé d'une interface permettant de connecter deux paddles (manettes de jeu). Chaque paddle peut avoir jusqu'à trois potentiomètres et un bouton-poussoir. La position physique de chaque potentiomètre correspond à un nombre entre 0 et 255.

La position d'un potentiomètre d'un paddle est trouvée en précisant l'adresse binaire du potentiomètre concerné sur 3 bits du port # FD06 ; puis le canal 0 du générateur de son est positionné en mode compteur et la position est lue dans la case mémoire # FD01. Le résultat est codé sur 8 bits.

Schéma de la prise PADDLE du DAI

Prise DIN femelle 6 broches (5 à 240° et 1 au centre)
résistance maximum du potentiomètre : 100 KΩ



vue arrière
du DAI

7.3. L'interface cassette audio

Le micro-ordinateur DAI contient toute la logique et les circuits d'interface audio pour connecter deux minicassettes permettant la sauvegarde et la lecture de données ou de programmes.

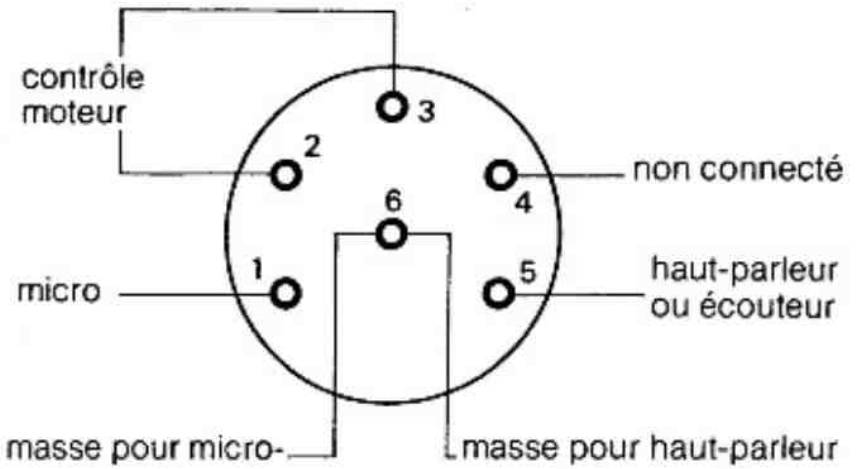
L'entrée cassette du DAI doit être reliée à la sortie «écouteur» ou haut-parleur» du magnétophone et la sortie du DAI doit être reliée à l'entrée «microphone». Si le minicassette ne possède pas de sortie «haut-parleur», brancher l'entrée du DAI en parallèle sur le haut-parleur.

Le volume du minicassette doit être réglé (sauf rare exception) au maximum.

Le réglage de tonalité doit toujours être au plus aigu.

Schéma de la prise cassette du DAI (vue arrière).

prise DIN femelle 6 (5 broches à 240° et 1 au centre)

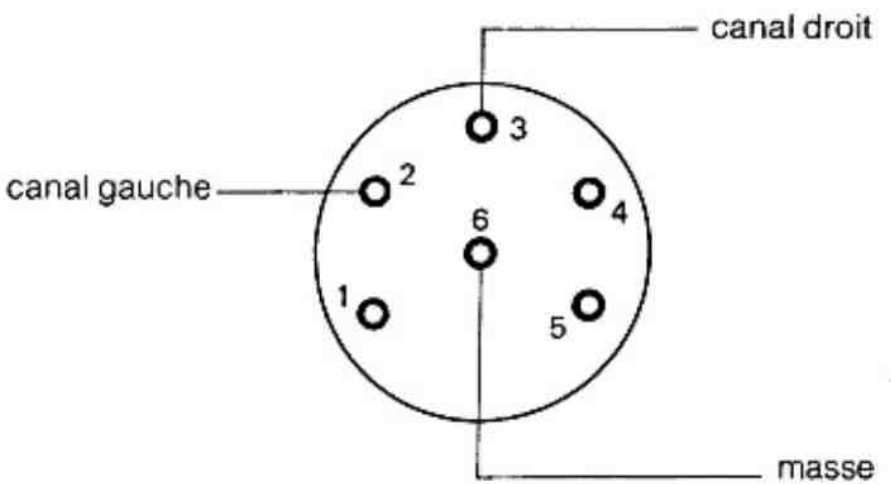


Remarque : si le DAI ne perçoit aucun programme sur une cassette, échanger les deux gros jacks.

7.4. La sortie Stéréo

Le générateur de son du micro-ordinateur DAI peut se brancher sur la prise AUX d'un amplificateur stéréophonique. Dans ce cas, le canal 0 est à droite, le canal 1 à gauche, le générateur de bruits et le canal 2 sortent sur les deux voies.

Schéma de la prise stéréo du DAI (vue arrière)



7.5. La prise péritel (ou Péritélévision)

le micro-ordinateur DAI possède une interface péritel permettant le branchement d'un moniteur couleur ou d'un poste de télévision via la prise péritel.

Schéma de la prise péritel du DAI (vue arrière)

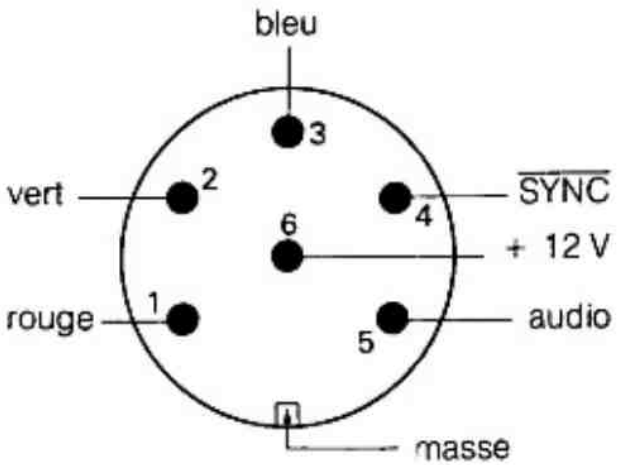
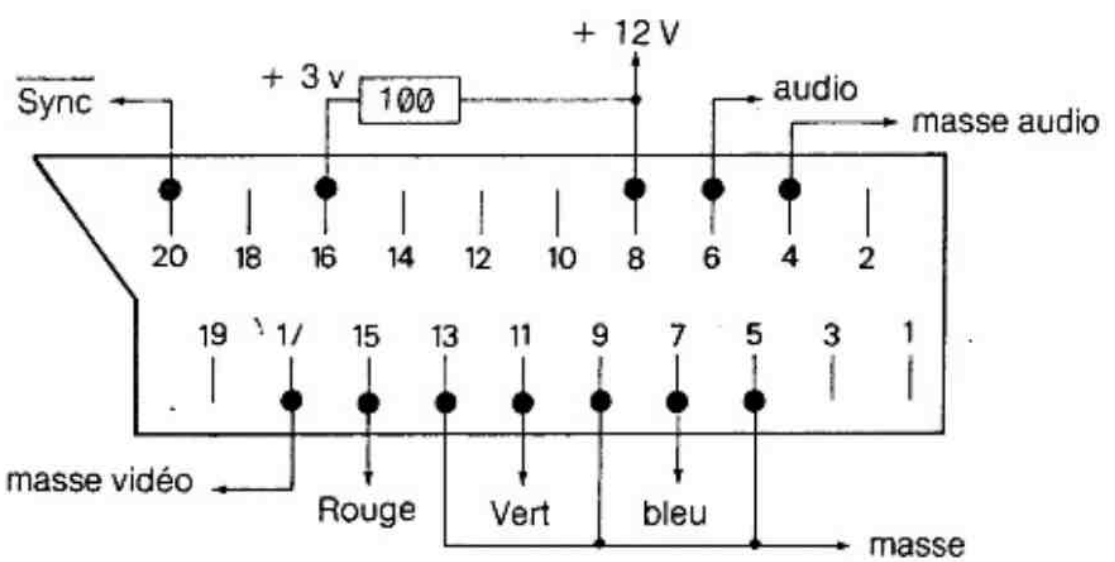


Schéma de la prise péritel, côté téléviseur



7.6. Le processeur arithmétique

En option, pour des calculs plus rapides, le DAI peut recevoir le processeur arithmétique 9511 d'ADVANCED MICRO-DEVICES.

Le coprocesseur est adressable en # FB00 (donnée) et en # FB02 (commande et status). Le signal «PAUSE» est utilisé pour synchroniser la CPU et les données.

Noter que les instructions SHLD et LHLD ne peuvent pas être utilisées avec ce processeur pour les transferts de deux octets.

Son utilisation est automatique. Dès sa mise en place sur son support, le BASIC détecte sa présence et lui confie tous les calculs qu'il exécute 10 à 15 fois plus vite.

7.7. Le clavier ASCII

Le clavier du DAI est constitué d'une matrice de contacts. Le codage, l'anti-rebond et la détection de l'enfoncement de plusieurs touches sont réalisés par un sous programme.

7.7.1. Aspect physique du clavier

A chaque touche correspond une ligne et une colonne

	0	1	2	3	4	5	6
0	0	R	re-turn	H	P	X	↑
1	1	9	A	I	Q	Y	↓
2	2	:	B	J	R	Z	←
3	3	;	C	K	S	[	→
4	4	,	D	L	T	^	Tab
5	5	-	E	M	U	space bar	ctrl
6	6	.	F	N	V	rept	break
7	7	/	G	O	W	char del	shift

lignes
Sorties (# FF07)

colonnes Entrées (# FF01)

7.7.2. Logique de balayage du clavier

Le micro-ordinateur DAI contient un sous programme de balayage et de codage du clavier. Cette routine peut être appelée par un autre programme qui utilise le même tableau de codification des touches.

Toutes les touches sont scrutées périodiquement, et le DAI détecte si une nouvelle touche est enfoncée. Si deux touches sont appuyées en même temps, les deux seront prises en compte et la touche **REPEAT** provoquera un affichage alterné des deux caractères.

La vitesse du **REPEAT** est fixe.

Chaque code de touche est inscrit dans une table. La touche **SHIFT** permet de sélectionner une des deux parties de la table. En changeant un octet, vous pouvez déclencher la scrutation de la touche **BREAK**

C'est le mode utilisé lors de l'exécution de programmes BASIC sans instruction INPUT.

A la mise sous tension, les touches alphabétiques (A-Z) sont majuscules si **SHIFT** est relâchée et minuscules si **SHIFT** est enfoncée. Le fait d'appuyer sur la touche **CTRL** inverse l'effet de la touche **SHIFT**. Vous pouvez inverser autant de fois que vous le voulez la fonction de **SHIFT**

Les codes standards renvoyés par chaque touche sont donnés dans les deux pages suivantes.

7.7.3. *Table des caractères ASCII*

Decimal	Caractere	Decimal	Caractere	Decimal	Caractere
000	NUL	027	£	054	6
001	SOH	028	©	055	7
002	STX	029	GS	056	8
003	ETX	030	RS	057	9
004	EOT	031	US	058	:
005	ENQ	032	SPACE	059	;
006	ACK	033	!	060	<
007	BEL	034	"	061	=
008	CH DEL	035	#	062	>
009	TAB	036	$	063	?
010	LF	037	%	064	@
011	VT	038	&	065	A
012	FF	039	'	066	B
013	CR	040	(	067	C
014	SO	041	)	068	D
015	SI	042	*	069	E
016	↑ CURS	043	+	070	F
017	↓ CURS	044	' ,	071	G
018	← CURS	045	-	072	H
019	→ CURS	046	.	073	I
020	Shift + ↑	047	/	074	J
021	Shift + ↓	048	0	075	K
022	Shift + ←	049	1	076	L
023	Shift + →	050	2	077	M
024	CAN	051	3	078	N
025	EM	052	4	079	O
026	SUB	053	5	080	P

Decimal	Caractère	Decimal	Caractère
081	Q	110	n
082	R	111	o
083	S	112	p
084	T	113	q
085	U	114	r
086	V	115	s
087	W	116	t
088	X	117	u
089	Y	118	v
090	Z	119	w
091	[	120	x
092	\	121	y
093	]	122	z
094	↑	123	{
095	←	124	|
096	`	125	}
097	a	126	~
098	b	127	DEL
099	c		
100	d		
101	e		
102	f		
103	g		
104	h		
105	i		
106	j		
107	k		
108	l		
109	m		

7.8. Le Bus D.C.E.

Le micro-ordinateur DAI peut se connecter vers l'extérieur grâce au Bus D.C.E. Une logique appropriée gère le bus D.C.E. exactement comme un circuit interne à la carte avec le même adressage et incluant les mêmes contrôles, tels que Reset et les lignes d'interruptions. Le bus D.C.E. peut être connecté directement à un équipement extérieur.

Des sous-programmes inclus dans le micro-ordinateur permettent de dialoguer avec les cartes professionnelles DAI compatibles DCE Bus.

Exemple de programme contrôlant une imprimante parallèle via le bus D.C.E.

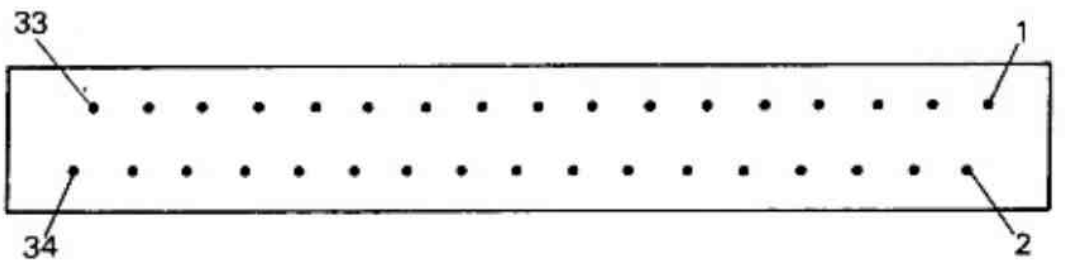
```
LIST
10    CLEAR 1000:REM ou plus, suivant votre programme
20    DIM PRI(10.0)
30    INPUT «Taper I si vous voulez l'imprimante» ; A$:PRINT
40    IF A$<>«I» GOTO 100
50    FOR X = #400 TO #419
55    READ C%
60    POKE X, C%
65    NEXT
70    POKE #FE03, #AC:REM mot de contrôle (MODE 1) pour 8255 (D.C.E.)
75    POKE #2DD, #C3
80    POKE #2DE,0
85    POKE #2DF, #4
90    DATA #E5, #D5, #C5, #11, #2, #FE, #6, #10, #21, #1, #FE
95    DATA #77, #2B, #36,0, #36, #1, #1A, #A0, #C2, #B, #4, #C1, #D1, #E1, #C9
100   PRINT CHR$(12)
110   IF A$<>«I» GOTO 200:REM suite du programme
120   IF A$ = «I» THEN POKE #131,3:REM sortie uniquement sur bus D.C.E.
```

7.8.1. Schéma de la prise DCE

Nom de la broche	description		numéro de la broche
P0B0	PORT A du 8255	bit 0	16
P0B1		bit 1	14
P0B2		bit 2	12
P0B3		bit 3	10
P0B4		bit 4	9
P0B5		bit 5	11
P0B6		bit 6	13
P0B7		bit 7	15
P1B0	PORT B du 8255	bit 0	30
P1B1		bit 1	31
P1B2		bit 2	32
P1B3		bit 3	25
P1B4		bit 4	24
P1B5		bit 5	23
P1B6		bit 6	22
P1B7		bit 7	21
P2B0	PORT C du 8255	bit 0	26
P2B1		bit 1	27
P2B2		bit 2	28
P2B3		bit 3	29
P2B4		bit 4	20
P2B5		bit 5	19
P2B6		bit 6	18
P2B7		bit 7	17

Nom de la broche	description	numéro de la broche
EX INTR	interruption extérieure	6
IN7	entrée parallèle Bit 7 (aux. interrupt.)	5
EX RESET	reset extérieur (masse pour reset)	7
+ 12 v	+ 12 V DC	2
+ 5 v	+ 5 V DC	1
− 5 v	− 5 V DC	3
INTR	PIN 14 du 8080 (interruption)	33
IN7		34
MASSE		4
N.C	non connectée	8

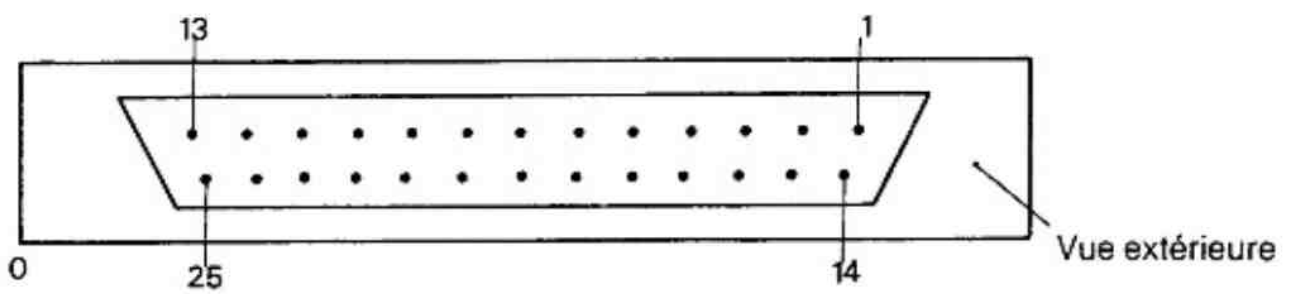
Vue extérieure du connecteur du bus DCE



7.9. L'interface RS 232 (ou Série)

Le micro-ordinateur DAI possède une interface RS 232 standard composée d'une ligne de sortie série, d'une ligne d'entrée série et d'une ligne de contrôle (DTR) permettant d'arrêter la sortie série. Ces lignes sont disponibles sur un connecteur à l'arrière de la machine. Le signal DTR est utilisé pour synchroniser la sortie des caractères vers une imprimante série. Si ce signal n'est pas utilisé, la transmission n'est pas contrôlée. Les interruptions allouées aux adresses #20 et #28 peuvent être utilisées comme «Receive Ready» (prêt à recevoir) et comme «Ready» (prêt à transmettre). L'interpréteur BASIC utilise dans certains cas, ces deux interruptions dans un autre but.

Schéma du connecteur femelle RS 232



broche	fonction	
1	masse	
2	sortie série	* + 12 V à travers une résistance de 220 Ω, 1/4 Watt
3	entrée série	
4	Data Terminal Ready (DTR)	
5	+ 12 V *	
6	+ 12 V *	
7	masse	
8	+ 12 V *	

– Les broches de 9 à 25 ne sont pas connectées.

– L'entrée DTR est à un niveau haut lorsque l'imprimante est prête à recevoir des caractères.

– Note : Ce connecteur peut servir à brancher un terminal. Dans ce cas il faudra croiser les signaux des broches 2 et 3 puisque *l'entrée* du DAI correspond à la *sortie* du terminal et la *sortie* du DAI correspond à *l'entrée* du terminal.

– POKE #296,0 attend un caractère au clavier.

– POKE #296,1 attend un caractère sur le RS 232

7.10. Les adresses des Entrées/Sorties

7.10.1. Les adresses (Hex) des principales Entrées/Sorties

F900-F9FF	libre	
FA00-FAFF	libre	
FB00/1	data	processeur arithmétique (9511)
FB02/3	Commande	
FC00/1	canal 0	générateur de son (8253)
FC02/3	canal 1	
FC04/5	canal 2	
FC06/7	commande	
FDXX	Voir 7.10.2	
FE00	port 0(A)	bus D.C.E. (8255)
FE01	port 1(B)	
FE02	port 2(C)	
FE03	commande	
FFXX	Voir 7.10.3	

7.10.2 Détail des différentes adresses

Adresses (Hex)	notes	IN/OUT (E/S)	fonction bit
FD00	1	entrée	0 – 1 – 2 signal page 3 sortie série prête 4 bouton paddle droite 5 bouton paddle gauche 6 donnée aléatoire 7 entrée cassette
	1 = fermé 1 = fermé		
FD01	3	entrée	une seule impulsion pour le timer des paddles
FD04	2	sortie	0 } volume du canal 1 1 2 3 4 } volume du canal 2 5 6 7
FD05	2	sortie	0 } volume du canal 3 1 2 3 4 } volume du noise 5 6 7
FD06	3	sortie	0 sortie cassette 0 } code de sélection du paddle 1 2 3 validation paddle 4 contrôle moteur 1 5 contrôle moteur 2 6-7 sélection ROM
	0 = marche 0 = marche		

Notes :

1 : l'utilisateur peut lire ou écrire dans ces adresses autant de fois qu'il veut. Cela ne gène en aucun cas le fonctionnement de la machine.

2 : la lecture de ces cases mémoire ne donne rien. L'écriture à ces adresses modifiera le volume approprié mais il se peut que l'interpréteur BASIC vienne aussi changer un volume.

3 : Vous ne devez pas écrire dans ces cases mémoire.

7.10.3. Entrée/sortie série, timer et contrôle des interruptions

Les informations données dans ce chapitre sont suffisantes pour comprendre l'utilisation des entrées/sorties série. Toutes ces possibilités sont données par un seul circuit 40 broches 5501 de TEXAS INSTRUMENT. L'interpréteur BASIC utilise lui aussi le maximum de ces possibilités.

Vous devez donc prendre certaines précautions pour ne pas déranger le fonctionnement normal du système.

Adresse	note	fonction
# FF00	1	buffer «entrée série» : contient le dernier caractère reçu sur le RS 232
# FF01	1	port d'entrée du clavier : les sept premiers bits viennent du clavier. Le huitième bit est IN7 (provenant du bus D.C.E.) et est lié au signal «page blanking» de la T.V.
# FF02	2	registre des adresses d'interruptions
# FF03	1	Registre de status Bit 7,6,5 non utilisés. Bit 4 : buffer de transmission vide. Mis à 1 si la sortie RS 232 est prête à accepter un autre caractère. Bit 3 : buffer de réception chargé. Mis à 1 si un caractère vient d'être reçu. Bit 2 : Entrée série : ce bit suit l'état de l'entrée du RS 232 Bit 1 : «Overrun» mis à 1 si un nouveau caractère est reçu alors qu'un caractère est toujours présent dans le buffer de réception. Bit 0 : «Frame error» : erreur de structure dans la réception d'un caractère. Le buffer «entrée série» contient #FF.
# FF04	2	Registre de commande
# FF05	3	Registre de baud rate pour RS 232 1 pour 110 baud 2 bits de stop 2 pour 150 baud 2 bits de stop 4 pour 300 baud 2 bits de stop 8 pour 1200 baud 2 bits de stop # 10 pour 2400 baud 2 bits de stop # 20 pour 4800 baud 2 bits de stop # 40 pour 9600 baud 2 bits de stop # 81 pour 110 baud 1 bit de stop # 82 pour 150 baud 1 bit de stop # 84 pour 300 baud 1 bit de stop # 88 pour 1200 baud 1 bit de stop # 90 pour 2400 baud 1 bit de stop # A0 pour 4800 baud 1 bit de stop # C0 pour 9600 baud 1 bit de stop # 00 inhibe le RS 232. Les autres combinaisons binaires ne sont pas à utiliser.
# FF06	3	Sortie série : écrire un octet dans cette adresse permet de le sortir sur le RS 232. A utiliser uniquement lorsque le bit 4 de FF03 est à 1.
# FF07	4	Port de sortie pour clavier. Sortie des données pour la scrutation du clavier (inutile pour l'utilisateur)

Adresse	note	fonction
FF08	2	Registre des masques d'interruption
FF09 FF0A FF0B FF0C FF0D	2	adresse des 5 timers

Notes :

1. peut être lue, mais ne peut pas être modifiée par l'utilisateur.
2. ne doit pas être modifiée par l'utilisateur.
3. peut être modifiée mais pas lue par l'utilisateur.
4. ne peut pas être lue et la modification de cette adresse est inutile ! Le système de scrutation du clavier viendra toujours (ou presque) réécrire sur la donnée de l'utilisateur.

8. Les facilités du système DAI

8.1. Liste des POKE les plus intéressants

retour

POKE # 2C4, #FF	force un BREAK
POKE #131, 0	sortie vers écran et RS 232
POKE #131, 1	sortie vers écran
POKE #131, 2	sortie vers buffer d'édition
POKE #131, 3	sortie vers disquettes.
POKE #135, 1	entrée via le clavier (surprise !! à éviter si vous avez un long programme en mémoire)
POKE #135, 2	transfert du buffer d'édition vers la zone mémoire
POKE #40, 28	moteur cassette 1 en marche
POKE #40, 18	moteur cassette 2 en marche
POKE #40, 30	arrêt des moteurs cassettes 1 et 2
POKE #13D, #10	cassette 1 sélectionné
POKE #13D, #20	cassette 2 sélectionné.
POKE #74, 0	change le mode du curseur ; l'adresse #75, dans ce cas, contient l'octet de couleur du curseur bit à 1 → 2e combinaison, bit à 0 → 1ère combinaison
POKE #75, X	change le caractère symbolisant le curseur si PEEK (#74) est différent de zéro. X est un code du générateur de caractères.
les adresses #72, #73	contiennent l'adresse mémoire du caractère pointé par le curseur (#73 contient l'adresse haute)
Les adresses #78, #79	contiennent l'adresse mémoire du contrôle de mode de la ligne où se trouve le curseur (#79 contient l'adresse haute).
L'adresse #76	contient la valeur de l'octet de couleur à assigner au premier caractère d'un PRINT

Exemple :

```
*LIST
5   COLORT 0 13 0 13
10  INPUT A$:PRINT
20  FOR I = 0.0 TO LEN(A$-1.0:POKE #76, #FF:PRINT
MID$(A$,I,1);:NEXT
30  COLORT 0 13 0 13:WAIT TIME 25
40  COLORT 0 13 0 3:WAIT TIME 25:GOTO 30
*
```

8.2. Comment détecter la présence du processeur arithmétique (9511)

Entrez la commande :

PRINT PEEK (# FB00)

Si la réponse est 255 (ou autre) vous n'avez pas de 9511
si la réponse est «DIVISION BY ZERO» vous avez un 9511.

8.3. Que faire si un Reset accidentel survient pendant l'exécution d'un programme

En prenant une simple précaution avant l'exécution du programme BASIC, le programme encore résidant en mémoire peut être restitué après un Reset accidentel. Avant l'exécution d'un programme, vous devez :

1) taper BREAK

2) UT ↵

3) D29F ␣ 2A4 ↵ noter les 6 octets a b c d e f

4) D ba ␣ ba + 1 ↵ noter les deux octets X-Y

5) B

Après un Reset accidentel :

1) taper UT ↵

2) S29F ↵
et changer éventuellement les octets différents des six octets notés.

3) Sba ↵
et changer éventuellement les octets différents des deux octets notés.

4) B

5) taper EDIT ↵ puis BREAK SPACE.

8.4. Sauvegarde et chargement d'une image

Après avoir dessiné une image, il faut, pour la sauver sur cassette :

1) Appuyer sur BREAK

2) UT ↵

3) Waaaa ␣ BFFF ␣ IMAGE ↵

Pour charger l'image

1) MODE ?

2) UT ↵

3) R ↵

MODE	Adresse aaaa
0	B350
1,2	B9C8
1A,2A	B7A0
3,4	A884
3A,4A	A65C
5,6	65E0
5A,6A	63B8

8.5. Réservation mémoire en-dessous de la zone de stockage variable

A l'initialisation, une place mémoire peut être réservée en-dessous de la zone de stockage de la manière suivante :

```
CLEAR 1000
DIM DUMMY (62) ..... utilise 256 octets
POKE #29C, PEEK (#29C) + 1 ..... déplace le début de stockage + 256
POKE # 29E, PEEK (# 29E) – 1 ..... dimension du stockage – 256
CLEAR 4 ..... détruit la zone stockage (DUMMY inclus)
```

8.6. L'utilisation du timer externe

La fonction WAIT TIME est commode mais son utilisation pour des timings n'est pas possible à cause des interruptions.

Une solution pour les timings est d'utiliser le timer extérieur.

POKE #1BE, TIME OUT démarre le timer, la période étant TIME OUT X 20 ms.

L'instruction WAIT MEM # 1BE, #FF, #FF suspendra l'exécution du programme jusqu'à ce que la période demandée soit écoulée.

8.7. Utilisation des Paddles

manipulateur connecté sur «entrée PADDLE 1»

- pour la lecture de X, taper : PRINT PDL (0)
- pour la lecture de Y, taper : PRINT PDL (1)
- pour la lecture de Z, taper : PRINT PDL (2)
- pour l'utilisation du bouton, mettre dans le programme :

IF PEEK (#FD00) IAND #20 <> 0 THEN (bouton appuyé)

manipulateur connecté sur «entrée PADDLE 2»

- pour la lecture de X, taper : PRINT PDL (3)
- pour la lecture de Y, taper : PRINT PDL (5)
- pour la lecture de Z, taper : PRINT PDL (4)
- pour l'utilisation du bouton, mettre dans le programme

IF PEEK (#FD00) IAND #10 <> 0 THEN (bouton appuyé)

8.8. Adresses des octets de contrôle de chaque ligne en mode texte

X curseur	n° de ligne	CONTROLE	COULEUR	
23	1	BFEF	BFEE	MODE 0
22	2	BF69	BF68	
21	3	BEE3	BEE2	
20	4	BE5D	BE5C	
19	5	BDD7	BDD6	
18	6	BD51	BD50	
17	7	BCCB	BCCA	
16	8	BC45	BC44	
15	9	BBBF	BBBE	
14	10	BB39	BB38	
13	11	BAB3	BAB2	
12	12	BA2D	BA2C	
11	13	B9A7	B9A6	
10	14	B921	B920	
9	15	B89B	B89A	
8	16	B815	B814	
7	17	B78F	B78E	
6	18	B709	B708	
5	19	B683	B682	
4	20	B5FD	B5FC	
3	21	B577	B576	
2	22	B4F1	B4F0	
1	23	B46B	B46A	
0	24	B3E5	B3E4	

3	21	BAE7	BAE6	MODE 1A/2A
2	22	BA61	BA60	
1	23	B9DB	B9DA	
0	24	B955	B954	
3	21	ACD3	ACD2	MODE 3A/4A
2	22	AC4D	AC4C	
1	23	ABC7	ABC6	
0	24	AB41	AB40	
3	21	7557	7556	MODE 5A/6A
2	22	74D1	74D0	
1	23	744B	744A	
0	24	73C5	73C4	

8.9. Routines en assembleur

Affichage d'un caractère sur écran : 2 méthodes

Mettre dans A le code ASCII du caractère n

```
1/ MVI  A,:n      2/ MVI  A,:n
   RST  5             CALL :D695
   DATA :03
```

Attente d'un caractère entré par le clavier :

```
Exp : POSE   CALL   :D6BB
             ORA    A
             JZ     POSE
```

La frappe d'un caractère fera sortir de la boucle et le code ASCII du caractère sera dans A.

Fonction MODE :

Mettre dans A le numéro du mode choisi :

```
FF              =>   MODE 0
0,2,4,6,8,A     =>   MODE 1,2,3,4,5,6
1,3,5,7,9,B     =>   MODE 1A,2A,3A,4A,5A,6A
```

```
MVI    A,:N
RST    5
DATA   :18
```

Fonction CURSOR :

Exp : CURSOR 20,12

```
MVI    H,12      (Valeur de Y dans H)
MVI    L,20      (Valeur de X dans L)
RST    5
DATA   :09
```

Fonction DOT :

```
MVI    H,:ah     (Poids fort de      X → H)
MVI    L,:al     (Poids faible de    X → L)
MVI    C,:n            (Valeur de   Y → C)
MVI    A,:c            (Couleur     c → A)
RST    5
DATA   :1E
```

Fonction DRAW :

```
MVI     D,:ah       (Poids fort de      X → D )
MVI     E,:al       (poids faible de    X → E ) > Extrêmité 1
MVI     B,:n              (Valeur de    Y → B )
MVI     H,:ah       (Poids fort de      X → H )
MVI     L,:al       (Poids faible de    X → L ) > Extrêmité 2
MVI     C,:n              (Valeur de    Y → C )
MVI     A,:c              (Couleur      c → A )
RST     5
DATA    :21
```

Fonction FILL :

```
MVI     D,:ah       (Poids fort de      X → D )
MVI     E,:al       (Poids faible de    X → E ) > Coin 1
MVI     B,:n              (Valeur de    Y → B )
MVI     H,:ah       (Poids fort de      X → H )
MVI     L,:al       (Poids faible de    X → L ) > Coin 2
MVI     C,:n              (Valeur de    Y → C )
MVI     A,:c              (Couleur      c → A )
RST     5
DATA    :24
```

Fonction COLORT :

```
VECTOR  DATA x,x,x,x    ( x,x,x,x Couleurs choisies)
RESERV  ***             ( Mise 00 à l'adresse RESERV )
        PUSH H
        PUSH PSW
        MVI  H,:04
        MVI  L,:00
        LDA  :FD06
        STA  RESERV
        MVI  A,:BF
        STA  :FD06
        CALL :E006
        LDA  RESERV
        STA  :FD06
        POP  PSW
        POP  H
```

Fonction COLORG :

```
VECTOR  DATA x,x,x,x    ( x,x,x,x Couleurs choisies )
RESERV  ***             ( Met 00 à l'adresse RESERV )
        PUSH H
        PUSH PSW
        MVI  H,:04
        MVI  L,:00
        LDA  :FD06
        STA  RESERV
        MVI  A,:BF
        STA  :FD06
        CALL :E018
        LDA  RESERV
        STA  :FD06
        POP  PSW
        POP  H
```

Fonction SCRN :

```
MVI   H,:ah     (Poids fort d X → H )
MVI   L,:al     (Poids faible de X → L )
MVI   C,:n            (Valeur de Y → C )
RST   5
DATA  :27
```

A contient la couleur du point demande
B contient YMAX
D contient Poids fort de XMAX
E contient Poids faible de XMAX

8.10. adresses des commandes du moniteur

B : C7A0 BASIC	CALL ou JMP C7A0 retour sous BASIC CD ou C3,A0 C7
D : EAB3 DISPLAY	CALL EAB3 : demande Ladr et Hadr CD,B3,EA
F : ED48 FILL	CALL ED48 : demande Ladr et Hadr CD.48.ED
G : ED8A GO	CALL ED8A : demande l'adresse de départ CD 8A ED
L : EB26 LOOK	CALL EB26 : demande les 3 adresses CD 26 EB
M : EC83 MOVE	CALL EC83 : demande les 2 adresses et une donnée CD 83 EC
R : EF0F READ	CALL EF0F : demande le nom (ou Return) CD 0F EF
S : ED5C SUBTITUTE	CALL ED.5C : demande l'adresse CD.5C.ED
V : ED77 Vecteur	CALL ED77 : demande le nom du vecteur CD 77 ED
W : EEE4 Write	CALL EEE4 : demande les deux adresses + le nom CD E4EE
X : ED6E EXAMINE	CALL ED6E : demande le registre CD 6EED
Z : ECBA INITIALIZE	CALL ECBA : demande un chiffre CD BA EC.

ces adresses sont à utiliser sous assembleur ou moniteur. Un CALLM à une de ces adresses, provoque la perte du contrôle du système.

En fait, un appel à ces sous-programmes remplace le caractère alphanumérique définissant la fonction du moniteur.

8.11. Les huit interruptions du DAI

8.11.1. Introduction

Le microprocesseur 8080 connaît huit codes-instructions réalisant un CALL sur un octet, ces instructions sont RST 0 à RST 7. Dans beaucoup de systèmes ces instructions sont utilisées en combinaison avec les interruptions. Ce chapitre décrit la manière dont le DAI utilise ces interruptions.

8.11.2. Les restart 1,4 et 5

Ces trois «restart» sont utilisés pour sélectionner une des quatre ROM. Dans la memory-map du DAI, de l'adresse E000 à EFFF, sont connectées «en parallèle» quatre ROM de quatre K octets. Une de ces ROM peut être sélectionnée via les bits 6 et 7 du port FD06. Généralement, c'est la ROM 0 qui est sélectionnée mais, grâce au programme, une des trois autres ROM peut être activée à son tour. Ce sont les instructions RST1, RST4 et RST5 qui réalisent ce décodage par progamme ; ces instructions sont suivies d'un octet de donnée.

Lorsque le compteur de programme rencontre une de ces instructions RST, il exécute une des 8 routines d'interruption définies en mémoire de l'adresse 000 à 003F. Une des adresses «renvoi d'interruptions» est chargée dans le compteur de programme et le programme continue à partir de cette adresse.

Ces routines, dont les adresses sont chargées grâce aux «renvois d'interruptions», préparent la sélection de la ROM désirée.

RST 1 : sélection ROM 3 (moniteur, son)
RST 4 : sélection ROM 1 (fonctions mathématiques)
RST 5 : sélection ROM 2 (fonctions écran).

Grâce à la routine de l'adresse C6CF, la sélection des ROM est activée.

L'octet de donnée qui suit l'instruction RST, indique l'adresse de la ROM choisie pour continuer l'exécution du programme. Cette adresse est égale à l'octet de donnée, plus E000. Exemple :

RST 5, data 18: ROM 2 sélectionnée ; saut à l'adresse E018.

Ce «Restart» provoque un branchement à la routine de changement de MODE. Lorsque la routine appelée est terminée, la sélection de ROM se repositionne sur la ROM précédémment sélectionnée.

8.11.3. Les autres instructions «restart»

Tous les autres «Restart» sont utilisés par les interruptions. Celles-ci sont générées par les timers du circuit intégré 5501 (timers et contrôleurs d'interruptions).

8.11.3.1. RST7 interruption d'horloge

Le signal de 20 ms «page blanking» de la gestion d'écran, utilise cette horloge. A chaque «front d'horloge», le compteur de programme est chargé avec 0038. Grâce aux «renvois d'interruptions», le programme se branche ensuite à l'adresse D9A9.

A chaque fois que l'interruption RST7 est présente, un compteur en 01BE/F (16 bits) est décrémenté. Si celui-ci n'est pas égal à zéro, il ne se passe rien. Lorsque ce compteur est égal à zéro, une routine RST5, data 12 est activée. Cette routine provoque le clignotement du curseur en fonction des paramètres stockés en mémoire, de l'adresse 0074 à l'adresse 0077.

Après avoir changé le contenu de l'écran pointé par le curseur, l'ancien «masque d'interruptions» est restitué et le programme retourne à la séquence interrompue.

8.11.3.2. RST6 interruption pour le clavier

A chaque fois qu'une interruption provenant du timer 4 est présente, le compteur de programme est chargé avec la valeur D578 grâce à la routine d'interruptions de l'adresse 0030.

Le «compteur pour le clavier» en 01C1 est décrémenté à chaque interruption RST6. Lorsque ce compteur est égal à zéro, la routine de scrutation du clavier est déclenchée (routine GETC).

Après avoir changé le contenu de l'écran pointé par le curseur, l'ancien «masque d'interruptions» est restitué et le programme retourne à la séquence interrompue.

8.11.3.3. RST3 interruption pour le son

A chaque fois qu'une intervention provenant du timer 3 est présente, le compteur de programme est chargé avec la valeur D755, grâce à la routine d'interruptions de l'adresse 0018.

Cette routine de RST3 autorise seulement les interruptions d'horloge et de son. Le timer 3 est rechargé et la ROM 1 est sélectionnée.

Maintenant le programme continue aux adresses EE6E de la ROM 1 où se trouve la «gestion du son». Ce sous-programme exécuté, la ROM précédémment sélectionnée l'est de nouveau, le «masque d'interruptions» est restitué et le programme retourne à la séquence interrompue.

8.11.3.4. RST2 interruption pour le «stack»

Lorsqu'un «STACK OVERFLOW» intervient, une interruption liée au RST2 est déclenchée. Le compteur de programme est chargé avec la valeur D9E2 grâce à la routine d'interruptions de l'adresse 0010. La routine en D9E2 réinitialise le pointeur de pile en F900. Les timers pour le son et le clavier sont réinitialisés. Avant de restituer le contexte, ce programme affiche «STACK OVERFLOW».

8.11.3.5. RST0 interruption pour le moniteur

Le Restart 0 est utilisé seulement par le programme moniteur. Le compteur de programme est chargé avec la valeur EB5D (ROM 3) grâce à la routine d'interruptions de l'adresse 0000, cette interruption n'est autorisée qu'après une commande Z2 ou Z3.

Cette interruption est reliée au timer 0 et est utilisée pour la fonction LOOK du moniteur.

Après cette interruption, tous les registres du 8080 sont préservés en RAM aux adresses 0053-005E. Puis le programme continue à l'adresse donnée par la routine LOOK et indique l'adresse de la prochaine instruction à exécuter. Ceci permet de détecter si cette adresse est incluse dans la «fenêtre» de visualisation du LOOK. Lorsque le résultat est positif, le contenu de tous les registres, y compris le pointeur de programme, les flags, et le pointeur de pile, est visualisé sur l'écran. Ceci terminé, le timer 0 est rechargé, le «masque d'interruptions» et le registre du 8080 sont restitués ; à la prochaine interruption RST0, l'instruction-machine sous LOOK qui suit, sera exécutée.

retour

Edité par **MULTISOFT**

25, RUE BARGUE, 75015 PARIS. TÉL. : 783.88.37