ORIC-1™

MANUEL DE PROGRAMMATION BASIC

John SCRIVEN

ORIC-1

MANUEL DE PROGRAMMATION BASIC

Traduction et adaptation française : LUCIEN AUGUSTONI

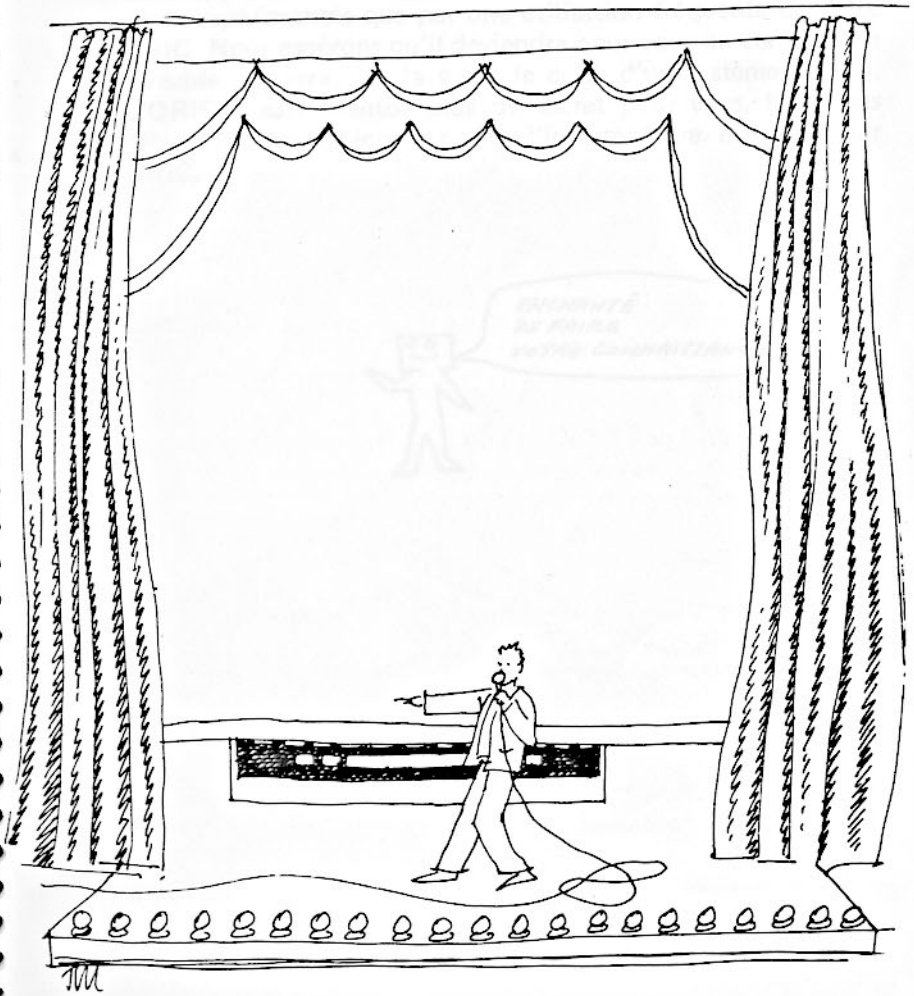
ASN DIFFUSION

MANUEL PROGRAMMATION BASIC
Enregistré sous le nº 6317

Propriété de ASN Diffusion
Z.I. La Haie Griselle - 94470 BOISSY-ST-LÉGER

TABLES DES MATIERES

CHAPITRE 1
Introduction



1. INTRODUCTION

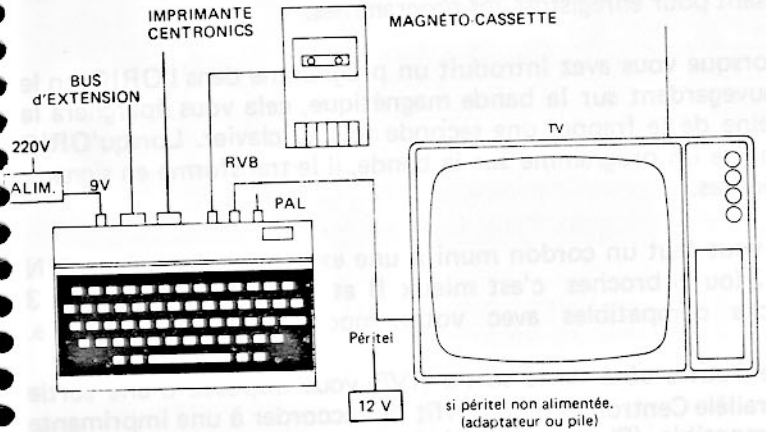
Félicitations ! Vous voilà en possession de l'un des micro-ordinateurs les plus évolués de ceux disponibles aujourd'hui. Ce livre est indispensable à ceux qui n'ont pas encore utilisé de micro-ordinateur. Il sera aussi bien utile à ceux qui ont acquis quelque pratique sur d'autres appareils, car ORIC a de nombreuses possibilités qui le rendent plus puissant que d'autres micro-ordinateurs. Si vous êtes initiés vous pouvez sauter le premier chapitre.

Vous apprendrez beaucoup à la lecture du livre, mais vous ne serez expérimentés que par une utilisation fréquente de votre ORIC. Nous espérons qu'il deviendra pour vous un compagnon agréable qui sera, par la suite, le cœur d'un système étendu. L'ORIC n'aura bientôt plus de secret pour vous. Même les débutants vont découvrir que l'informatique est aisée sur ORIC.

CHAPITRE 2
Mise en Œuvre

2 MISE EN OEUVRE

FIG 2



Une fois déballé, votre ORIC posé devant vous se présente comme un clavier avec à l'arrière diverses prises.

La plus à gauche recevra l'alimentation en courant électrique 9 volts continu délivré par l'adaptateur fourni.

Le câble de liaison ORIC-Téléviseur (ou ORIC-Moniteur couleur) est muni d'une fiche DIN côté ORIC, d'une prise PÉRITEL pour l'écran, avec adapteur 12V.

Si votre téléviseur couleur n'a pas la prise Péritel, renseignez-vous pour savoir comment l'adapter.

Un téléviseur noir et blanc recevant les signaux de l'ORIC donnera des gris variés.

L'ORIC étant alimenté, allumer votre écran :

Une image se forme, le mot *Ready* apparaît : il signifie que le micro-ordinateur est *Prêt* et attend vos instructions : vous les entrerez par le clavier.

A côté de la sortie RVB (en anglais RGB), la sortie son (aux normes D.I.N.) vous permettra soit de brancher un magnétophone à cassette, soit un amplificateur et des haut-parleurs.

Un magnéto-cassette portatif ordinaire est largement suffisant pour enregistrer vos programmes.

Lorsque vous avez introduit un programme dans l'ORIC, en le sauvegardant sur la bande magnétique, cela vous épargnera la peine de le frapper une seconde fois au clavier. Lorsqu'ORIC envoie un programme sur la bande, il le transforme en signaux sonores.

Il vous faut un cordon muni à une extrémité d'une fiche DIN à 3 (ou 5 broches c'est mieux !) et à l'autre extrémité de 3 jacks compatibles avec votre modèle de magnétophone.

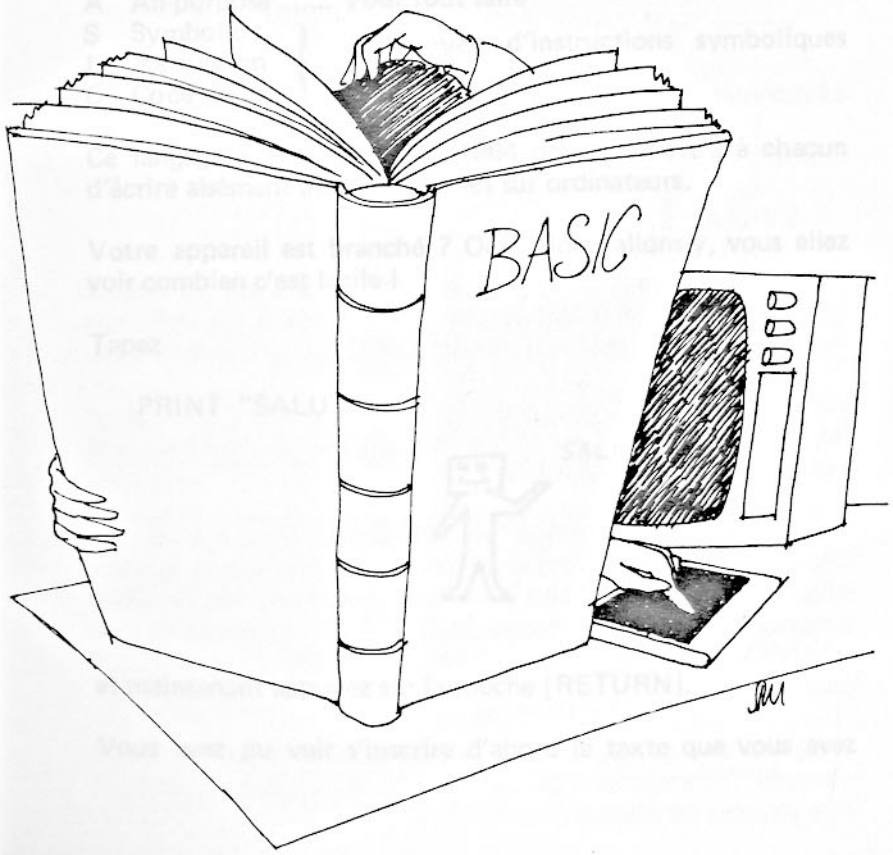
De l'autre côté de la sortie RVB vous disposez d'une sortie parallèle Centronics qu'il suffit de raccorder à une imprimante compatible. (Plusieurs modèles existent sur le marché).

Enfin la large prise multibroche voisine autorise le branchement à de nombreuses extensions : mémoires supplémentaires, cartouches de jeux, poignées de jeux, et bien sûr un MODEM. Avec ce système vous serez en relation avec des banques de données, d'autres possesseurs d'ORIC, etc ...

A la partie inférieure de l'ORIC, accessible à travers un trou, un bouton, qui nécessitera un crayon pour être poussé, vous sera fort utile. Il est appelé RESET et ne sert qu'en dernier recours lorsque l'ordinateur est entré dans une "boucle folle". Aucun des moyens usuels indiqués ailleurs ne permet de commander l'ORIC.

Ce bouton RESET n'éteint pas l'appareil, il arrête seulement l'exécution du programme. On peut le considérer comme un "démarrage à chaud" du fait qu'il ne détruit pas le contenu de la mémoire : en clair, le programme n'est pas perdu.

CHAPITRE 3
Programmer en Basic



3. PROGRAMMER EN BASIC

D'abord les mauvaises nouvelles — ORIC ne comprend pas le français. Ensuite les bonnes nouvelles — vous n'avez pas à apprendre des langages compliqués, car ORIC utilise le BASIC :

B Beginners ..... Pour débutants
A All-purpose ..... Pour tout faire
S Symbolic
I Instruction
C Code
} au moyen d'instructions symboliques codées

Ce langage a été inventé en 1964 pour permettre à chacun d'écrire aisément des programmes sur ordinateurs.

Votre appareil est branché ? Oui. Alors, allons-y, vous allez voir combien c'est facile !

Tapez

PRINT "SALUT"

et maintenant appuyez sur la touche [RETURN].

Vous avez pu voir s'inscrire d'abord le texte que vous avez

tapé, puis le mot SALUT s'inscrire juste en-dessous après la pression sur [RETURN].

Le carré clignotant est appelé curseur. Il est à l'endroit où le prochain caractère frappé au clavier va s'afficher. PRINT est un mot BASIC (une commande), il signifie imprime, écris, affiche.

Utilisez plusieurs fois cette *commande*, cette *instruction* pour obtenir l'écriture à l'écran de quelques mots de votre choix. N'oubliez pas les guillemets avant et après le message, et souvenez-vous qu'il faut presser [RETURN] pour obtenir l'exécution de l'ordre qu'on vient d'écrire.

Maintenant essayez

```
ECRIS "SALUT"
```

et appuyez sur [RETURN]

Ouille ! Vous recevez un message d'erreur ! Les mots

```
SYNTAX ERROR
```

signifient que vous avez fait une faute. BASIC est un langage facile à apprendre, il faut utiliser les mots exacts qui sont seuls reconnus. A la moindre erreur, ORIC ne comprend plus.

Les mots du BASIC sont proches du sens qu'ils ont en Anglais; apprendre le BASIC est légèrement plus facile pour ceux qui connaissent un peu l'Anglais.

Un coup d'œil à l'intérieur de l'ORIC peut aider à y voir un peu plus clair. ORIC contient plusieurs composants électroniques miniaturisés. Le plus important : l'unité centrale, le microprocesseur, qui est le cerveau de l'ORIC. Ces composants sont parcourus par des courants dont la tension peut être soit haute, soit basse.

Imaginez une rangée de huit petites lampes, et au-dessous de chacune l'interrupteur qui la commande. Chaque lampe peut être allumée ou éteinte.

Vous pouvez vous représenter les composants comme des séries, en grandes quantités, de groupes de huit interrupteurs. Chaque interrupteur est appelé un bit, et chaque bloc de 8 un octet.

En y réfléchissant un peu, vous verrez qu'il y a 256 combinaisons possibles en les manipulant.

Si l'on décide de représenter par 1 un interrupteur qui allume sa lampe et par 0 un interrupteur qui coupe le courant. On arrive à la représentation suivante des nombres de 0 à 255. C'est la numérotation binaire.

Numérotation binaire (base deux)		Numérotation décimale (base dix)
00000000	=	0
00000001	=	1
00000010	=	2
00000011	=	3
00000100	=	4
etc		
11111110	=	254
11111111	=	255

Voilà pourquoi certaines personnes pensent que les ordinateurs sont seulement destinés à des applications mathématiques. En fait, on peut utiliser ce codage par 0 et 1 pour des lettres, des mots et presque n'importe quoi. Il y a une similitude avec le code morse qui permet d'exprimer n'importe quel message en utilisant seulement des points et des traits.

Un autre composant important de l'ORIC est la ROM BASIC (ROM = Read Only Memory, une mémoire que l'on peut seulement lire).

Cette ROM traduit les mots du BASIC en zéros et uns que le cerveau de l'ORIC peut comprendre. Il n'y a pas tellement de mots en BASIC, entre cent et deux cents environ.

Il existe diverses versions du langage BASIC, comme il y a plusieurs versions de l'Anglais.

Par exemple, l'Anglais parlé à Londres n'est pas tout à fait le même que celui parlé à New-York. Si un Londonien parle du *trottoir* il dit "pavement", tandis que l'Américain dit "sidewalk". Voilà pourquoi vous devez surveiller le langage que vous utiliserez sur l'ORIC, afin d'éviter des messages vous signalant votre erreur.

Tapez

```
PRINT "5"
```

puis [RETURN], puis tapez

```
PRINT 5
```

puis [RETURN]. Apparemment, il n'y a pas de différence.

Maintenant tapez

```
PRINT "5 + 2"
```

puis tapez

```
PRINT 5 + 2
```

(a partir de maintenant vous êtes habitués à l'utilisation de [RETURN], aussi je cesse de vous le rappeler).

Si vous entrez les informations entre guillemets, on appelle cela une chaîne de caractères ; les chaînes peuvent contenir des lettres, des chiffres, des signes divers et même des caractères graphiques !

Si vous entrez les informations sans guillemets, alors l'ORIC reconnaît les nombres, le signe de l'opération, et comme c'était une addition, il vous a calculé la somme.

Essayez

```
PRINT 75 + 25
```

Vous devez obtenir 100.

En abrégé, vous pouvez utiliser ? à la place de PRINT pour aller plus vite. ORIC rétablira PRINT dans les listes.

(Notez bien la différence entre un zéro 0 et la lettre majuscule O, et entre le chiffre 1 et la lettre majuscule I : vous ne confondriez pas, mais l'ORIC a besoin d'être informé avec rigueur).

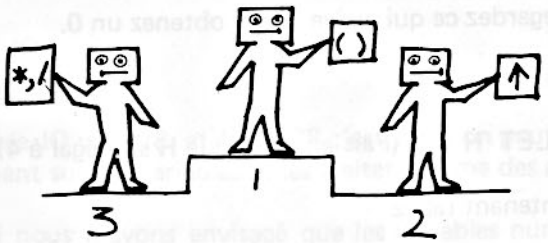
Essayez d'autres calculs. Le trait de soustraction est près de la touche 0. La division utilise la barre oblique / et multiplié c'est * (tenir une touche SHIFT appuyée et enfoncer simultanément la touche 8). Remarquez que le zéro est barré.

En cas de fausse manœuvre, appuyer sur CTRL et sur X. Une barre oblique inversée \ apparaîtra et toute la ligne sera effacée.

Si vous avez envie de vous livrer à des calculs plus compliqués, sachez que l'ORIC n'utilise pas les nombres dans l'ordre où ils sont écrits, de gauche à droite.

PRINT 4 + 3 * 2 ne vous donne pas 14, mais 10 car * a priorité sur +. Voici l'ordre des priorités, en haut les plus fortes :

( )
↑ (symbole d'élévation à une puissance)
*, /
+, –



Les signes écrits sur une même ligne ont même priorité. C'est ainsi qu'Oric calcule.

Essayer avec les exemples suivants pour vous familiariser avec ces priorités :

```
PRINT 2 * 3 * 4
PRINT 4 + 3 * 2
```

```
PRINT 4 / 2 + 3
PRINT 2 + 3/4
PRINT 3 + 4↑2
PRINT 3 – 4↑2
PRINT 2 + 4↑3 * 2
```

Si vous hésitez, mettez des parenthèses autour du calcul que vous voulez voir effectuer d'abord :

$4 + 3 * 2 = 10$ mais $(4 + 3) * 2 = 14$

Si les parenthèses sont superflues, ORIC cependant ne les refuse pas :

$4 + 5 * 3 = 19$ et $4 + (5 * 3) = 19$

ORIC considère comme nombre tout ce qui n'est pas entre guillemets.

Tapez

```
PRINT H
```

et regardez ce qui arrive : vous obtenez un 0.

Tapez

```
LET H = 4  (Fais en sorte que H soit égal à 4)
```

Maintenant tapez

```
PRINT H
```

Cette fois ORIC répond 4. Il a obéi à l'instruction précédente : on l'appelle *affectation*. C'est comme en algèbre.H est appelée une variable. ORIC se rappellera que H vaut 4, tant qu'on ne lui aura pas indiqué d'en changer la valeur, ou tapé CLEAR, ou éteint l'ordinateur, ou changé de programme.

Faites de nouveaux essais avec d'autres lettres, puis tapez CLEAR et voyez si ORIC a remis à zéro les variables. Vous pouvez utiliser des variables de plus d'une lettre, ainsi AB peut prendre une valeur différente de celles de A et de B. Vous pouvez aussi choisir A5 ou A6. ORIC accepte des variables de plus de 2 caractères, mais ne reconnaîtra que les deux premiers. Notez que le premier caractère doit être une lettre.

Essayez

```
LET JOUR = 36
LET JOIE  = 28
```

Maintenant

```
? JOUR
?JOIE
```

La réponse est 28 à chaque fois, cela provient du fait que seules les deux premières lettres JO sont testées par ORIC : ainsi en écrivant JOIE = 28, la valeur précédente de la variable est "écrasée", remplacée.

Entrez

```
? JO
? 4*JO
```

La variable JO vaut 28, et 4 fois 28 c'est 112 ; on peut calculer directement sur les variables et les traiter comme des nombres.

Jusqu'ici nous n'avons envisagé que les variables numériques.

Comment ORIC peut-il mémoriser les mots, les phrases ? Eh bien ! pratiquement de la même manière que si c'étaient des nombres ; il traite les chaînes et les variables chaînes.

Tapez

```
LET N$ = "PAUL"
```

Puis

```
?N$
```

ORIC écrit PAUL, qui est le contenu de la variable N $. La terminaison $ caractérise les variables chaînes.

Faites d'autres essais avec PF$ pour désigner votre prénom favori.

Enregistrez PF$, puis demandez :

```
PRINT N$, PF$
```

Vous obtenez les deux prénoms à l'écran. La virgule décale l'écriture du second prénom. En général, elle introduit un espace entre les chaînes. Cela ressemble à la tabulation sur une machine à écrire. ORIC utilise des taquets électroniques qui décalent de 5 caractères à gauche. Si vous utilisez le point-virgule (;) cela supprime l'espace entre les chaînes.

Exemple :

```
PRINT N$ ; PF$
```

Il est possible d'utiliser l'addition pour les chaînes : on obtient une nouvelle chaîne.

Essayez

```
LET A$ = "BON"
LET B$ = "JOUR"
LET C$ = A$ + A$
```

puis

```
PRINT A$, B$, C$
```

On dit qu'on a concaténé les chaînes A$ et B$. C'est la concaténation. Utilisez CLEAR puis demandez à nouveau

```
? A$, B$, C$
```


L'instruction CLEAR a vidé les variables chaînes, plus rien ne s'inscrit à l'écran.

Nota : Pour affecter une valeur à une variable le mot LET est facultatif. Ainsi A = 10 est équivalent à LET A = 10, de même pour les chaînes.

Quand nous avons étudié les variables numériques vous avez probablement remarqué que les fractions étaient acceptées de la même manière que les nombres entiers.

Essayons : ou en abrégé :

```
LET X = 1/3          X = 1/3
PRINT X              ? X
```

Nous obtenons

```
.333333333
```

Notez le point de décimalisation : notation anglo-saxonne. C'est une valeur approchée de 1/3, ici avec 9 chiffres décimaux.

(ORIC peut manipuler les nombres depuis 2.93874×10^{-39} jusqu'à 1.70141×10^{38}). Pour en savoir plus, voir le chapitre 6.

Les variables désignées par de simples lettres, ou par deux lettres, sont appelées variables à virgule flottante.

Si l'on ajoute la terminaison %, exemple A% = 4762 on obtient une variable entière, entre — 32768 et + 32767. En général la manipulation des variables entières est plus rapide que celle des variables à virgule flottante.

A noter que A%=3.82 suivi de ?A% donne 3, et que B%=—3.17 suivi de ?B% donne — 4. ORIC ne refuse pas l'affectation à une variable entière d'un nombre non décimal, seulement il transforme ce nombre en sa partie entière : le plus grand entier immédiatement inférieur.

Jusqu'ici, nous n'avons utilisé l'ORIC qu'à la manière d'une calculette, en mode de calcul immédiat. On utilise plutôt les ordinateurs en mode programme. On introduit une série d'instructions numérotées et on en demande l'exécution quand on veut, par la suite.

On peut numéroter les lignes de 0 à 63 999.

Il est usuel et commode de numéroter de 10 en 10. Ainsi, en cas de besoin, d'autres instructions peuvent être intercalées. ORIC remet les instructions dans l'ordre de leur numérotation, indépendamment de l'ordre de leur introduction en mémoire.

Essayez ce court programme :

```
10 CLS
20 PRINT "INDIQUEZ VOTRE NOM"
30 INPUT N$
40 PRINT "ENCHANTÉ DE FAIRE VOTRE CONNAISSANCE, " ; N
```

ORIC lira l'instruction n° 10 et nettoiera l'écran, écrira le message entre guillemets indiqué ligne 20, puis exécutera l'instruction en ligne 30. Celle-ci, INPUT N$ provoque l'affichage d'un point d'interrogation et une attente de l'ordinateur. Vous tapez alors votre nom, suivi d'un [RETURN]. La variable N$ contient maintenant votre nom. ORIC passe alors à la ligne 40 et écrit le début de phrase entre guillemets, la virgule, un espace suivi du nom que vous avez indiqué. Le point-virgule après un PRINT empêche le retour à la ligne, il est facultatif. Dans certains cas il est cependant indispensable.

Taper RUN, suivi de [RETURN] pour obtenir l'exécution du programme. Cette commande initialise toutes les variables numériques à zéro et toutes les variables chaînes à vide. Elle amène l'ORIC à la lecture de l'instruction 10 etc ... Vous pouvez ainsi répéter plusieurs fois le processus en changeant de nom à chaque fois.

Bien que ce ne soit pas nécessaire avec ORIC, il est courant d'écrire l'instruction END à la fin du programme.

Tapez LIST, puis [RETURN], et voilà votre programme qui s'inscrit proprement en repoussant vers le haut le texte qui occupe l'écran au cas où il ne reste pas assez de place en bas.

Maintenant vous ajoutez ces lignes :

```
 50 ?"ENTRER VOTRE ANNÉE DE NAISSANCE"
 60 INPUT AN
 70 LET AGE = 1983 – AN
100 ?"VOUS AVEZ "; AGE ;" ANS, ENVIRON, "; N$
110 GOTO 200
200 END
```

L'instruction GOTO 200 indique à l'ORIC d'aller en 200.

Si vous avez commis une erreur lors de l'introduction d'une ligne, vous pouvez l'effacer tout simplement en écrivant le n° de la ligne suivi de [RETURN].

Essayez, tapez 60, [RETURN], puis LIST et [RETURN].

La ligne 60 a disparu. Ecrivez-la à nouveau, lister à nouveau, recommencer l'exécution du programme ... Tout est en ordre.

.....

EXECUTION SOUMISE A UNE OU PLUSIEURS CONDITIONS.

Dans ce qui précède l'ordinateur exécutait les instructions dans l'ordre sans se poser de questions. Amenons le à prendre des décisions après un test sur les variables. Ajoutons ces lignes : (Tout le monde ne sera pas forcément d'accord).

```
 80 ? "EXCUSEZ MA QUESTION, MAIS ETES-VOUS DU
    SEXE FEMININ "; N$ ; " (OUI/NON) ? "
 90 INPUT A$
 95 IF A$ = "OUI" THEN 150
150 ? "EH BIEN ! "; N$ ; "UNE SEDUISANTE JEUNE
    FILLE COMME VOUS ";
```

160 ? "DOIT AVOIR A PEU PRES 18 ANS."

La ligne 95 contient un test sur le contenu de la variable A$. Si la réponse est oui, alors ORIC passe à la ligne 150. (IF ... THEN correspond à : SI ... ALORS).

Si A$ contient autre chose que "OUI" alors l'assertion A$ = "OUI" est fausse ; dans ce cas le programme continue à la ligne suivante, écrit l'âge et s'arrête.

Remarquez le point virgule à la fin de la ligne 150, il provoque l'écriture du mot DOIT à côté du mot VOUS sans retour à la ligne. Essayez le même programme sans ce point virgule pour voir la différence. (Instructif!). Observez les espaces.

A la fin de ce chapitre il sera question de ELSE.

Vous avez pu remarquer, en variant les essais, que seule la réponse OUI produisait l'effet attendu. OK ou VRAI ou EN QUELQUE SORTE ne sont pas admis à la place de OUI et considérés comme réponse fausse.

Soyez très rigoureux, vous voilà avertis ! Si vous souhaitez accepter plusieurs réponses alors écrivez une nouvelle ligne 100 qui supplantera automatiquement l'ancienne :

100 IF A$ = "OUI" OR A$ = "OK" OR A$ = "FEMININ" THEN 150

Tout ce que nous avons écrit jusqu'ici, l'a été en majuscules. La touche SHIFT qui correspond au passage en majuscules sur une machine à écrire ne produit pas le passage MIN/MAJ.

Si vous actionnez simultanément les touches CTRL et T le clavier de l'ORIC devient semblable à celui d'une machine à écrire. On obtient les minuscules et le passage aux majuscules se fait par appui sur SHIFT. Si vous faites à nouveau contrôle T (CTRL et T) ORIC revient à l'état précédent, vous n'avez que les majuscules. Sur la ligne du haut de l'écran, à droite, l'indication CAPS vous permet de savoir que le clavier n'écrit

que des CAPITALES (autre nom des MAJUSCULES).

La différence est importante. Si vous tapez run en minuscules, vous obtenez

? SYNTAX ERROR

Toutes les commandes BASIC et les variables doivent être écrites en majuscules.

Vous pouvez pianoter sur le clavier à votre guise, cela ne détériorera pas l'ORIC. Le pire qu'il puisse arriver est que l'ordinateur se mette en "boucle folle" ou que l'écran soit altéré. Si des choses bizarres apparaissent à l'écran, n'hésitez pas, actionnez le bouton RESET. Si les choses ne rentrent pas dans l'ordre alors il n'y a plus qu'à débrancher l'ORIC. Attendez quelques instants, puis branchez à nouveau. Tout repart à zéro. ORIC n'a pas souffert, mais vous, peut être, car vous avez perdu le programme en cours, et tout est à recommencer !

Que faire quand vous êtes fatigué d'un programme et que vous voulez vous en débarrasser ? Au lieu d'éteindre, il suffit de taper NEW. Cela efface le programme et remet à zéro les variables numériques, et à vide les variables chaînes.

LES BOUCLES

Avec l'ORIC nous avons pu voir comment un ordinateur est capable de prendre une décision selon qu'une condition est vraie ou fausse. Il est également capable de répéter une action autant de fois qu'on le désire.

Par exemple, si vous désirez que votre ORIC affiche tous les nombres de 1 à 1000, dans l'ordre, en les faisant défiler de bas en haut sur la partie gauche de l'écran, vous pourriez écrire :

```
10 ? 1
20 ? 2
30 ? 3
40 ? 4 .....
```

Mais vous allez vite vous lasser, bien avant d'être arrivé à 1000! Par bonheur, il existe une instruction BASIC qui va nous simplifier la vie :

FOR . . . . TO . . . . NEXT

C'est une boucle qui va répéter une instruction en comptant ; ici de 1 à 1000.

```
10 FOR X = 1 TO 1000 STEP 1
20 PRINT X
30 NEXT X
40 PRINT "OUF ! C'EST FINI ! "
```

La ligne 10 initialise le compteur X à 1, on passe à la ligne 20. Le contenu de la variable X est affiché : c'est 1. On passe à 30 : NEXT signifie "LE SUIVANT", alors on remonte en ligne 10, la variable X est augmentée de 1, comme l'indique l'instruction STEP 1 (STEP signifie PAS). En 20, on affiche 2, nouveau contenu de X, et le processus se répète jusqu'à ce que X atteigne 1000. Comme on a spécifié en ligne 10 de ne pas dépasser 1000, quand X vaut 1000 en ligne 30, on ne remonte plus ligne 10, on passe en ligne 40 et le message prévu s'inscrit.

Si vous changez, et mettez STEP 2, vous obtiendrez la liste 1, 3, 5, 7 etc ... Si vous n'écrivez pas l'instruction STEP, ORIC ajoutera 1.

Essayez avec d'autres pas de votre choix.

Les boucles FOR/NEXT peuvent décompter, alors l'instruction STEP est obligatoire et négative :


```
10 FOR X = 1000 TO 1 STEP – 1
```

Si vous commettez des erreurs comme :

```
FOR X = 4 TO 2 STEP 5
```

ou

```
FOR X = 5 TO 100 STEP – 1
```

L'action située dans la boucle sera quand même exécutée au moins une fois car le test se fait avec l'instruction NEXT.

Une autre utilisation de FOR/NEXT est la pause. Vous avez sûrement eu du mal à lire les nombres qui s'affichaient tellement ils défilaient vite. Pour ralentir la vitesse d'affichage vous pouvez écrire :

```
25 FOR PAUSE = 0 TO 10 : NEXT PAUSE
```

ou, tout aussi bien

```
25 FOR I = 0 TO 10 : NEXT
```

Le I après NEXT est optionnel.

Cela revient à dire à l'ORIC chaque fois qu'il vient d'afficher un nombre : «compte jusqu'à dix avant de continuer».

Attention à l'utilisation de IF . . . THEN à l'intérieur des boucles FOR/NEXT, car le branchement inconditionnel peut faire sortir de la boucle en ignorant toutes les instructions qui suivent et cela peut causer des erreurs.

Une méthode plus aisée pour obtenir une pause est l'utilisation de l'instruction WAIT (ATTEND).

25 WAIT N provoquera un arrêt de l'exécution du programme de N fois 10 millisecondes. N=100 correspond à 1 s.

. . . . . .

SOUS-PROGRAMMES

Au point où nous en sommes, vous pouvez vous demander ce qui arrive lorsqu'une partie de programme doit être utilisée plusieurs fois et que la boucle FOR/NEXT n'est pas l'outil approprié.

Par exemple, dans le programme précédent, vous pourriez avoir envie de signaler que l'arrêt entre l'affichage de 2 nombres est voulu.

Vous envoyez le programme à un sous-programme où il attend, écrit le message et revient au programme juste après l'endroit d'où il s'était détaché.

```
 10 FOR X = 1 TO 10
 20 GOSUB 1000
 30 PRINT X
 40 NEXT X
 50 END
1000 ? "PETITE PAUSE"
1010 WAIT 50
1020 RETURN
```

Notez ici que l'instruction END ligne 50 est nécessaire.

x x x x x x x x x x

BRANCHEMENT MULTIPLE OU GOTO CALCULÉ

Parfois, dans un programme, il peut être utile d'aller à tel ou tel endroit du programme selon le résultat d'un calcul. C'est facile avec ON ... GOTO.

Tout ce qu'il faut savoir c'est les résultats possibles du calcul et les adresses où l'on désire se rendre.

```
 50 INPUT "CHOISIR 1, 2 ou 3" ; X
 60 ON X GOTO 100, 200, 300.
 70 ? "IL FALLAIT CHOISIR 1, 2 ou 3! " : STOP
100 ? "CHOIX 1" : STOP
200 ? "CHOIX 2" : STOP
300 ? "CHOIX 3" : STOP
```


En ligne 50 on attend 1, 2 ou 3. Si vous avez tapé 2 par exemple la ligne 60 va vous brancher en ligne 200.

Si vous n'entrez pas de nombre le programme va continuer ligne 70, où l'on vous rappelle au bon sens. STOP provoque l'arrêt. Notez les deux points (:) pour séparer deux instructions sur une même ligne de programme.

Une commande similaire est ON .... GOSUB qui vous branche à un sous-programme, au retour le programme reprendra à la ligne qui suit l'instruction ON ... GOSUB.

ELSE

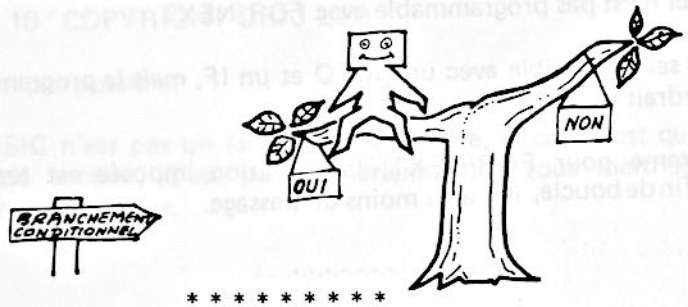
IF ... THEN, nous l'avons vu est une instruction utilisable, telle quelle.

On peut l'améliorer avec ELSE (SINON)

Etudiez :

```
10 FOR X = 1 TO 5
20 INPUT A
30 IF A > 10 THEN ? "TROP GRAND" ELSE ? "OK"
40 NEXT
```

Si la condition est vraie, le message TROP GRAND va s'afficher, sinon ce sera OK et le programme va continuer.



* * * * * * * * *

REPEAT / UNTIL

Si vous voulez répéter des instructions un certain nombre de

fois, connu d'avance, alors il est opportun d'utiliser la boucle FOR/NEXT. La répétition se fera autant de fois qu'indiqué au début.

Exemple :

avec FOR N = 1 TO 5, la boucle sera exécutée 5 fois. Si vous voulez que la répétition ait lieu jusqu'à ce qu'une certaine condition soit remplie, il vous est difficile, à priori, de connaître le nombre de boucles nécessaires et vous ne pouvez pas l'indiquer dans le compter de FOR/NEXT.

REPEAT (RÉPÉTER) autorise une répétition indéfinie, et teste en fin de boucle pour voir si une condition est satisfaite. La fin de boucle contient le mot UNTIL (JUSQU'A CE QUE).

Voici un court programme de démonstration :

```
10 REPEAT
20 D = D + INT (RND(1)* 6) + 1
30 ? D
40 UNTIL D 〉 20
50 END
```

C'est la simulation d'un jet de dé qui est jeté autant de fois qu'il faut tant que le total des points ne dépasse pas 20. On ne peut pas savoir d'avance le nombre de coups nécessaires.

Ceci n'est pas programmable avec FOR/NEXT.

Ce serait possible avec un GOTO et un IF, mais le programme perdrait en clarté.

Comme pour FOR/NEXT, la condition imposée est testée en fin de boucle, il y a au moins un passage.

. . . . . .

Avant d'aller plus loin pour explorer des possibilités plus captivantes comme les sons et les dessins, il reste une dernière chose à vous indiquer pour favoriser la relecture de vos programmes.

L'utilisation de REM (abréviation de remarque).

Lorsqu'ORIC trouve l'instruction REM, il passe à la ligne suivante et ignore ce qui suit : ce sont des renseignements pour le programmeur.

Voici un exemple :

```
10 REM COPYRIGHT F. BLOGGS
20 FORN= 1TO 10 : REM COMPTEUR DE BOUCLES
30 ? "FRED EST SURDOUÉ"
40 NEXT
50 END
60 REM ASSEZ NIAIS CE PROGRAMME !
```

REM est utile pour indiquer le rôle d'un sous-programme.

```
1000 REM COURTE PAUSE
1010 WAIT 100
1020 RETURN
```

L'apostrophe (') est une abréviation de REM, mais non utilisable en début de ligne.

```
10 ? "SALUT" ' POUR SALUER
```

est licite, mais

```
10 ' COPYRIGHT ORIC LTD
```

n'est pas possible.

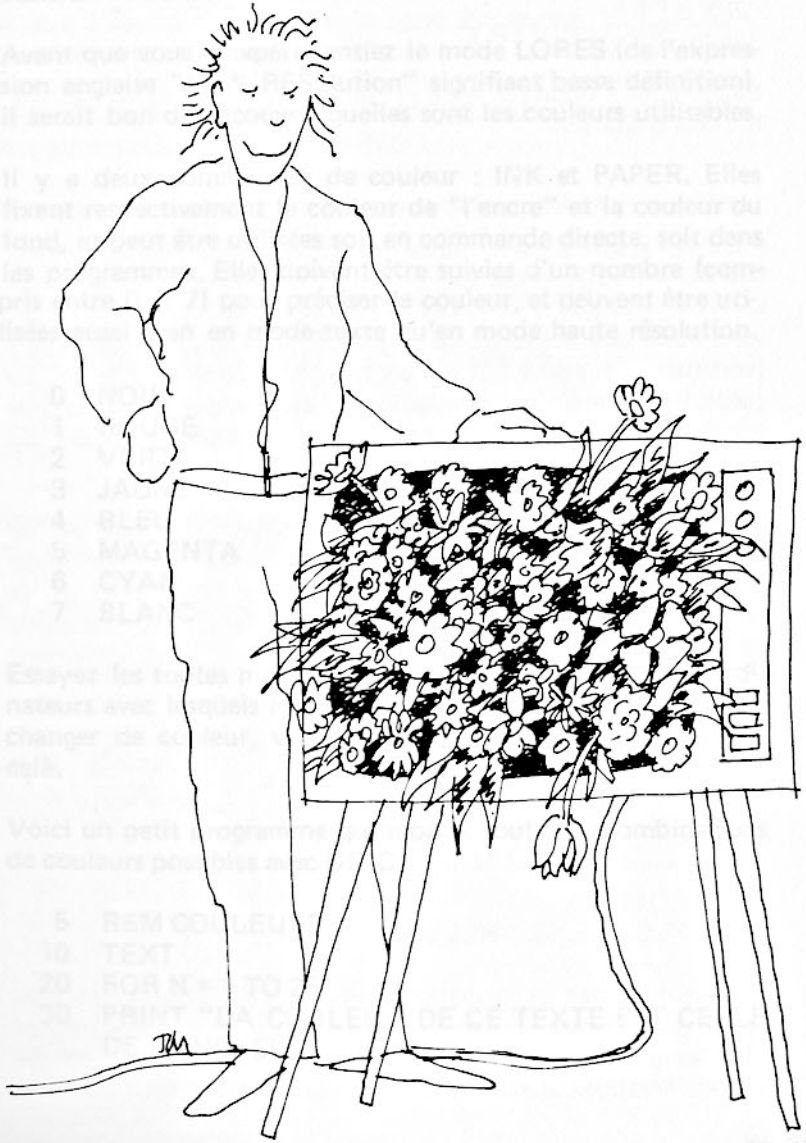
BASIC n'est pas un langage bien difficile, et ceci n'est qu'un guide succint. C'est par l'expérience que vous maîtriserez mieux ce langage.

. . . . . .

J'entends et j'oublie.
Je vois et je me souviens.
Je fais et je comprends.

Vieux proverbe chinois.

CHAPITRE 4
Couleurs et Graphiques



4. COULEURS ET GRAPHIQUES

Quand vous branchez ORIC, il se met automatiquement en mode texte (commande : TEXT), c'est à dire que vous pouvez utiliser l'écran directement pour écrire, et, quand ce dernier est rempli, les lignes se décalent automatiquement vers le haut. Le mode texte est également utilisé pour les graphiques en basse définition. (on dit aussi basse résolution).

Avant que vous n'expérimentiez le mode LORES (de l'expression anglaise "LOW RESolution" signifiant basse définition). il serait bon de découvrir quelles sont les couleurs utilisables.

Il y a deux commandes de couleur : INK et PAPER. Elles fixent respectivement la couleur de "l'encre" et la couleur du fond, et peut être utilisées soit en commande directe, soit dans les programmes. Elles doivent être suivies d'un nombre (compris entre 0 et 7) pour préciser la couleur, et peuvent être utilisées aussi bien en mode texte qu'en mode haute résolution.

0 NOIR
1 ROUGE
2 VERT
3 JAUNE
4 BLEU
5 MAGENTA
6 CYAN
7 BLANC

Essayez les toutes maintenant. Si vous avez l'habitude d'ordinateurs avec lesquels l'écran doit être effacé avant de pouvoir changer de couleur, vous verrez qu'ORIC n'a pas besoin de celà.

Voici un petit programme qui montre toutes les combinaisons de couleurs possibles avec ORIC.

```
 5 REM COULEURS
10 TEXT
20 FOR N = 1 TO 25
30 PRINT "LA COULEUR DE CE TEXTE EST CELLE
   DE L'ENCRE"
```

```
 40  NEXT N
 50  FOR I = 0 TO 7
 60  FOR P = 0 TO 7
 70  INK I : PAPER P
 80  WAIT 100
 90  NEXT P
100  NEXT I
210  INK7 : PAPER 4
```

variante : 55 INK I
70 PAPER P

Evidemment, quand le fond et l'encre sont de la même couleur, le texte disparaît !

****

Pour les graphiques en basse résolution, vous pouvez utiliser l'écran en mode texte ou introduire les ordres LORES 0 ou LORES 1. La partie utilisable de l'écran est repérée de 0 à 38 suivant l'axe des x (horizontal) et de 0 à 26 suivant l'axe des y (vertical). Le point 0,0 est en haut à gauche de l'écran. La colonne située à l'extrême gauche n'est pas disponible car elle contient le préfixe de commande qui contient la couleur du fond (PAPER) de cette ligne.

La colonne suivante commande la couleur du fond ou de l'encre (INK), mais peut être utilisée en mode texte. En mode LORES 0 ou LORES 1, l'écran est effacé (fond noir), et la commande préfixée qui sélectionne les caractères standards ou un 2ème jeu de caractères (graphiques ...) est également placé dans la colonne de gauche.

LORES 0 utilise le jeu de caractères standard, tandis que LORES 1 utilise le 2ème jeu de caractères.

Essayez ceci :

```
10 LORES 0
20 PLOT 16, 12, "HELLO".
```

Si vous exécutez ce court programme, HELLO apparaîtra au centre de l'écran. Vous pourriez utiliser le programme du chapitre 9 pour définir de nouveaux caractères à la place du jeu de caractères standard. L'ordre PLOT (marquer) vous évite

d'intervenir dans la mémoire de gestion de l'écran (ordre POKE : introduire).

Si maintenant vous tapez

```
10 LORES 1
```

et que vous ne changez pas la ligne 20, le mot HELLO sera remplacé par d'étranges symboles. Ce sont les caractères graphiques, et, plus particulièrement, ceux qui ont le même code ASCII que les lettres du mot HELLO: C'est la seule différence entre LORES 0 et LORES 1.

Le programme suivant imprimera tous les caractères du 2ème jeu :

```
 5  REM ** CARACTERES GRAPHIQUES **
10  FOR N = 32 TO 128
20  PRINT N, CHR$ (27) ; "I" ; CHR$ (N)
30  PRINT
40  WAIT 25
50  NEXT N
```

Vous pouvez utiliser ce programme pour choisir des caractères vous permettant de créer vos propres graphiques.

C'est l'une des façons de l'employer.

```
 1  REM *** MONSTRES ***
 2  REM *** LORES 0/1 ***
 5  LORES 1
 6  D = 0
 9  REPEAT
10  A$ = "F9" : B $ = "6I"
20  FOR C = 0 TO 35
30  PLOT  C, D, A$
35  PLOT  C, D + 1, B$
45  PLOT  C, D, "  "
50  PLOT  C, D + 1, "  "
55  NEXT C
56  SHOOT
60  D = D + 2
```

VARIANTE : ajouter
25 IF C = 35 GOTO 45

```
70  UNTIL D = 26
75  EXPLODE
80  CLS
90  RUN
```

Les caractères dans A$ et B$ n'apparaissent pas sous leur forme normale puisque le mode LORES 1 a été choisi. La suite du programme déplace le caractère composite le long des rangées successives jusqu'à ce que la ligne 26 soit atteinte ; à ce moment il explose !

Pour voir les caractères standard, il suffit d'écrire LORES 0 en ligne 5. Si vous désirez mélanger sur la même image des caractères standard et des caractères graphiques, par exemple pour mélanger des textes et des dessins, c'est très facile. En mode LORES 1, l'instruction CHR$ (8) ramène aux caractères standard, puis l'instruction CHR$ (9) restitue le mode initial.

En les utilisant dans l'ordre inverse, vous pouvez bien sûr imprimer des caractères graphiques en mode LORES 0. Voici deux programmes illustrant cet effet. Dans tous les programmes utilisant LORES 0 et LORES 1, il est commode de supprimer le curseur clignotant en appuyant simultanément sur CTRL et sur Q. On fait réapparaître le curseur en répétant l'opération.

```
 5  REM ** TEXTE LORES 1 **
10  LORES 1
20  A$ = CHR$(8)+"SALUT" + CHR$(9)
30  FOR N = 2 TO 24
40  PLOT N, N, "KKKK"
50  PLOT N,26-N,AS
60  NEXT N
70  WAIT 500
80  CLS
```

```
 5  REM ** 2ND CLAVIER LORES 0**
10  LORES 0
20  A$ = CHR$(9) + "SALUT" + CHR$(8)
30  FOR N = 2 TO 24
40  PLOT N, N, "KKKK"
50  PLOT N,26-N,A$
```



```
60  NEXT N
70  WAIT 500
80  CLS
```

POSITIONS SUR L'ECRAN

Si vous désirez savoir quel caractère figure à une position donnée de l'écran en mode texte ou en mode basse résolution, utilisez l'instruction SCRN(X,Y).

Effacez l'écran au moyen de la touche CLS. Le curseur se trouve maintenant en haut et à gauche de l'écran. Ecrivez :

```
PLOT 10, 20, "A"
```

Un A majuscule apparaîtra près du bas de l'écran.

Ecrivez :

```
PRINT SCRN(10, 20)
```

Le nombre 65, qui est le code ASCII de la lettre A, sera affiché

Voici un petit programme qui répète (instruction: REPEAT) une boucle jusqu'à ce qu'un missile tombe sur une cible. SCRN(X,Y) sert à détecter que le missile est sur la case voisine de la cible (le code ASCII du signe + est 43, voir ligne 220) ; le programme se termine par une explosion. Après exécution de ce programme, changez le mode dans lequel il travaille en ajoutant :

```
115 LORES 0  ou  115 LORES 1
```

Ceci vous montrera l'effet obtenu en fonction du mode choisi.

```
100   : REM EMPLOI DE SCRN(X,Y)
110   :CLS:INK1:PAPER4
120   : FOR N = 20 TO 25
130   :       PLOT N,26,"+"
140   : NEXT N
150   : REPEAT
160   :       A=INT(RND(1)*36 + 2)
```

```
170   :     FOR P = 0 TO 24
180   :     PLOT A,P,"V"
190   :     WAIT 4
200   :     PLOT A,P," "
210   :     NEXT P
220   : UNTIL SCRN(A,P + 1) = 43
230   : EXPLODE
```

Ce programme est présenté de façon à être plus compréhensible. Il fonctionne tout aussi bien si l'on supprime les deux-points et les espaces. Les explications détaillées sur ce sujet sont données au chapitre 12.

Pour terminer ce paragraphe sur les dessins en basse résolution, voici un programme qui montre comment on peut disposer en cercle des carrés colorés.

```
10   REM * RONDES BARIOLÉES *
20   LORES 0
30   STP = 2*PI/50
40   R = 10 : X=20:Y = 10
50   REPEAT
60   E = 18 + RND(1)*6
70   PLOT X + R*SIN(C),Y + R*COS(C), E
80   C = C + STP
90   UNTIL C > 2*PI
100  REPEAT:UNTIL KEY$ <> " "
110  CLS
120  RUN
```

x x x x x x x x x x

DESSINS EN HAUTE RESOLUTION (Définition fine)

Pour dessiner en haute résolution, il faut frapper HIRES (de l'anglais, HIgh RESolution) au clavier. Essayez maintenant.

Vous constatez que l'ensemble de l'écran devient noir, ne laissant que trois lignes destinées au texte à la partie inférieure. Ceci est très commode, car cela permet d'introduire des instructions de tracé, et d'en voir l'effet au-dessus.

En mode direct, vous pouvez donc utiliser ORIC comme une table traçante pour essayer vos instructions. Lorsqu'elles sont

correctes, vous pouvez les intégrer dans vos programmes.

Pour revenir en mode texte, il suffit d'introduire TEXT, et l'écran reprendra son aspect initial. Les instructions TEXT et HIRES peuvent être utilisées en tant qu'ordres à l'intérieur des programmes.

Avant de commencer à dessiner, il faut imaginer que l'écran est divisé en 240 positions horizontales (repérées de 0 à 239) et en 200 positions verticales (repérées de 0 à 199). Les positions horizontales sont baptisées X, . Les positions verticales sont baptisées Y. Si vous avez déjà utilisé des graphiques, ceci vous parait familier ; la seule différence est que l'origine (0, 0) est en haut et à gauche de l'écran, l'axe des Y étant orienté vers le bas.

L'utilisation de l'ORIC pour dessiner est facile grâce à plusieurs instructions particulières, que nous allons examiner une par une pour voir leurs effets.

CURSET place le curseur à la position X, Y, ou il marquera ce point. Elle doit être suivie de trois paramètres (ce sont des nombres dont ORIC a besoin). Le curseur HIRES ne clignote pas comme celui du mode texte. Exemple :

```
CURSET 120, 100, 1
```

déplace le curseur au centre de l'écran, et marque un pixel (petit point). Le premier paramètre représente la position horizontale X (ici 120), le second la position verticale Y (ici 100), le troisième est le nombre FD, dont les codes sont :

0 couleur du fond (du papier)
1 couleur de l'encre (devant)

(FD pour Fond/Devant)

2 couleur inversée
3 pas de couleur (ne rien faire)

Maintenant introduisez HIRES au clavier et expérimentez l'instruction CURSET.

L'instruction suivante est CURMOV. Elle est similaire à CURSET, mais les déplacements X et Y se comptent à partir de la dernière position du curseur. Ici encore, il faut donner les trois paramètres X, Y et FD. Assurez vous bien que l'instruction ne fait pas sortir le curseur de l'écran, ce qui donnerait un message d'erreur. DRAW X, Y, FD trace une ligne droite entre la position actuelle du curseur et cette position augmentée de X et de Y.

Essayez ce petit programme. Vous constaterez qu'il dessine un carré. Remarquez que les nombres négatifs dessinent de droite à gauche et de bas en haut. Si la forme n'est pas suffisamment carrée, essayez de modifier les lignes 30 et 50.

```
 5 REM **CARRÉ**
10 HIRES
20 CURSET 60, 40, 3
30 DRAW 120, 0, 1
40 DRAW 0, 120, 1
50 DRAW – 120, 0, 1
60 DRAW 0, – 120,1
```

NOTA : Un changement de mode, ou simplement une introduction de HIRES, effacera définitivement votre dessin !

MOTIFS : Instruction PATTERN

ORIC a encore un tour dans son sac. Quand vous le branchez, l'instruction DRAW est définie pour tracer une ligne continue. Il est possible, cependant, de tracer à volonté des lignes en tirets, pointillés, etc ...

Voici comment faire. Vous vous souvenez qu'ORIC travaille en octets de 8 bits, de sorte qu'il peut compter de 0 à 255 en utilisant 8 zéros ou uns.

Au départ, le nombre 255 est le paramètre définissant le motif (255 s'écrit 11111111 en binaire). Vous pouvez mettre à la place n'importe quel nombre compris entre 0 et 255 pour obtenir des motifs différents. Pour obtenir des pointillés égaux, écrivez PATTERN 15.

En effet 15 s'écrit 00001111 en binaire, donc seule la moitié du trait s'imprime. Essayez le programme de tracé du carré, mais cette fois-ci en modifiant le motif. Rien ne vous empêche de faire deux traits continus, un pointillé et un tireté. Pour mieux comprendre le principe, ajoutez la ligne suivante au début :

```
15 PATTERN 170
```

(170 s'écrit 10101010 en binaire)

Vous obtenez des pointillés !

xxxxxxxxxx

Pour avoir une idée saisissante des possibilités d'ORIC en haute résolution, essayez ce court programme qui crée des jeux d'interférence en traçant des lignes proches l'une de l'autre.

```
 5 REM **MOIRE**
10 HIRES
20 FORA = 0TO1
30 FORB = 0TO239STEP6
40 CURSET 0,199*A,3
50 DRAW B,199–398*A,1
60 CURSET 239,199*A,3
70 DRAW – B, 199–398*A,1
80 NEXT B : NEXT A
```

CHAR

Si vous utilisez l'instruction PRINT en mode haute résolution, vous n'obtenez le texte que sur les 3 lignes inférieures. Il y a cependant une instruction qui permet d'écrire à tout endroit de l'écran en haute résolution. Ecrivez NEW pour effacer la mémoire, puis HIRES. Placez le curseur au milieu de l'écran par

```
CURSET 120, 100, 3
```

Puis écrivez CHAR 65, 0, 1. Un A majuscule apparaît au centre de l'écran.

L'instruction CHAR est suivie de trois paramètres X, S, FD.

X est le code ASCII (de 32 à 127)
S détermine les caractères (0 = standard, 1 = graphiques)
FD a été défini plus haut (de 0 à 3)

Les lettres ASCII (que l'on prononce habituellement "ASKI") sont les initiales de American Standard Code for Information Interchange. L'usage de ce code, qui attribue un numéro conventionnel aux lettres, aux chiffres et aux symboles, est très répandu (voir annexe D). Il existe en BASIC une instruction, ASC, qui détermine le code d'un caractère contenu dans une chaîne, et qui permet une programmation plus condensée, comme le montre le programme suivant.

```
 5 REM **CASCADE**
10 HIRES
20 CURSET 10, 30, 3
30 N$ = "ORIC, C'EST MOI"
40 FOR A = 1 TO LEN (N$)
50 CHAR ASC (MID$(N$,A,1)),0,1
60 CURMOV 10,10,0
70 NEXTA
```

Les lignes 40 à 70 forment une boucle qui découpe la chaîne N$ et décale les caractères suivant l'instruction CURMOV.

Remplacez N$ par votre nom ou modifiez les paramètres de CURMOV et voyez ce qui se produit. Faites attention de ne pas sortir de l'écran, ce qui produirait un message d'erreur.

xxxxxxxxxx

CERCLE

Pour tracer des cercles, il suffit d'écrire CIRCLE, suivi de deux nombres, d'abord le rayon, puis le code FD. Le centre du cercle se trouve à la position initiale du curseur. Veillez à ce que la valeur du rayon ne fasse pas sortir le cercle de l'écran. Essayez ceci en mode haute résolution.

```
CURSET 120, 100, 3
CIRCLE 50, 1
```


Si préalablement vous avez modifié l'instruction PATTERN, le cercle apparaîtra en pointillés, etc ... Le programme suivant permet un effet intéressant.

```
100 : REM **NAPPERON**
110 : HIRES
120 : CURSET 120, 100, 3
130 : FOR N = 99 TO 1 STEP – 1
140 :   CIRCLE N,1
150 :   PATTERN 100 – N
160 : NEXT N
```

POINTS

Si vous désirez connaître la couleur (fond ou encre) d'un pixel déterminé (par exemple pour savoir s'il y a un extra-terrestre au centre de l'écran), utilisez l'instruction POINT. Pour l'étudier, travaillez en mode direct.

Ecrivez successivement :

```
HIRES
CURSET 0,0,0 (curseur placé au point 0,0 en couleur de
              fond)
TEXT
PRINT POINT (0,0)
```

Le point 0,0 étant en couleur de fond, vous obtenez 0.

Maintenant écrivez

```
HIRES
CURSET 0,0,1 (point 0,0 en couleur d'encre)
TEXT
PRINT POINT (0,0)
```

Cette fois-ci vous trouvez – 1 puisque le pixel est en couleur d'encre.

FILL

C'est une instruction très utile qui remplit une zone déterminée sur un certain nombre de rangées et un certain nombre

de segments avec une couleur et un motif déterminé par la commande préfixée. (voir chapitre 3 et l'annexe C). Il y a 200 rangées et 40 segments par rangée.

Voici un court programme qui montre la finesse de détails qu'ORIC peut réaliser.

```
 5 HIRES
10 FOR N=0 to 199
20 X=RND(1)*8+16
30 FILL 1,1,X
40 NEXT N
```

La ligne 20 détermine au hasard la couleur du fond, de la ligne 0 jusqu'à la ligne 199.

Faites des essais pour voir ce que produisent les autres préfixes de commande.

Cet autre programme illustre le mélange de motifs graphiques, de couleurs et de clignotements.

Remarquez que la ligne 130 permet d'éviter que l'utilisation des préfixes ne perturbe la synchronisation du balayage.

```
100    : REM **INSTRUCTION FILL**
110    : HIRES
120    : REPEAT
130    : A=RND(1)*128+1:IF A> 23 AND A< 32 THEN
         130
140    : CURSET RND(1)*90+10,RND(1)*90+10,1
150    : FILL RND(1)*90+1,1,A
160 : UNTIL KEY$<>""
```

Lettres en double hauteur et caractères clignotants

ORIC contient un programme qui permet de réaliser des caractères clignotants ou en double hauteur.

Si vous consultez l'appendice C, vous trouverez un tableau précisant les effets possibles. Il faut également connaître le tableau des caractères de contrôle (appendice B).

Control D commande la double hauteur des lettres. Elle est aussi réalisée au moyen d'un ordre d'impression. Ecrivez PRINT CHR$ (4). Tout ce que vous écrirez maintenant apparaîtra deux fois, sur deux rangées consécutives. Faites plusieurs essais. Comment arrêter ? Cette instruction étant à simple commande, il suffit de taper une seconde fois ? CHR$ (4) pour l'annuler. Toutes les commandes de contrôle fonctionnent de cette façon. Le programme suivant montre l'effet obtenu :

```
10 PRINT CHR$(12)
20 PRINT CHR$(4);CHR$(27);"N GRANDES LETTRES
   CLIGNOTANTES"
30 PRINT CHR$(4)
```

Ne vous inquiétez pas si cela paraît terrifiant. Nous allons le commenter ligne par ligne.

La ligne 10 efface l'écran. En outre, elle garantit que l'on commence à la ligne supérieure ; vous verrez pourquoi c'est important quand nous ajouterons la ligne 15.

La ligne 20 contient plusieurs ordres. CHR$(4) contrôle la double hauteur (ce qui évite de taper 2 fois). CHR$(27) est le code ASCII pour "escape" (pour entrer en mode "escape" c'est à dire sortir du mode normal), et le N entre guillemets fait apparaître le reste du texte en double hauteur et clignotant ; cependant il n'est pas imprimé lui-même.

La ligne 30 rétablit le mode normal.

Amusez-vous à modifier le message entre guillemets, ainsi que la commande préfixée, par exemple : " J HELLO" apparaîtra en double hauteur fixe (voir annexe C touche ESC "Escape" = échapper, sortir).

Quand vous aurez réussi à obtenir plusieurs effets, ajoutez la ligne 15 PRINT puis exécutez le programme. A l'aspect du résultat, vous comprendrez qu'il est important de débuter sur un numéro de ligne pair (0, 2, 4, ...).

Pour terminer, voici un programme qui utilise plusieurs instructions de tracé en haute résolution. Il montre aussi com-

ment les informations nécessaires aux instructions de tracé peuvent être fournies par des instructions DATA.

```
100   : REM **LE VELO DE VOS REVES**
110   : HIRES
120   : X = 100:Y = X
130   : CURSET X, Y, 1
140   : PAPER6:INK1
150   : CIRCLE70,1 '   **ROUE AVANT**
160   : REPEAT
170   :   CURSETX,Y,3
180   :   DRAW69*SIN(F),69*COS(F),1
190   :   F = F + .1
200   : UNTIL F> 2*PI
210   : F = 0
220   : CURSET 200,140,3
230   : CIRCLE30,1          **ROUE ARRIERE**
240   : REPEAT
250   :   DRAW29*SIN(F),29*COS(F),1
260   :   CURSET200,140,3
270   :   F = F + .1
280   : UNTILF> 2*PI
290   : CURSET 100,15,3
300   : REPEAT
310   :   READ A,B
320   :   DRAW A, B, 1
330   : UNTIL B = 25
340   : CURSET 160,20,3
350   : FOR N = 1 TO 10' **TEXTE 1 **
360   :   READ L
370   :   CHAR L,0,1
380   :   CURMOV 7,0,3
390   : NEXT
400   : CURSET 160,32,3
410   : FOR N = 1 TO 9'** TEXTE 2 **
420   :   READ L
430   :   CHAR L,0,1
440   :   CURMOV 7,0,3
450   : NEXT
500   : DATA – 10,0,10,10,0,20,0,– 20,40,0
510   : DATA–10,–10,15,0,–5,10,60,60,0,25
520   : DATA 79,82,73,67,32,82,73,68,69,83
```



```
530   : DATA 84,79,32,87,79,82,75,33,33
```

Exercice : Amusez-vous à changer les données en DATA pour obtenir le texte de votre choix.

CHAPITRE 5
Éditeur du Basic

5. EDITEUR DU BASIC

Quand vous écrivez un programme et que vous voulez modifier une ligne, il existe plusieurs méthodes pour changer ou effacer les instructions.

- Pour supprimer toute la ligne, il suffit d'écrire le numéro suivi d'un [RETURN].

```
10 ? "HELLO"
20 ? "AIE !"
30 ? "AU REVOIR"
```

Tapez

20 puis [RETURN]

Faire

LIST, la ligne 20 a disparu.

- Si vous voulez effacer la totalité du programme alors c'est NEW qui convient.

- Si lors de l'écriture d'une ligne, vous vous apercevez d'une erreur, vous pouvez effacer le dernier caractère en appuyant sur la touche DEL. Un deuxième appui et vous effacez un deuxième caractère.

- Pour effacer toute la ligne en cours d'écriture vous pouvez utiliser CTRL X. Une barre oblique (\ ) apparaît en bout de ligne, et le curseur passe en début de ligne suivante.

- Lors du listage d'un long programme, il défile trop vite pour permettre la lecture. Appuyez une fois sur la barre d'espace, cela arrête le défilement. Il suffit d'appuyer sur une touche quelconque pour obtenir l'affichage de la suite. Pour arrêter complètement le listage, appuyer sur CTRL et C simultanément. CTRL C sert aussi à interrompre un programme BASIC en cours d'exécution.

  Après un arrêt provoqué par CTRL C, la reprise est pos-

sible en tapant CONT (suivi d'un [RETURN]) CONT (continuer). Cette manœuvre échoue si pendant l'arrêt on a modifié le programme ou simplement une variable. On doit utiliser RUN ou GOTO ou LIST.

- Ce serait une grosse perte de temps s'il fallait taper à nouveau des lignes déjà introduites, surtout si elles sont longues. Noter en passant qu'elles ne doivent pas excéder 78 caractères. Au-delà, la sonnette, au joli timbre de l'ORIC, vous rappellera à l'ordre. Ne vous obstinez pas, vous perdriez du temps et des informations. Les très longues lignes dans un programme sont difficiles à lire et perturbent la disposition des instructions lors du listage.

Si vous aviez à modifier une ligne, ORIC vous offre un moyen facile pour la recopier. Voici un court programme pour l'expérimenter :

```
10 REM ** TEST D'EDITION**
20 A=20 : B=30
30 C=A * B
40 PRINT C
```

Si vous décidez que la ligne 20 doit être A = 25 : B = 5, et que la ligne 30 doit être C = A + B, voilà ce qu'il convient de faire. LIST, [RETURN], vous disposez alors du programme sur l'écran. Avant cela il est intéressant de nettoyer l'écran par (CTRL et L simultanés) (contrôle L).

Le déplacement du curseur est aisément obtenu grâce aux 4 touches fléchées qui sont de part et d'autre de la barre d'espace. Amener le curseur sur la ligne 20, à gauche du 2. Maintenir appuyée la touche CTRL, et en même temps appuyer sur la touche A. Le curseur bouge vers la droite et chaque caractère sur lequel il passe est stocké dans une mémoire tampon. (Stockage provisoire). C'est la relecture.

Quand le curseur est sur le 0 de A = 20, lâcher CTRL et A et taper 5. Un 5 va remplacer le 0. Continuer à recopier la ligne en utilisant CTRL A jusqu'à ce que le curseur soit

sur le 3 de B = 30. Relâcher CTRL et A, taper 5. Un 5 remplace le 3.

Comme vous ne voulez pas du 0 qui suit dans la ligne modifiée, appuyer tout bonnement sur [RETURN] et la nouvelle ligne va remplacer l'ancienne dans le programme. Sur l'écran vous voyez A = 25 : B = 50, ce qui peut vous faire douter de la réussite de l'opération entreprise.

Nettoyer l'écran par CTRL L par exemple et demander le listage. Eh voilà ! La ligne 20 est devenue A = 25 : B = 5.

Comme vous n'avez pas relu le 0 à la fin de la ligne, il n'a pas été stocké comme élément de cette ligne. Pour modifier la ligne 30, faire monter le curseur devant le 3, et copier en utilisant CTRL A jusqu'à ce que le curseur soit sur *. Remplacez * par +, copiez B et appuyez sur [RETURN].

Souvenez-vous que le déplacement du curseur sur l'écran commandé par les flèches ne modifie en rien les lignes du programme. Les lignes de programme sont créées ou modifiées en copiant des caractères figurant sur l'écran grâce à CTRL A, ou en ajoutant de nouveaux caractères à partir du clavier. CTRL X permet de sortir de la ligne, les touches fléchées permettent un déplacement sans copie, DEL permet l'effacement des caractères indésirés et [RETURN] va valider la nouvelle ligne.

Jusqu'à ce que vous soyez familiarisés avec ces méthodes, il est prudent après un vidage de l'écran, de lister la nouvelle ligne pour s'assurer qu'elle a bien été entrée en mémoire sous la forme désirée.

Vous verrez que vous pouvez créer et copier des lignes très rapidement avec un peu de pratique.

x x x x x x x x x x

TRON ET TROFF

Lorsque vous avez écrit un programme BASIC, et que, contrairement à votre attente, il ne produit pas les effets espérés,

s'obstine à fournir d'étranges résultats, ou plus simplement s'arrête en écrivant un message d'erreur, alors il devient très utile de pouvoir s'assurer que le déroulement pas à pas du programme est bien tel que vous le vouliez.

TRON (Trace ON), introduit dans une ligne de programme (il n'est pas possible de l'utiliser en mode direct) permet la recherche d'erreurs. A l'exécution, au-delà de l'instruction TRON, les numéros des lignes exécutées vont s'afficher à l'écran, en même temps que ce que le programme prévoit à l'affichage. Ces numéros sont entre crochets pour qu'on ne les confonde pas avec d'autres nombres donnés par le programme en cours.

Voici un exemple de programme qui ne marche pas :

```
 10 FOR N = 1 TO 4
 20 READ D
 30 ON D GOSUB 100, 200, 300, 400
 40 NEXT N
 50 STOP
100 ? "C'EST";
110 RETURN
200 ? "MOI";
210 RETURN
300 ? "ORIC";
310 RETURN
400 ?"VOYEZ-VOUS!"
410 RETURN
500 DATA 1,2,3
```

Le texte écrit en ligne 400 n'est pas affiché, le ON ... GOSUB ne fonctionne pas la 4ème fois.

La ligne 20 READ D cherche à lire les données de la ligne 500 DATA. Or il manque le 4.

Si vous entrez

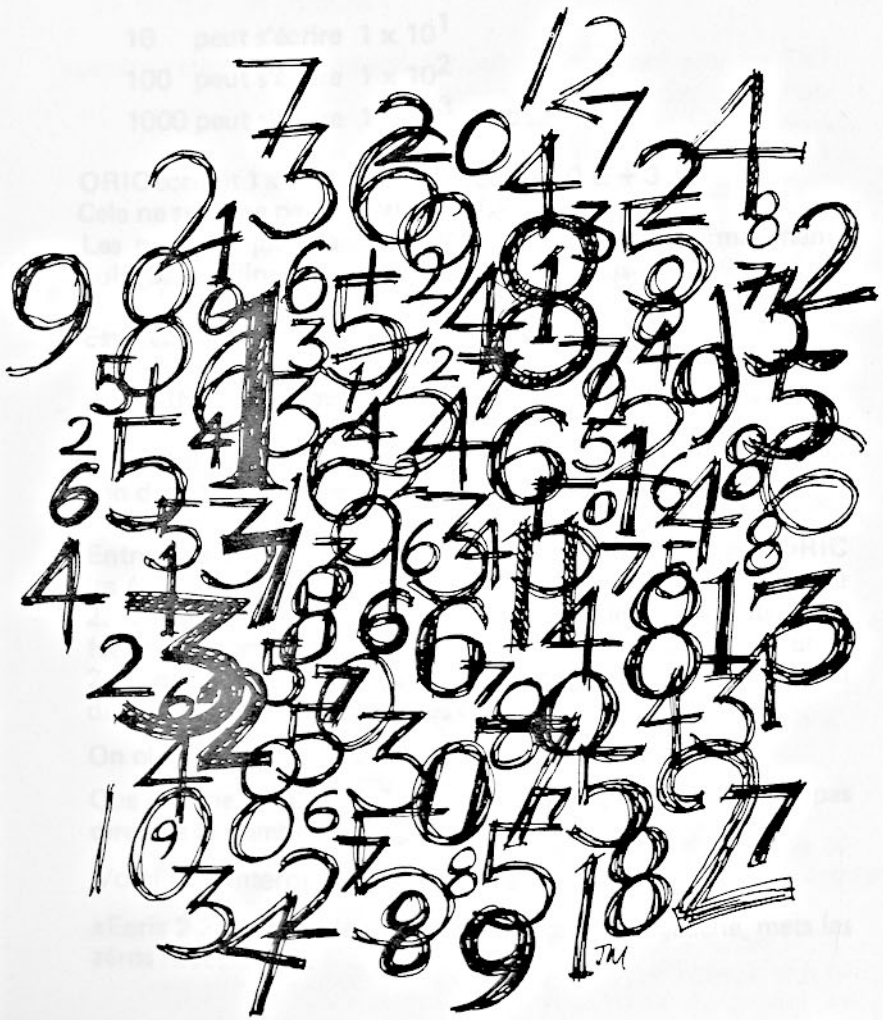
5 TRON et que vous faites RUN, l'écran se remplit de nombres et vous voyez que la ligne 400 n'est jamais atteinte.

Si vous voulez seulement faire une recherche locale, encadrez la zone par 2 lignes, l'une contenant TRON et l'autre TROFF (TRace OFF).

Cette méthode peut être utilisée pour un sous-programme.

Attention TRON et TROFF ne sont pas utilisables en mode direct.

CHAPITRE 6
Jongler avec les Nombres



6. JONGLER AVEC LES NOMBRES

Comme vous l'avez déjà vu, ORIC peut manipuler de très grands nombres et aussi de très petits, aussi bien positifs que négatifs. Plus les nombres sont grands, plus il faut de chiffres pour les écrire. 10 s'écrit avec 2 chiffres, 100 avec 3, et ainsi de suite. Avec des nombres très grands à lire et à écrire, cela peut être difficile à maîtriser. Il existe un moyen pour écrire les grands nombres sous une forme plus compacte), appelée notation scientifique.

10 peut s'écrire 1×10^1
100 peut s'écrire 1×10^2
1000 peut s'écrire 1×10^3 etc ...

ORIC écrirait 1×10^3 ainsi : 1.00000000 E + 3
Cela ne se passe pas toujours ainsi.
Les nombres jusqu'à 999999999 sont affichés normalement, puisque l'ordinateur ORIC donne une précision de 9 chiffres.

Essayez ceci :

```
PRINT 999999999*1    puis  PRINT 9999999999*1
```

Ceci vous montre la notation scientifique et également la façon dont les nombres sont arrondis.

Entrez de très grands nombres et observez comment ORIC les écrit. Vous pouvez aussi entrer des nombres sous la forme 2. 3 E + 4 pour voir leur écriture usuelle équivalente. Une façon de comprendre cette écriture est de se dire : «Ecris 2. 3, déplacę la virgule (le point pour ORIC) de 4 rangs vers la droite, mets des zéros dans les colonnes vides».

On obtient ici 23000

Que signifie 2.3 E − 4 ? Le signe moins après le E ne veut pas dire que le nombre est négatif.

Voici une interprétation possible :

«Ecris 2.3, déplace le point de 4 rangs vers la gauche, mets les zéros nécessaires».

On obtient ici 0.00023

– 2.3 E + 4 et – 2.3 E – 4 correspondent respectivement à – 23 000 et – 0.00023.

Il vous faut connaître ces notations si vous voulez toujours comprendre les résultats fournis par votre ORIC.

xxxxxxxxxx

INT

INT est une fonction qui fournit le plus grand entier inférieur ou égal au nombre entre parenthèses.

Essayez :

```
? INT (1.5)
? INT (2)
? INT (–2)
? INT (–1.5)
```

Attention en particulier au dernier exemple.

Pensez à un thermomètre, si la température n'est pas un nombre entier de degrés on tombe à la valeur entière placée en-dessous.

xxxxxxxxxx

ABS

ABS est une fonction qui fournit la valeur absolue du nombre écrit entre parenthèses.

```
? ABS (4.3)
? ABS (–4.3)
```

SGN

SGN est une fonction qui donne –1 si le nombre est négatif, 0 si le nombre est zéro, et 1 si le nombre est positif.

Essayez :

```
10 FOR N = – 5 TO 5
20 ? N, SGN (N)
30 NEXT
```

xxxxxxxxxx

DATA

En français les données

Données numériques

Si l'on doit faire appel à une série de nombres déterminés dans un programme on peut les écrire après l'instruction DATA, et à n'importe quel endroit du programme.

L'instruction READ (lire) en provoque la lecture dans l'ordre.

Exemple :

```
10  FOR N = 1 TO 5
20  READ A
30  S = S + A
40  NEXT N
50  PRINT "SOMME = " S
60  DATA 1, 3, 8, 6, 4
```

La boucle 10 – 40 se répète 5 fois, en ligne 20 on commande la lecture d'un nombre figurant en ligne 60. Ce nombre est ajouté à 5 (initialement à zéro) et en fin de boucle la somme est affichée en 50.

Quand le programme s'exécute, un pointeur se déplace et retient l'emplacement du dernier nombre lu. Si vous tapez GOTO 10, le pointeur n'est pas remis à son point de départ, et le READ en ligne 20 cherchera au-delà du 4 un nombre inexistant :

Un message d'erreur, signale : IL N'Y A PLUS DE DONNÉES c'est OUT OF DATA. RESTORE (Rétablir) est

une instruction qui remet le pointeur au début de la liste des données (DATA). Ajoutez 15 RESTORE et voyez l'effet produit sur ce programme. Le pointeur est remis à chaque tour en début de liste, si bien que 1 est ajouté à S à chaque fois et le reste des données (en DATA) est ignoré. Pour les *données non numériques*, voir le chapitre des mots.

x x x x x x x x x x

VARIABLES INDICÉES

En mathématiques on écrit A_1, A_2, A_3 . . . A_n et on appelle indice le numéro écrit en bas à droite. Cela permet d'identifier plus facilement une série de nombres. En programmation BASIC on écrit le numéro après la lettre et entre parenthèses. C'est une notation très pratique en programmation, elle permet de gérer de nombreuses variables différentes avec un minimum de lignes de programme.

Exemple :

N(0), N(1), N(2), N(3) sont des éléments du tableau N.

ORIC réserve automatiquement 11 places pour les éléments d'un tel tableau. Si vous voulez dépasser 10 il faut réserver de l'espace en mémoire par l'instruction DIM. Ainsi DIM N(14) autorise 15 éléments (De 0 à 14).

L'utilisation des variables indicées est particulièrement propice dans les boucles FOR/NEXT.

```
10 FOR N= 1 TO 5
20 A (N) = N * N
30 NEXT N
40 FOR X = 1 TO 5
50 PRINT X, A(X)
60 NEXT X
```

La boucle 10—30 calcule les 5 valeurs A(1) à A(5). A(2) par exemple vaut 4, c'est le carré de 2. La boucle 40—60 affiche à l'écran le contenu du tableau. Observez qu'il n'est pas nécessaire de remplir tous les éléments d'un tableau.

A(7) par exemple vaut 0 ainsi que tout élément non affecté.

Un tel tableau est en fait rien de plus qu'une liste de nombres qu'on peut imaginer écrits en colonne aussi bien qu'en ligne (ou rangée).

On peut remplir un tableau à 2 dimensions en tenant compte à la fois des lignes et des colonnes. Un tableau à plusieurs dimensions doit être déclaré par DIM avant utilisation.

```
10 DIMA(5,5)
20 FOR L = 1 TO 5
30 FOR C = 1 TO 5
40 A (L,C) = L * C
50 NEXT C,L
```

Ce programme remplit un tableau de 25 cases. Vous reconnaissez un fragment de table de multiplication. L est le numéro de la ligne, C celui de la colonne. Notez l'abréviation de la commande NEXT en ligne 50, et l'ordre:C d'abord !

C

↗	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	6	8	10
3	0	3	6	9	12	15
4	0	4	8	12	16	20
5	0	5	10	15	20	25

L

Voici le tableau rempli . Demandez ?A(4,3)

La notation à double indice est ici particulièrement adaptée : ainsi A(2,3) vaut 6 et A(5,5) vaut 25.

Si vous souhaitez davantage de dimensions c'est possible. Mais il est important de savoir que DIM N (10, 10, 10) a déjà 11 x 11 x 11 = 1331 éléments, et que la capacité mémoire de l'ORIC risque de ne pas suffire si vous abusez de tels tableaux !

LOGARITHMES

- Logarithme décimal

  10 x 10 x 10 x 10 = 10000 ou 10^4 (dix à la puissance 4, dit-on usuellement). 4 est l'exposant. 4 est le logarithme décimal de 10 000. On écrit log (10 000) = 4. De même log (100) = 2.

  log (1) = 0 et log (10) = 1

  Le log d'un nombre entre 1 et 10 est compris entre 0 et 1.

  Tapez

  ```
  ? LOG(5)
  ```

  ORIC vous indique à quelle puissance (approximativement) le nombre 10 doit être élevé pour obtenir 5.

  ```
  0.698970004
  ```

  Réciproquement, pour vérifier demandons

  ```
  ? 10↑0.698970004
  ```

  et regardons si c'est bien cela. (Le signe ↑ se lit "à la puissance" ou "exposant" et est obtenu avec SHIFT 6 au clavier). On peut retenir que :

  $$10^{\log x} = X$$

- Logarithme à base e

  On appelle ces logarithmes à base 10, logarithmes vulgaires il existe aussi sur ORIC les logarithmes Népériens.

  Tapez

  ```
  PRINT LN(5)
  ```


Vous devez obtenir

```
1.60943791
```

qui est le logarithme népérien de 5.

Quelle différence y-a-t-il avec le logarithme décimal ? Le logarithme népérien de 5 est la puissance à laquelle il faut élever le nombre e pour obtenir 5.

e = 2.718281828 en valeur approchée

e est donné par le calcul de la série :

$$1 + 1 + \frac{1}{2} + \frac{1}{2 \times 3} + \frac{1}{2 \times 3 \times 4} + \ldots\ldots \text{ etc ...}$$

La fonction réciproque de LN (X) est EXP (X), par conséquent :

```
X = EXP(LN(X))
```

Pour trouver le logarithme d'un nombre dans une autre base, voici la formule :

$$LOG_{\text{base z}}(X) = LOG_e(X) / LOG_e(Z)$$

LOG_e est, bien sûr, LN sur ORIC.

. . . . . .

CHANGEMENT DE BASE EN NUMÉRATION

Jusqu'ici nous avons rencontré des nombres en base deux (binaire) et en base dix (décimale). Vous êtes peut être gênés d'apprendre les différentes manières de représenter un nombre. En fait, ce n'est pas si difficile à maîtriser qu'il semble à première vue. Tout provient de la façon de grouper les objets qu'on compte.

La façon la plus courante est de les grouper par dix, probablement parce que nous avons 10 doigts, il ne semble pas y avoir d'autres raisons. Tout notre système de numération utilise des paquets de 10. Avec 10 paquets de 10 on a une centaine, et 10 centaines font mille et ainsi de suite.

Exemple : 3742 c'est 3 mille, 7 cents, 4 fois dix, et 2 unités. Le chiffre le plus grand qu'on puisse avoir dans une colonne est 9. Un de plus et vous avez de quoi former un paquet à inscrire dans la colonne suivante, à gauche.(Influence des arabes). Ainsi :

```
  9
+ 1
---
 10
```

Supposez, juste un instant, que les humains aient 8 doigts seulement (4 à chaque main). Ils utiliseraient les chiffres 0, 1, 2, 3, 4, 5, 6, 7 mais pas 8 ni 9. Ainsi :

```
  7
+ 1
---
 10      pour eux.
```

Le plus grand chiffre possible est 7. Un de plus fait un paquet de huit, ce qui s'écrit 10. Ce qui ne se lit pas "dix" mais "un zéro" en "base huit". Pour marquer la différence avec la numération usuelle on écrit 10_8.

En base dix chaque colonne à gauche vaut dix fois celle de droite.

Milliers	Centaines	Dizaines	Unités
10x10x10	10x10	10	1

De même, en base huit, chaque colonne à gauche vaut huit fois celle de droite.

512	64	8	1
8x8x8	8 x 8	8	1

Ainsi 1245_8 vaut

(1x512) + (2x64) + (4x8) + (5x1) = 677 en base 10

L'ordinateur utilise beaucoup la base deux, les mêmes règles s'appliquent, mais on n'a que deux chiffres 0 et 1 et on fait des paquets de deux.

32	16	8	4	2	1
2x2x2x2x2	2x2x2x2	2x2x2	2x2	2	1

Le nombre binaire 10111 se traduit ainsi :

(1x16) + (0x8) + (1x4) + (1x2) + (1x1) = 23 en base 10

Ceci peut nous conduire à des écritures encombrantes avec des grands nombres. Observons : il faut 8 chiffres pour écrire 255 en base deux et 65535 s'écrit 1111111111111111, avec 16 chiffres !

On pourrait laisser tomber la base deux pour les grands nombres et revenir à la base dix, mais cela ne nous renseigne pas sur la façon dont les nombres sont stockés dans l'ordinateur.

Un compromis est procuré par l'utilisation de la base seize (numération hexadécimale). La base seize nécessite 16 chiffres, aussi au-delà de 9 on emploie les lettres du début de l'alphabet.

Voici les chiffres

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Vous voyez que F vaut 15 en base seize.

Et 10 vaut seize. La valeur des colonnes est donc :

4096	256	16	1
16x16x16	16x16	16	1

Ainsi 12AF en hexadécimal vaut :

(1x4096)+(2x256)+ (10x16)+(15x1) = 4783 en base 16

Pourquoi avoir choisi un système si rébarbatif ?

Peut être l'avez-vous deviné, il permet de voir d'un simple coup d'œil comment les nombres sont mémorisés dans l'ordinateur, en utilisant 2 blocs de 4 bits chacun.

F0 est rangé dans un octet :

F	0
1111	0000

L'énorme nombre binaire correspondant à 65535 en base dix, devient FFFF en hexadécimal (en abrégé hex). Vous devez avoir compris que 65535 est le plus grand nombre qu'il est possible d'adresser en utilisant 2 octets. C'est la raison pour laquelle c'est le numéro de la mémoire la plus haute.

ORIC reconnaît un nombre hexadécimal à la condition qu'il soit précédé du caractère # . Ce caractère correspond à n° (numéro) des claviers français. La prononciation proposée en Angleterre est "h", on a envie de dire "dièze", l'usage n'est pas fixé. Ainsi PRINT # 1A donne 26. Exercez-vous à des conversions hex - déc, puis vérifiez sur ORIC. Vous ne pouvez pas dépasser 4 chiffres, le plus grand est #FFFF. En appendice vous trouverez une table fort commode, vérifiez vos résultats.

NOTA : Dans certains manuels BASIC il est dit que l'identificateur de nombre hexadécimal est $, ce n'est pas valable sur ORIC qui considèrera $A1FB comme une chaîne et non comme un nombre.

PRINT # 10 vous donne 16,

Réciproquement :

PRINT HEX$ (16) vous donne # 10

Voici un programme qui va vous fournir les nombre de 0 à 255 en base dix et à côté leur traduction en base seize.

```
10 FOR N = 0 TO 255
20 ? N, HEX $ (N)
30 NEXT
```

Ralentissez-le avec un WAIT par exemple.

x x x x x x x x x x

MATHÉMATIQUES USUELLES

Bien qu'il soit important de ne pas considérer les ordinateurs comme de simples manipulateurs de nombres, il n'en est pas moins vrai qu'ils se livrent, avec une remarquable aisance, à de nombreuses tâches mathématiques qui, pour vous, sont fastidieuses, surtout quand elles sont répétitives.

ORIC dispose d'une quantité de fonctions mathématiques prêtes à l'emploi.

Si vous voulez la liste des racines carrées des nombres de 1 à 100, il vous faut consulter une table dans un livre ou appuyer sur les touches d'une calculette et cela vous prend du temps.

ORIC va en avoir terminé en un rien de temps.

Le programme est tout simple :

```
10 CLS
```

```
20 FOR N = 1 TO 100
30 ? N, SQR (N)
40 NEXT
```

A l'exécution les nombres de 1 à 100 et leurs racines carrées s'écrivent si vite que vous n'avez pas le temps de les lire ! Ajoutez 35 WAIT 10 pour ralentir le programme, et recommencez ... Cela va mieux ?

SQR est l'abréviation de SQuare Root, qui signifie racine carrée.

ORIC peut même calculer les racines carrées sans utiliser la fonction SQR. Cette méthode de recherche des racines carrées est appelée la méthode itérative de Newton-Raphson. Itérative est l'adjectif correspondant à itération et au verbe itérer (répéter). Une itération consiste à répéter un grand nombre de fois le même processus. C'est une occasion idéale de mettre en œuvre la boucle FOR/NEXT. A chaque passage, le résultat s'affine et se rapproche de plus en plus de la valeur cherchée. Ce court programme montre la méthode ; il s'arrête quand la réponse est suffisamment approchée : en ligne 60 on sort de la boucle si l'écart est inférieur à 0.000001. De toutes façons la valeur exacte de la racine carrée de 5, par exemple, ne peut pas être écrite avec des décimales.

```
5 REM**RACINE CARRÉES PAR ITÉRATION**
10 INPUT "INDIQUER UN NOMBRE" ; S
20 ? : INPUT "DONNER UNE ESTIMATION DE SA RA-
     CINE "; G
30 G = ABS(G) : ? G
40 G = (S/G + G) / 2
50 R = G*G
60 IF R < (S+0.000001) AND R > (S–0.00001) GOTO 80
70 GOTO 30
80 ? "RACINE CARRÉE DE "; S;" = "; G" ENVIRON"
90 END
```

x x x x x x x x x x

Voici un autre exemple d'utilisation de la rapidité de calcul de l'ORIC. Le mathématicien Leibnitz, au 17ème siècle, a découvert une façon de calculer π .

π est un nombre irrationnel ; en d'autres termes il ne peut pas être écrit avec un nombre déterminé (fini) de décimales.

Leibnitz a trouvé que la suite que voici approche de plus en plus la valeur de π .

$$\pi \approx 4\,(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \ldots\ldots\ldots)$$

Nous voyons une progression régulière au dénominateur des fractions. ORIC aime les progressions, car on peut les traiter dans des boucles. Si vous aviez à utiliser la formule ci-dessus avec du papier et un crayon, cela vous prendrait beaucoup de temps, même pour n'aller que jusqu'à 1/9. Le pire, c'est de s'apercevoir que même après des centaines de boucles on n'est encore bien loin du compte ! Pauvre Leibnitz, mais quel bonheur pour nous !

Essayez ce programme :

```
 5 REM **CALCUL LENT DE PI**
10 CLS
20 DEF FNA (N) = (–1/N + 1/(N + 2))
30 FOR X = 3 TO 10003 STEP 4
40 S = S + FNA (X)
50 APPROX = 4* (1 + S)
60 ? APPROX
70 NEXT X
```

La ligne 20 définit une fonction A, contenant la variable N. Cela nous évite une ligne surchargée un peu plus loin. FNA calcule chaque terme de la série et est appelée ligne 40. Le pas de la boucle est 4 et commence à 3, ainsi X prend successivement les valeurs 3, 7, 11, 15, etc ... De la sorte, le résultat calculé ligne 50 est le bon et il s'affiche en ligne 60. A l'exécution de ce programme, ORIC va écrire un résultat qui va progressivement s'approcher de la valeur vraie de PI. Comparez à 3.1416 et vous voyez que ce n'est pas la façon la plus rapide de calculer PI, même avec ORIC.

Pour avoir PI bien plus vite, tapez simplement :

```
PRINT PI
```

Ceci vous donne une valeur approchée de π avec pas mal de décimales, c'est que π est en mémoire permanente.

Attention à ne pas choisir PI comme nom de variable, ni même aucun mot commençant par PI comme PIRE ou PIPE et tenter PIPE = 12 ou PIRE = 0.7 : PI est un mot réservé.

NOMBRES TIRÉS AU HASARD

ORIC possède une fonction très utile, qui sert souvent pour les programmes de jeux. C'est RND, qui fournit un nombre tiré au hasard ; enfin presque, car il s'agit d'une simulation calculée. On parle de pseudo-hasard. Il existe une méthode sous-jacente de génération. Pour s'en rendre vraiment compte il faudrait se livrer à une étude approfondie des séries. Aussi RND peut être considéré pour l'usage qu'on en fait ici comme un vrai tirage aléatoire.

Si vous ne savez pas bien ce que sont les nombres tirés au hasard, observez un dé à jouer. Il y a autant de chance de sortir 1 que 2 ou 3 ou 4 ou 5 ou 6. L'ordre de succession des nombres de 1 à 6 lors des jets successifs est tout à fait aléatoire.

Voici une simulation sur ORIC :

```
 5 REM *** JET DE DÉ ***
10 FO N = 1 TO 10
20 ? "APPUYER SUR UNE TOUCHE POUR JETER LE
   DÉ"
30 GET A$
40 A = INT (RND (1) * 6) + 1
50 ? A,
60 NEXT
```

L'instruction GET attend qu'on enfonce une touche au clavier. En ligne 40 RND (1) génère un nombre de 0 inclus à 1 exclus. Ce nombre est multiplié par 6. On en prend la partie entière. On ajoute 1. En fin de compte on obtient n'importe quel nombre de 1 à 6. La virgule décale de 5 cases.

RND (n) donne toujours un nombre de 0 inclus à 1 exclus, si

n est positif. Si n est négatif, alors le générateur de nombres aléatoires commence une série particulière et les "n" positifs qui suivront produiront toujours la même série. Si n vaut zéro, c'est le dernier nombre généré qui est fourni.

x x x x x

Pour conclure ce chapitre, voici un programme qui utilise un grand nombre de fonctions que l'on vient d'étudier. Il utilise aussi quelques procédures de manipulation des chaînes que vous ne comprendrez pas vraiment avant d'avoir étudié le chapitre 8. Au besoin , revenez examiner ce programme plus tard.

Les calendriers sont difficiles à concevoir, car la terre nous joue le mauvais tour de ne pas accomplir un nombre entier de rotations en une révolution autour du soleil. En fait, il faut 365.242216 jours pour faire exactement un an.

Diverses personnes, de Numa Pomilius et Jules César jusqu'au Pape Grégoire 1er, ont tenté de mettre au point le calendrier, mais il reste imparfait.

Tout cela fait que, savoir le jour correspondant à une date donnée, est relativement compliqué à calculer.

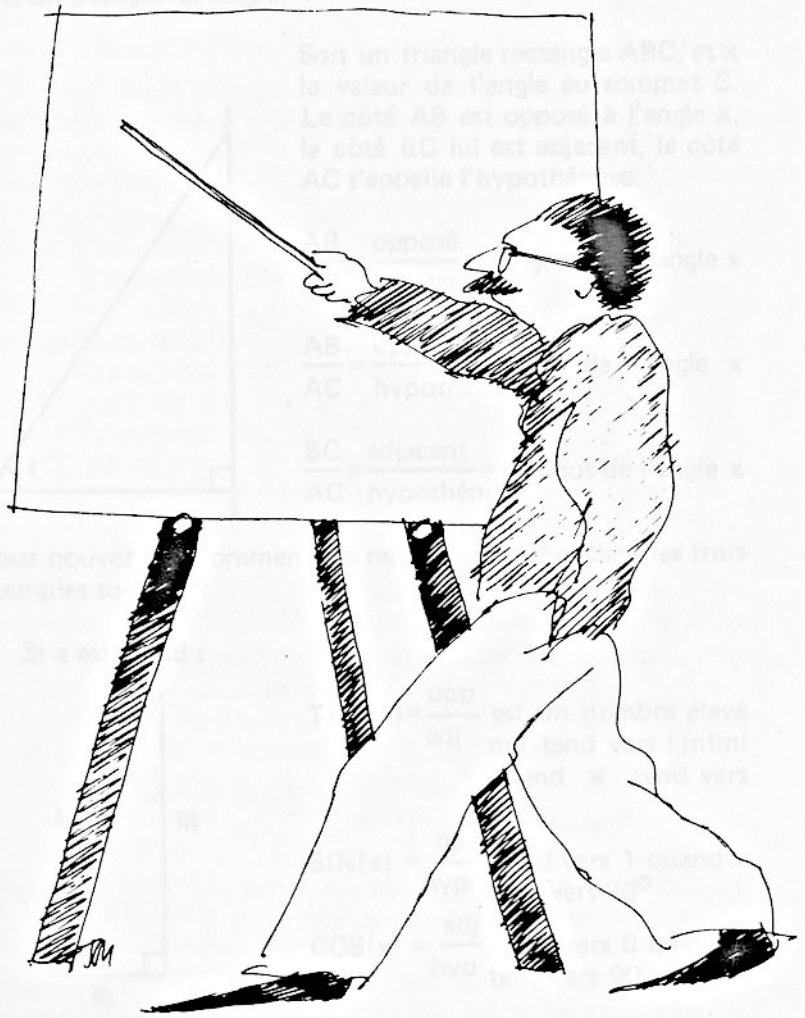
Le mathématicien allemand Gauss, a mis au point une formule qui convient pour toute date postérieure à 1752, début du calendrier Grégorien.

```
  5 REM *** CALCUL DU JOUR SELON LA DATE***
 10 CLS
 20 ? "ENTRER JOUR: MOIS, AN"
 30 INPUT "JOUR" ; J
 40 IF J< 1 OR  J >31 THEN 30
 50 ? : INPUT "MOIS" ; M
 60 IF M< 1  OR  M >12 THEN 50
 70 ? : INPUT "AN" ; A
 80 IF A <1752  OR  A> 8000 THEN 70
 90 M = M – 2 : IF M< 1 THEN M = M + 12 : A = A – 1
100 A$ = STR $ (A)
```

```
110 C = INT (A/100)
120 A = VAL (RIGHT$ (A$, 2))
130 B = INT (2.6* M − 0.19) + J + A + INT (A/4) + INT
    (C/4) − C * 2
140 JOUR = INT((B/7 − INT (B/7))* 7 + 0.1)
150 JOUR = JOUR + 1
160 FOR N = 1 TO JOUR
170 READ JOUR$
180 NEXT N
190 PRINT JOUR$
200 DATA DIMANCHE, LUNDI, MARDI, MERCREDI,
    JEUDI, VENDREDI, SAMEDI
```

A la ligne 30, le jour demandé est le "quantième", c'est un nombre de 1 à 31.

CHAPITRE 7
Davantage de Fonctions Mathématiques

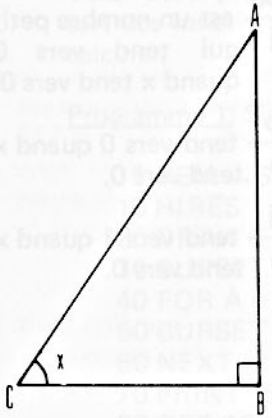


7. DAVANTAGE DE FONCTIONS MATHÉMATIQUES

Trigonométrie

ORIC dispose de nombreuses fonctions que vous reconnaîtrez si vous avez une machine à calculer scientifique ou si vous n'avez pas oublié votre cours de mathématique. Ce sont les fonctions SIN (sinus), COS (cosinus), TAN (tangente).

Elles représentent des rapports entre les longueurs des côtés dans un triangle rectangle.



Soit un triangle rectangle ABC, et x la valeur de l'angle au sommet C. Le côté AB est opposé à l'angle x, le côté BC lui est adjacent, le côté AC s'appelle l'hypothénuse.

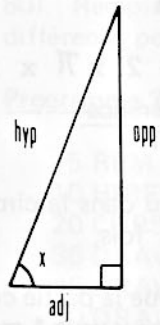
$\frac{AB}{BC} = \frac{\text{opposé}}{\text{adjacent}}$ = tangente de l'angle x

$\frac{AB}{AC} = \frac{\text{opposé}}{\text{hypothénuse}}$ = sinus de l'angle x

$\frac{BC}{AC} = \frac{\text{adjacent}}{\text{hypothénuse}}$ = cosinus de l'angle x

Vous pouvez voir comment ces rapports évoluent dans les trois exemples suivants :

1. Si x est grand :

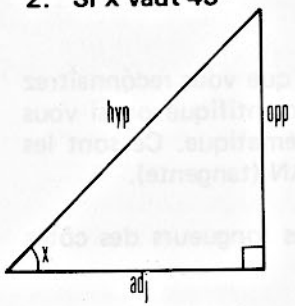


$TAN(x) = \frac{opp}{adj}$ est un nombre élevé qui tend vers l'infini quand x tend vers 90^o

$SIN(x) = \frac{opp}{hyp}$ tend vers 1 quand x tend vers 90^o

$COS(x) = \frac{adj}{hyp}$ tend vers 0 quand x tend vers 90^o

2. Si x vaut 45°

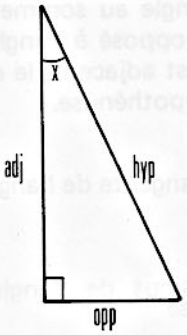


$$TANX(x) = \frac{opp}{adj} = 1$$

$$SIN(x) = \frac{opp}{hyp} = \frac{1}{V2}$$

$$COS(x) = \frac{adj}{hyp} = \frac{1}{V2}$$

3. Si x est petit

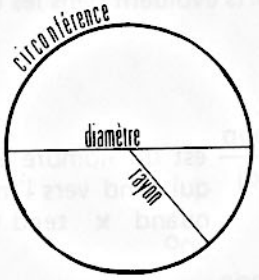


$TAN(x) = \frac{opp}{adj}$ est un nombre petit qui tend vers 0 quand x tend vers 0.

$SIN(x) = \frac{opp}{hyp}$ tend vers 0 quand x tend vers 0.

$COS(x) = \frac{adj}{hyp}$ tend vers 1 quand x tend vers 0.

Quelques rappels sur le cercle



diamètre = 2 x rayon

$$\pi = \frac{\text{circonférence}}{\text{diamètre}}$$

$$\pi = \frac{\text{circonférence}}{2 \times \text{rayon}}$$

Circonférence = $2 \times \pi \times$ rayon

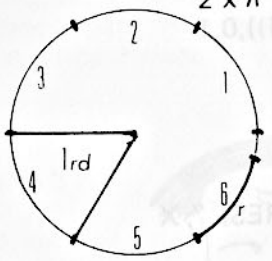
$$\text{rayon} = \frac{\text{circonférence}}{2 \times \pi}$$

Combien de fois le rayon est-il contenu dans la circonférence ? Plus de 6 fois : exactement $2 \times \pi$ fois.

Si l'un coupe un morceau du cercle tel que la partie courbe soit égale au rayon, alors l'angle au centre x vaut 1 radian ;

le cercle complet valant 360°, on trouve

$$1 \text{ radian} = \frac{360}{2 \times \pi} = 57.29578^{o} \text{ environ}$$



Les formules de conversion sont les suivantes :

degrés = radians x 57.29578

$$\text{radians} = \frac{\text{degrés}}{57.29578}$$

Voici deux programmes qui montrent comment ORIC se sert des fonctions trigonométriques pour dessiner ou pour calculer.

Programme 1: Sinusoïde

```
 5 REM **SINUS**
10 HIRES
20 DRAW 0,199,1
30 CURSET 0,100,3 : DRAW 239,0,1
40 FOR A = - PI TO PI STEP 0.02
50 CURSET A*38 + 120,SIN(A)*99+99,1
60 NEXT
70 PRINT "SINUSOIDE"
80 GET A$
```

Ceci trace le graphique d'une fonction sinus de $-\pi$ à π. La ligne 70 imprime sur la zone réservée au texte et met en attente jusqu'à ce qu'une touche soit enfoncée (ligne 80). Remplacez SIN par COS en ligne 50 et voyez la différence pendant qu'ORIC "trace une fonction" cosinus.

Programme 2: Hauteur d'une tour

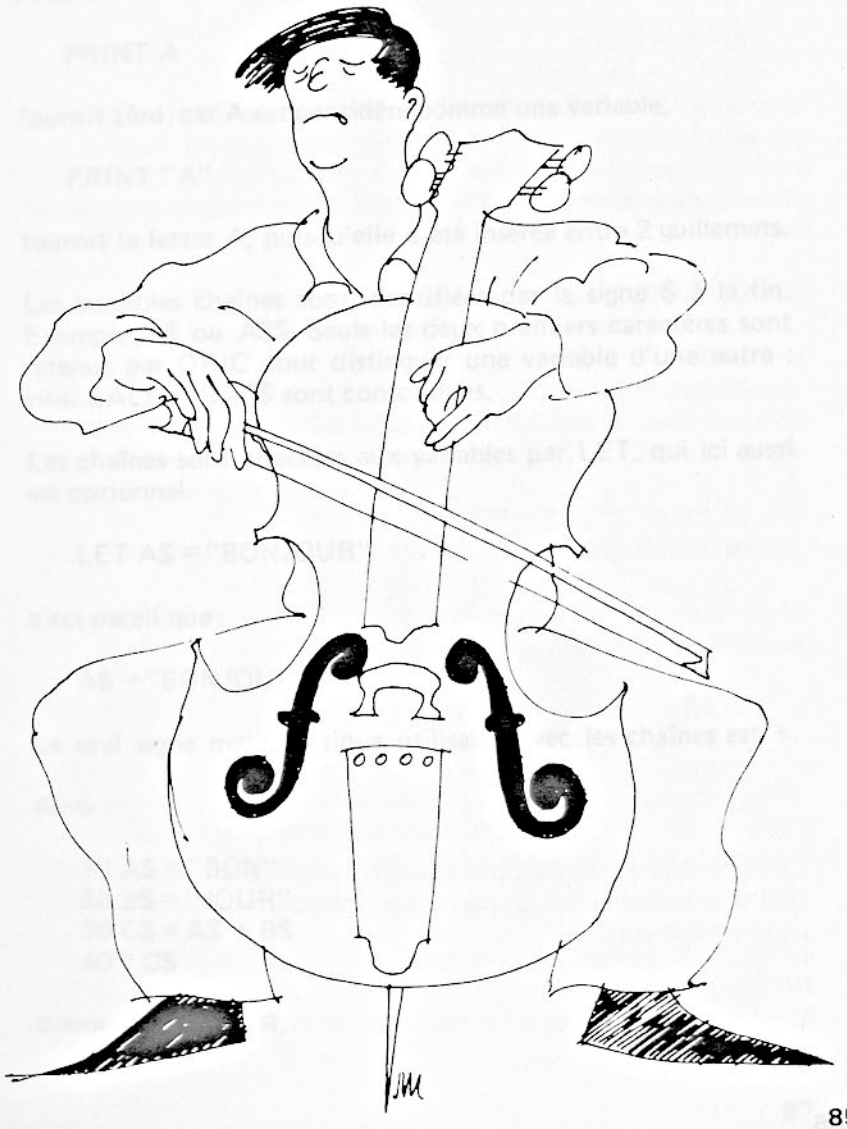
```
 5 REM ***TOUR***
10 HIRES
20 CURSET 20,20,3
30 DRAW 0,160,1
40 DRAW 200,0,1
50 DRAW - 200, - 160,1
```

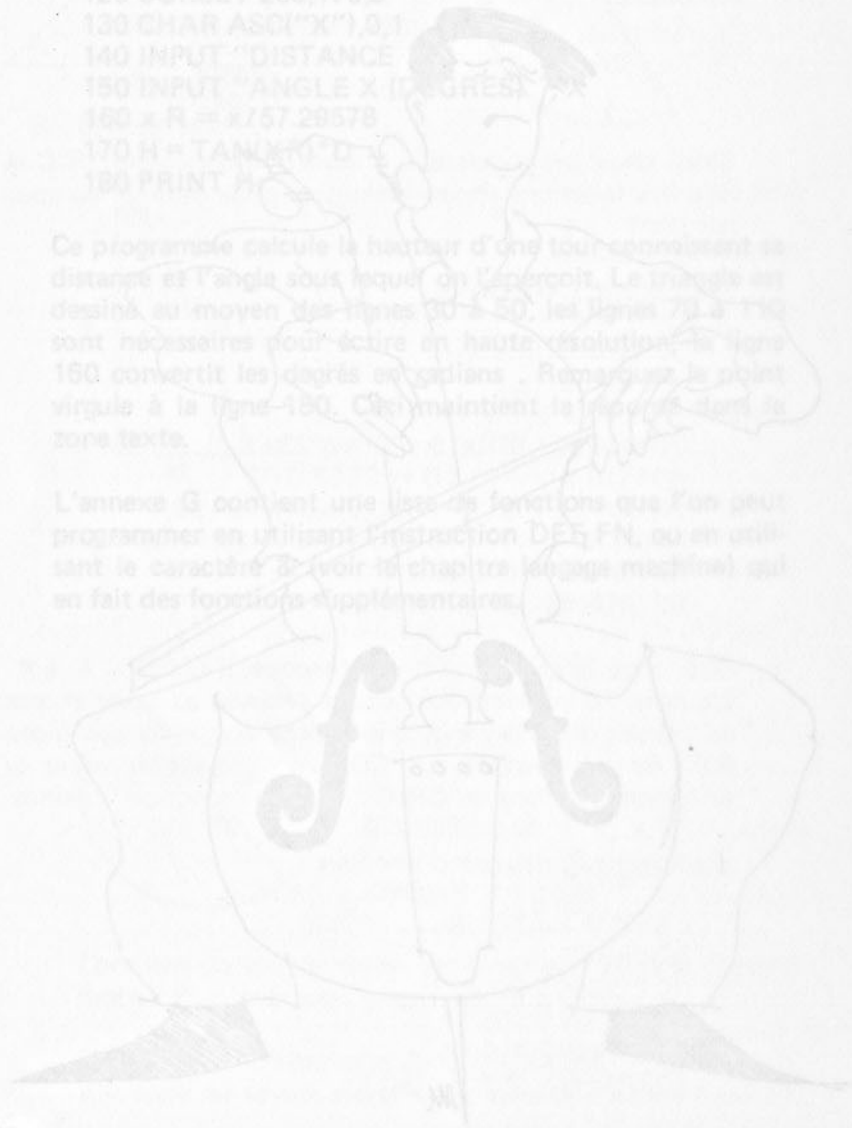
```
 60 CURSET 25,170,3
 70 A$ = "Hauteur de la tour"
 80 FOR N = 1 TO LEN (A$)
 90 CHAR ASC (MID$(A$,N,1)),0,1
100 CURMOV 8,0,3
110 NEXT
120 CURSET 200,170,3
130 CHAR ASC("X"),0,1
140 INPUT "DISTANCE "; D
150 INPUT "ANGLE X (DEGRES) "; X
160 x R = x/57.29578
170 H = TAN(XR)*D
180 PRINT H;
```

Ce programme calcule la hauteur d'une tour connaissant sa distance et l'angle sous lequel on l'aperçoit. Le triangle est dessiné au moyen des lignes 30 à 50, les lignes 70 à 110 sont nécessaires pour écrire en haute résolution, la ligne 160 convertit les degrés en radians . Remarquez le point virgule à la ligne 180. Ceci maintient la réponse dans la zone texte.

L'annexe G contient une liste de fonctions que l'on peut programmer en utilisant l'instruction DEF FN, ou en utilisant le caractère & (voir le chapitre langage machine) qui en fait des fonctions supplémentaires.

CHAPITRE 8
Des Mots





8. DES MOTS

Nous avons déjà vu que les ordinateurs peuvent manipuler n'importe quels signes, pas seulement des nombres. Afin qu'ORIC distingue les caractères des nombres, et qu'il ne les considère pas comme d'éventuels noms de variables, il faut les écrire entre guillemets.

Ainsi :

```
PRINT A
```

fournit zéro, car A est considéré comme une variable.

```
PRINT "A"
```

fournit la lettre A, puisqu'elle a été insérée entre 2 guillemets.

Les variables chaînes sont identifiées par le signe $ à la fin. Exemple A$ ou A3$. Seuls les deux premiers caractères sont retenus par ORIC pour distinguer une variable d'une autre : ainsi BAL$ et BAR$ sont confondues.

Les chaînes sont affectées aux variables par LET, qui ici aussi est optionnel.

```
LET A$ = "BONJOUR"
```

c'est pareil que

```
A$ = "BONJOUR"
```

Le seul signe mathématique utilisable avec les chaînes est +.

Ainsi :

```
10 A$ = "BON"
20 B$ = "JOUR"
30 C$ = A$ + B$
40 ? C$
```

donnera BONJOUR.

La longueur maximum d'une chaîne est limitée à 255 caractères.

Pour connaître la longueur d'une chaîne on dispose de l'instruction LEN.

Exemple :

```
10 INPUT A$
20 L = LEN (A$)
30 ? A$ " CONTIENT "L" CARACTERES."
40 GOTO 10
```

Une chaîne ne peut pas être traitée dans un calcul comme un nombre.

Il faut distinguer :

```
10 A = 7 : B = 5
20 ? A + B
```

de

```
10 A$ = "7" : B$ = "5"
20 ? A$ + B$
```

Mais il existe un moyen de passer d'une chaîne à sa valeur numérique : l'instruction VAL

```
10 A$ = "56"
20 A = VAL (A$)
30 ? A
40 ? 2*A
```

La ligne 40 ne pourrait pas s'écrire 2*A$. Si le premier caractère d'une chaîne est un caractère alphabétique, alors la valeur de la chaîne est 0.

```
10 A$ = "ORIC"
20 V = VAL (A$)
30 ?V
```


La fonction réciproque de VAL est STR$.

STR$ transforme une variable numérique en une variable chaîne.

```
10 A = 128
20 A$ = STR$(A)
30 ?A
40 ?A$
```

On ne voit pas la différence entre l'effet de la ligne 30 et celui de la ligne 40. Cependant PRINT A + A donne 256 tandis que ?A$ + A$ donne 128128 car le premier signe + est l'addition des nombres, tandis que le deuxième est la concaténation des chaînes de caractère.

A la fin du livre vous avez la table des codes ASCII. On en parle aussi au chapitre 4.

La fonction ASC fournit le numéro de code de n'importe quel caractère du clavier. La fonction CHR$ fait le contraire ; elle fait correspondre à chaque nombre de 32 à 128 le caractère dont c'est le code.

Pour en obtenir la liste complète, exécutons ce programme :

```
10 FOR N = 32 TO 128
20 PRINT N " EST LE CODE ASCII DE " CHR$(N)
30 WAIT 20
40 NEXT
```

Comme tous les caractères ont un code ASCII, on peut les mettre en ordre. Comme vous le voyez, les lettres qui se suivent dans l'alphabet ont des numéros de code qui se suivent. Z correspond à 90, et A à 65. 90 est supérieur à 65. Z est après A.

Les signes (<) et (>) qui servent à comparer les nombres, peuvent de ce fait être utilisés pour comparer les chaînes. Cependant, il faut faire attention à ne pas mélanger minuscules et majuscules.

Bien que "arbre" précède "zèbre", "arbre" suit "Zèbre".

Pour vous aider à manipuler les chaînes voici 3 fonctions fort utiles : RIGHT$, LEFT$, MID$.

- RIGHT$ fournit la partie droite d'une chaîne.

```
A$ = "ABCDEFGHIJ"
PRINT RIGHT$ (A$,2)
```

donnè IJ. Le nombre de caractères à extraire de la chaîne A$ est écrit après la virgule.

- LEFT$ fournit la partie gauche d'une chaîne

```
PRINT LEFT$ (AS, 3)
```

donne ABC

- MID$ fournit une partie de la chaîne extraite n'importe où : il faut indiquer le rang du premier caractère extrait et le nombre de caractères désirés.

Ainsi :

```
PRINT MID$ (A$, 5, 2)
```

donne EF.

Si l'on ne donne pas le 2ème nombre, la totalité des caractères situés à droite est fournie, à partir du rang indiqué.

```
PRINT MID$ (AS, 5)
```

donne EFGHIJ

Voici un court programme d'utilisation de ces fonctions :

```
10 ? "ENTRER UN TEXTE"
20 INPUT A$
30 IF LEN (A$) < 3 THEN ? "TROP COURT" : GOTO
   10
40 ?A$ " CONTIENT "LEN(A$)" CARACTERES."
```



```
50 ? "COMMENCE PAR " LEFT$(A$,1)
60 ? "ET SE TERMINE PAR " RIGHT$(A$,1)
70 ? "AVEC " MID$ (A$, 2, LEN (A$) — 2) " AU MI-
   LIEU"
```

xxxxxxxx

Il est souvent plus pratique de disposer de données dans le programme lui-même. L'instruction DATA, lue par READ est utilisable pour les mots, les chaînes comme pour les nombres.

```
10 FOR X = 0 TO 3
20 READ NOM $ (X)
30 ? NOM $ (X)
40 NEXT
50 DATA  DENIS, MARTIN, DURAND, DUPONT
```

A chaque passage dans la boucle FOR/NEXT un nom est mis dans une variable indicée. DENIS est affecté à NOM$ (0), MARTIN à NOM$(1), DURAND à NOM$ (2) et DUPONT à NOM$ (3).

Ces listes de variables chaînes indicées, (ces tableaux), sont semblables à celles de variables numériques. ORIC réserve automatiquement la place pour les éléments de l'indice 0 à l'indice 10, ce qui fait 11 places. Si vous en désirez davantage, vous devez déclarer la dimension souhaitée avec l'instruction DIM..Exemple DIM A$ (19) réservera 20 places pour des chaînes A$ (0) à A$ (19) de chacune 255 caractères au plus.

A noter une particularité concernant les chaînes inscrites en DATA. Si vous laissez des espaces au début ou à la fin d'une chaîne, ils seront ignorés par ORIC.

```
DATA, AB,  C, DE
```

perdra l'espace devant le C. Si vous voulez que ce "blanc" soit dans la chaîne il faut mettre des guillemets ainsi :

```
DATA AB," C", DE
```

Quand vous commencez l'exécution avec RUN, le pointeur de

données se met au premier mot de la liste en DATA. S'il y a plusieurs lignes comportant l'instruction DATA, le pointeur ira à la première ligne d'abord. La lecture se fait ensuite par l'instruction READ, pas à pas, à partir de là.

Le programme exécuté, le pointeur reste placé en fin de liste. Si l'on commande GOTO 10 en mode direct, le pointeur n'indiquant plus rien, vous recevez un message d'erreur :

```
OUT OF DATA IN 20
```

C'est l'instruction RESTORE qui envoie le pointeur en début de liste. Ajouter

```
35 RESTORE
```

Exécuter ce nouveau programme. Faites GOTO 10, vous n'avez plus le message d'erreur, mais vous obtenez 4 fois le même nom, car à chaque passage le pointeur est remis au début.

xxxxxxxx

TRI

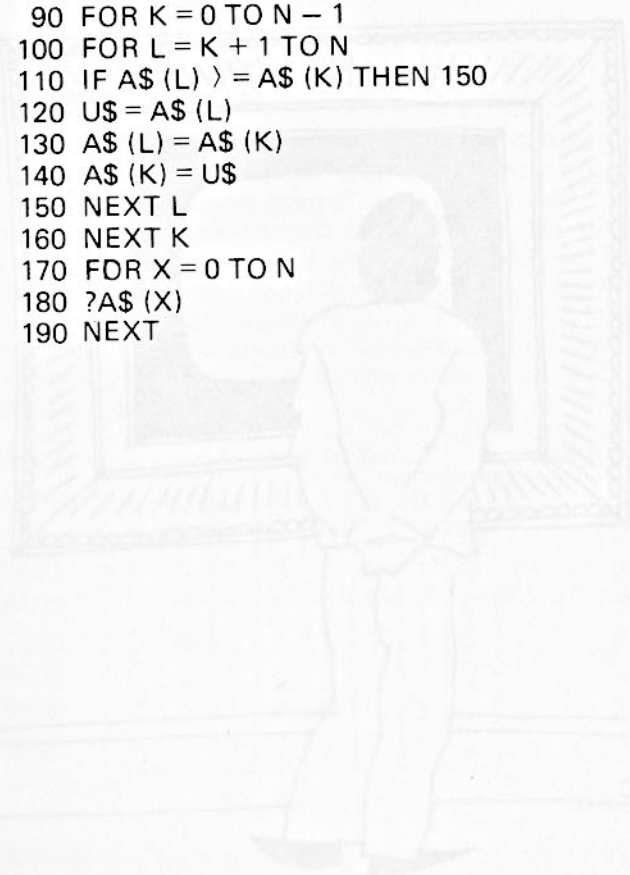
Pour compléter ce chapitre, voici un programme qui illustre certaines des techniques de manipulation de chaînes par l'ORIC, spécialement les tableaux et la mise en ordre "alphabétique". (On dit aussi ordre lexicographique).

Il existe un grand nombre de méthodes de tri utilisables sur ordinateur. Tous utilisent le principe de correspondance de chaque caractère avec son code ASCII. En fait l'ordinateur met en ordre les nombres du code et restitue ensuite les caractères correspondants qui se trouvent mis en ordre alphabétique. Voyez l'appendice D.

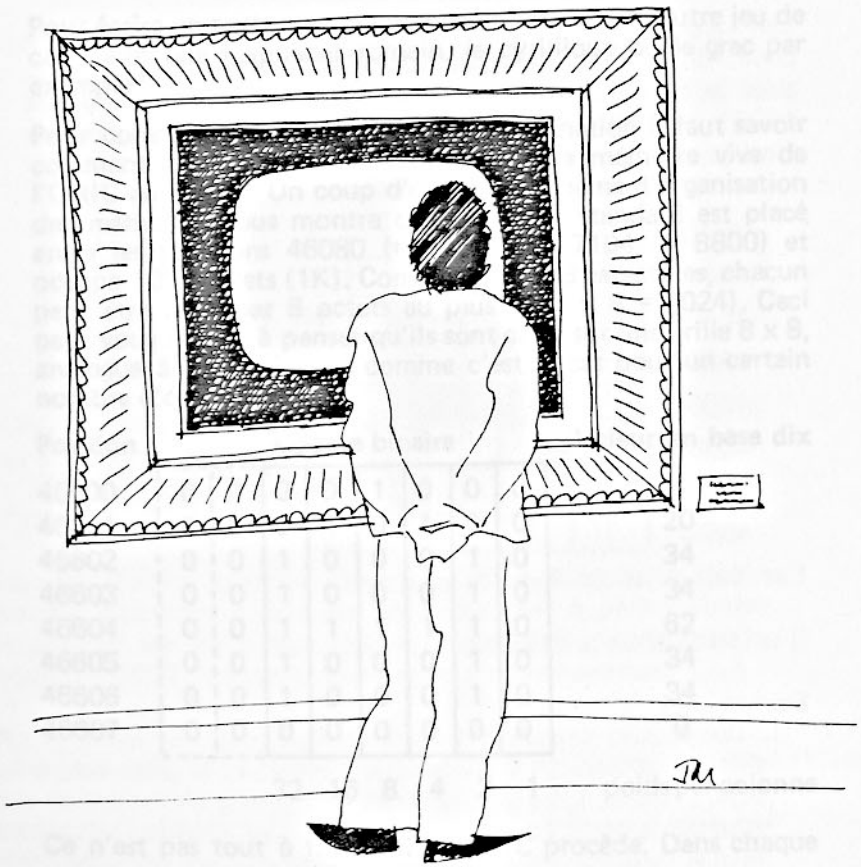
Ce programme donné à titre d'exemple, introduit les mots à mettre en ordre à partir du clavier dans un tableau A$. Il utilise la variable U$ pour retenir provisoirement des mots qui sont échangés. A$ est brassé jusqu'à ce que les mots soient rangés, ils sont alors écrits à l'écran (lignes 170 - 196).

Vous pouvez conserver ce programme en le numérotant à partir de, disons 2000. Il vous servira ultérieurement de sous-programme.

```
  5 REM ***TRI***
 10 INPUT "NOMBRE DE MOTS "; N
 20 DIM A$ (N + 1)
 30 FOR X = 0 TO N – 1
 40 INPUT A$ (X)
 50 NEXT
 60 FOR X = 0 TO N- 1
 70 ?A$ (X)
 80 NEXT
 90 FOR K = 0 TO N – 1
100 FOR L = K + 1 TO N
110 IF A$ (L) > = A$ (K) THEN 150
120 U$ = A$ (L)
130 A$ (L) = A$ (K)
140 A$ (K) = U$
150 NEXT L
160 NEXT K
170 FOR X = 0 TO N
180 ?A$ (X)
190 NEXT
```



CHAPITRE 9
Graphismes Élaborés



9. GRAPHISMES ÉLABORÉS

CARACTERES GRAPHIQUES DÉFINIS PAR L'UTILISATEUR

Lorsque vous allumez ORIC, les 2 jeux de caractères, le standard et un deuxième en réserve, sont chargés dans la mémoire vive. Le clavier standard est celui qui correspond aux caractères ASCII comme il est indiqué en appendice, et le deuxième clavier contient des signes graphiques. L'un aussi bien que l'autre peut être complètement ou partiellement reconfiguré.

Pour un jeu, vous pouvez souhaiter utiliser à la fois du texte et quelques caractères graphiques que vous avez définis vous-même (petits ALIENS, peut être ?), vous pouvez alors choisir un caractère du premier clavier qui sert peu souvent comme @ (a commercial) ou © (copyright) et le redéfinir.

Pour écrire un texte vous pouvez avoir besoin d'un autre jeu de caractères que l'alphabet romain, le cyrillique ou le grec par exemple.

Pour comprendre la méthode de transformation il faut savoir comment un caractère est stocké dans la mémoire vive de l'ORIC au départ. Un coup d'oeil sur le schéma d'organisation des mémoires nous montre que le clavier standard est placé entre les positions 46080 (# B400) et 47104 (# B800) et occupe 1024 octets (1K). Comme il y a 128 caractères, chacun peut être codé par 8 octets au plus (128 x 8 = **1024**). Ceci peut vous inciter à penser qu'ils sont créés sur une grille 8 x 8, analogue à un échiquier, comme c'est le cas pour un certain nombre d'ordinateurs.

Position	Codage binaire								Valeur en base dix
46600	0	0	0	0	1	0	0	0	8
46601	0	0	0	1	0	1	0	0	20
46602	0	0	1	0	0	0	1	0	34
46603	0	0	1	0	0	0	1	0	34
46604	0	0	1	1	1	1	1	0	62
46605	0	0	1	0	0	0	1	0	34
46606	0	0	1	0	0	0	1	0	34
46607	0	0	0	0	0	0	0	0	0
			32	16	8	4	2	1	poids par colonne

Ce n'est pas tout à fait ainsi qu'ORIC procède. Dans chaque

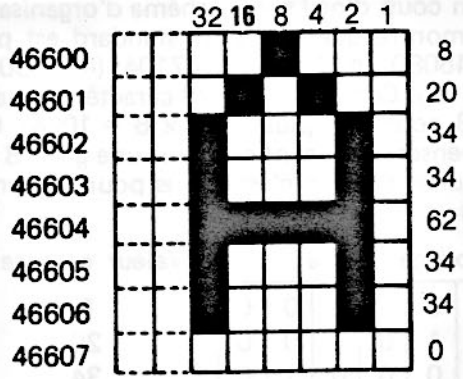
octet, seuls les 6 bits de droite sont utilisés pour le dessin d'un caractère.

Chaque caractère peut donc être imaginé dans une matrice de 8 rangées de 6 points. Chaque rangée est un octet dont on n'utilise que les 6 bits de droite. Un case contenant 1 est "active", une case contenant 0 est "inerte". (ON/OFF disent les anglais).

Les cases "actives" sont de la couleur de l'avant plan, les cases "inertes" sont de la couleur de l'arrière plan.

Pour savoir le contenu d'une position mémoire, nous utilisons PEEK. Pour en modifier le contenu, nous nous servons de POKE.

Les caractères d'un clavier sont rangés dans l'ordre de leur code ASCII. Le code ASCII de la lettre A majuscule est 65, aussi le dessin de la lettre A sera placé à partir de l'adresse 46080 + (65*8), c'est à dire 46600, et sur 8 octets comme l'indique la figure suivante :



Les zéros et les uns forment la lettre A.

Il est très aisé de la changer.

Essayez ceci :

```
POKE   46600, 31
POKE   46601, 21
POKE   46602, 31
POKE   46603, 4
```



```
POKE   46604, 31
POKE   46605, 4
POKE   46606, 10
POKE   46607, 17
```

Appuyez sur la touche A au clavier maintenant, vous voilà envahi par ORIC ! Chaque A va maintenant apparaître sous cette nouvelle forme, puisque vous avez redéfini la façon de dessiner un A pour ORIC.

Cela prendrait très longtemps d'utiliser POKE pour changer chacune des 8 lignes de chaque caractère que l'on désire modifier. Aussi est-il préférable d'écrire un court programme de redéfinition de caractères.

Voici un programme effarant qui redéfinit tout le clavier :

```
 5 REM***LES ENVAHISSEURS***
10 FOR X=46344 TO 47088 STEP 8
20 FOR I=0 TO 7
30 READ M
40 POKE I+X, M
50 NEXT I
60 RESTORE
70 NEXT X
80 DATA 18, 12, 30, 45, 45, 30, 18, 0
```

Après l'avoir introduit, faire LIST pour vérifier, puis RUN, et que voyez-vous sur l'écran ?

Vous voilà sous la domination totale des ALIENS !

Essayez d'écrire quelque chose ..... Pour vous sortir d'une pareille situation, il n'existe qu'un moyen, à part éteindre et allumer (ce qu'il vaut mieux éviter) c'est d'actionner le bouton RESET. Vous retrouvez alors le clavier normal.

Après cette fantaisie, passons à un programme plus sérieux, bien plus utile. Il va pouvoir vous servir à redéfinir la totalité des caractères du clavier. Un tableau X à une dimension est créé pour contenir les nouveaux caractères. C$ est la variable d'introduction du caractère quelconque que vous aurez choisi.

A est l'adresse de départ du premier clavier en mémoire, D est la position mémoire du premier octet du caractère choisi. Le sous-programme qui commence à la ligne 1000 va lire (par PEEK) chaque octet du caractère et transformer sa valeur décimale en sa valeur binaire, par exemple 45 sera converti en 00101101 , et chaque chiffre zéro ou un sera mis dans le tableau X.

La fin du sous-programme envoie en mémoire écran soit un gros pavé (128) si la case vaut 1, soit un espace (32) si la case vaut 0. Ainsi un agrandissement du caractère apparaît sur l'écran. Le reste du programme permet à l'utilisateur d'entrer ses données, une ligne à la fois, et d'observer à tout instant l'effet obtenu en grand et en taille normale.

```
  5 REM***GÉNÉRATEUR DE CARACTERES***
 10 CLS
 20 PRINT
 30 ? "ENTRER LE CARACTERE QUE VOUS VOULEZ",
    "REDÉFINIR"
 40 GET C$
 50 PRINT C$
 60 C = ASC(C$)
 70 A = 46080:D=C*8
 80 GOSUB 1000
 90 PRINT "ENTRER LES VALEURS DES OCTETS"
100 FOR N = 0 TO 7
110 PRINT "LIGNE NUMÉRO " ; N+1
120 INPUT X(N)
130 IF X (N)› 63 OR X (N)‹ 0 THEN 120
140 POKE  A + D + N , X(N)
150 NEXT
160 GOSUB 1000
200 STOP
1000 REM  ***S/P DE GÉNÉRATION DE CARAC-
     TERES***
1010 FOR N = 0 TO 7
1020 X(N) = PEEK(A + D + N) : NEXT N
1030 FOR N = 0 TO 7 : Z = 64 : M = 1
1040 NB = X(N) – Z
1050 IF NB‹0 GOTO 1090
1060 X(N) = X(N) – Z
```



```
1070 POKE 48 220 + (N*40) + M, 128
1080 M = M+1 : Z = Z/2 : IF M›8 GOTO 1100 ELSE 1040
1090 POKE 48 220 + (N*40) + M, 32
1095 GOTO 1080
1100 NEXT N  : RETURN
```

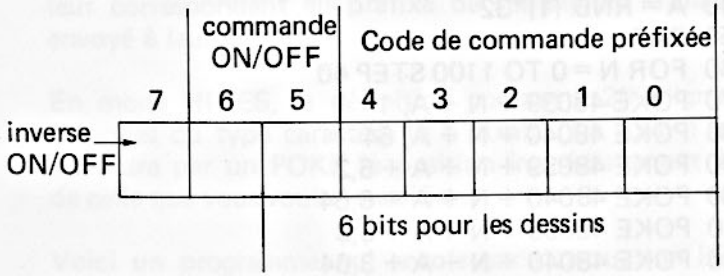
COMMANDES PRÉFIXÉES

Pour envoyer des indications à l'écran, ORIC se sert de préfixes de commandes. Cela veut dire qu'un octet envoyé à l'écran peut être considéré tantôt comme un motif à dessiner, tantôt comme une commande concernant la couleur, le clignotement, etc ...

La configuration de l'octet n'est pas la même s'il s'agit d'une commande ou d'un dessin. Si les bits 5 et 6 sont tous deux à 0, alors les 5 bits de droite sont considérés comme une commande (il y en a 32).

Si les bits 5 et 6 ne sont pas tous deux à 0, alors les bits de 0 à 5 serviront à créer un dessin.

En mode Haute RESolution (HIRES), les bits de 0 à 5 servent aux tracés. En mode TEXT ou BASSE RESolution (LORES), (LOW = BASSE), les bits de 0 à 6 sont les codes ASCII dont nous avons déja parlé. Les codes de contrôle (bits 5 et 6 à la valeur 0) deviennent des commandes. Le bit numéro 7 détermine si le caractère est écrit en mode normal ou en mode inverse, auquel cas il vaut 1.



Lorsqu'une commande préfixée est envoyée, elle persiste pour tout ce qui s'affiche jusqu'en fin de ligne, à moins qu'elle ne soit modifiée. Pour voir comment les commandes

préfixées peuvent être utilisées pour gérer l'affichage en mode TEXT, essayez ce programme :

```
10 FOR A=0 TO 255
20 FOR N=1 TO 24
30 PRINT A ; "UN PEU DE TEXTE SUR L'ÉCRAN"
40 NEXT N
50 FOR J=48042 TO 49002 STEP 40
60 POKE J, A
70 NEXT J, A
```

Les lignes 20 à 40 servent à remplir l'écran avec du texte. Les lignes 50 et 60 modifient le contenu de la mémoire écran en fonction de la variable A.

Vous pouvez observer quelle valeur de A commande la couleur le clignotement, etc... et ce qui en résulte selon que le texte est écrit en mode normal ou en mode inverse. Quand A est entre 24 et 31 l'écran prendra un aspect bizarre. C'est que la synchronisation est provisoirement changée. Voir appendice C.

Voici un programme qui montre comment les commandes préfixées peuvent modifier la couleur d'un caractère déjà affiché à l'écran en mode TEXT.

```
  5 REM **CHUTE D'ALIENS**
 10 GOSUB 1000:CLS
 20 FOR M=1 TO 20
 30 PAPER INT(RND(1)*4) + 4
 40 A = RND (1)*32
 50 ZAP
 60 FOR N=0 TO 1100 STEP 40
 70 POKE 48039 + N + A, 1
 80 POKE 48040 + N + A, 64
 90 POKE 48039 + N + A + 6,2
100 POKE 48040 + N + A + 6,64
110 POKE 48039 + N + A + 3,3
120 POKE 48040 + N + A + 3,64
130 SOUND 1,N/2,0
140 PLAY 1,0,5,5
150 POKE 48040 + N + A, 32
```



```
160 POKE 48040 + N + A + 6,32
170 POKE 48040 + N + A + 3,32
180 NEXT N
190 EXPLODE
200 WAIT  RND (1)*200 + 100
210 NEXT M
1000 REM**DÉFINITION DE CARACTERE**
1010 FOR N=0 TO 7
1020 READ X : POKE 46080 + (64*8) + N,X
1030 NEXT N
1040 DATA 18, 12, 30, 45, 45, 30, 18, 0
1050 RETURN
```

Vous pouvez voir, avec ce programme, que vous pouvez obtenir toutes les couleurs en avant plan sur un même écran, aussi bien que tous les changements de couleurs en arrière plan. L'instruction POKE n'est pas indispensable, avec PLOT on peut obtenir le même effet.

Normalement, les colonnes de gauche à l'écran contrôlent la couleur de l'"encre" (INK) et du "papier" (PAPER) pour tout l'écran. Si l'on "POKE" une commande numérique en mémoire écran, elle affecte une case de caractère et toutes les cases carrées situées à sa droite jusqu'au prochain endroit où l'on "POKE" une autre commande de couleur.

La commande INK étant distincte de la commande PAPER, il n'est pas nécessaire de la changer tant que d'autres caractères n'apparaissent pas à sa droite. Ceux-ci prendront la couleur correspondant au préfixe de commande le plus proche envoyé à laur gauche.

En mode HIRES, la définition comporte 200 lignes de 40 colonnes du type caractère (à 6 pixels). Là aussi il vous faut atteindre par un POKE la position immédiatement à gauche de celle que vous voulez modifier.

Voici un programme qui envoie par l'instruction POKE des commandes numériques concernant l'arrière plan dans la zone centrale de l'écran et des commandes de l'avant plan partout ailleurs. Les cercles qui sont tracés ensuite prennent

la couleur correspondant à la commande selon leur position sur l'écran.

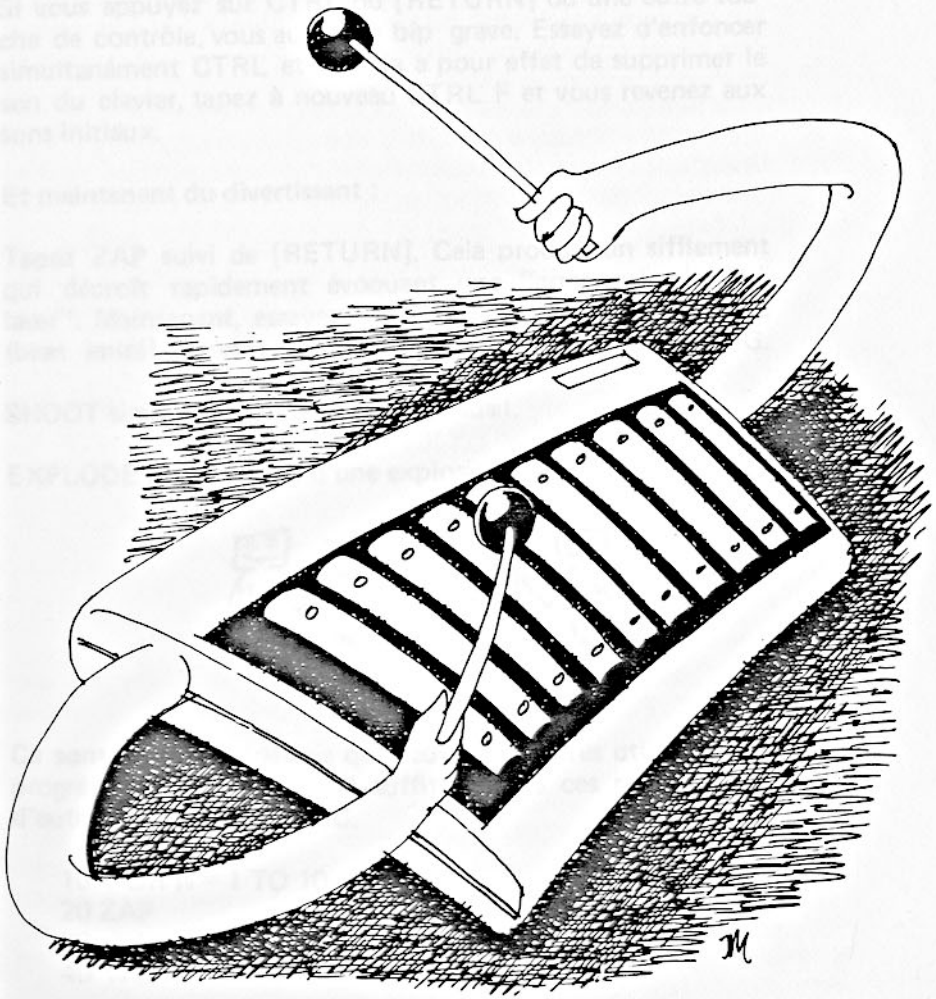
```
 5 REM**CERCLES BARIOLÉS**
10 HIRES
20 FOR N=41060 TO 48979 STEP 40
30 POKE N, INT (RND(1)*7)+1
40 POKE N-45, INT (RND(1)*7) + 1
50 NEXT N
60 CURSET 120,100,3
70 FOR X=95 TO 1 STEP-1
80 CIRCLE X,1
90 NEXT X
```

Cet autre programme montre comment un graphique peut être au départ, de toutes les couleurs et, ensuite, n'avoir plus qu'une couleur à partir d'un certain endroit.

Vous pouvez, bien sûr, obtenir par des "POKE" le clignotement ou la double hauteur à l'affichage par ce procédé.

```
 5 REM**SINUSOÏDE EN COULEURS**
10 HIRES
20 FOR N=40960 TO 49079 STEP 40
30 POKE N,INT (RND(1)*7)+1)
40 POKE N + 100,1
50 NEXT N
60 FOR A=-PI TO PI STEP 0.02
70 CURSET A*38 + 120,SIN(A)*99+99,1
80 NEXT A
```


CHAPITRE 10
Son et Musique



10. SON ET MUSIQUE

ORIC comporte quelques instructions sophistiquées produisant des sons, grâce à un composant électronique miniature spécial qui peut synthétiser 3 notes différentes simultanément aussi bien que des bruits variés.

Vous avez déjà entendu certains de ces sons. Chaque fois que vous pressez une touche, ORIC émet un bip aigu.

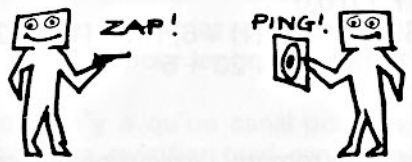
Si vous appuyez sur CTRL ou [RETURN] ou une autre touche de contrôle, vous aurez un bip grave. Essayez d'enfoncer simultanément CTRL et F. Cela a pour effet de supprimer le son du clavier, tapez à nouveau CTRL F et vous revenez aux sons initiaux.

Et maintenant du divertissant :

Tapez ZAP suivi de [RETURN]. Cela produit un sifflement qui décroît rapidement évoquant une "arme galactique à laser". Maintenant, essayez PING, un tintement de clochette (bien imité) retentit. Vous pouvez l'obtenir par CTRL G.

SHOOT simule le bruit d'un coup de fusil.

EXPLODE produit celui d'une explosion.



Ce sont 4 sons prédéfinis qui peuvent être très utiles pour la programmation de jeux. Il suffit d'écrire ces mots comme d'autres instructions BASIC.

```
10 FOR N = 1 TO 10
20 ZAP
30 WAIT 5
40 NEXT
```

Ce programme produit une salve de ZAPS !

NOTA : L'introduction, ligne 30, d'une pause est nécessaire pour permettre de détacher nettement les coups successifs. La longueur de WAIT dépend des sons.

Les commandes sonores principales sont SOUND, MUSIC et PLAY. Dans la plupart des programmes il sera nécessaire de définir le type de sons par les deux premières instructions, et de choisir sa modulation grâce à la troisième. La modulation (on dit aussi enveloppe) détermine la "forme" du son, c'est à dire s'il est attaqué avec force comme sur une guitare ou en douceur comme avec un orgue. Il vous faudra un certain temps pour vous familiariser avec ces instructions car elles offrent la possibilité de faire produire par ORIC des sons analogues à de nombreux instruments qui existent aussi bien que ceux que vous ne manquerez pas d'inventer.

Des "bruits blancs" peuvent être ajoutés pour simuler les bombes, les avions en piqué, etc ... Les possibilités ne se limitent qu'à celles de votre imagination et la mise en oeuvre n'est pas si complexe qu'il y paraît à première vue.

En voici deux exemples courts pour vous initier aux possibilités de ces instructions sonores.

```
 5 REM **MUSIQUE ?**
10 PLAY 1,0,0,0
20 MUSIC 1, RND (1) * 6, RND (1) * 12 + 1, 7
30 WAIT RND (1) * 20 + 5
40 GOTO 10
```

Vous pouvez interrompre l'exécution de ce programme par CTRL C. Pour en comprendre le fonctionnement, voir un peu plus loin les précisions sur l'instruction MUSIC.

Le programme suivant est un peu plus long, mais il vous permet d'employer l'ORIC à la façon d'un instrument de musique. Les touches supérieures de "1" à "=" produiront les sons de la gamme de demi-ton en demi-ton en commençant par do et finissant par si. La touche " / " sert à interrompre le programme. (Toujours insérer l'instruction PLAY 0,0,0,0 à la fin sinon la dernière note est jouée tant qu'on n'enfonce pas une nouvelle touche).


```
 5 REM**
10 PLAY 1, 0, 0, 0
20 GET A$
30 A = VAL(A$)
40 IF A$ = "–" THEN A = 11
50 IF A$ = "=" THEN A = 12
60 IF A$ = "/" THEN PLAY 0, 0, 0, 0 : STOP
70 IF A = 0 THEN A = 10
80 MUSIC 1,3,A,5
90 GOTO 10
```

La ligne 20 attend l'action sur une touche. La ligne 30 met la valeur numérique dans la variable A. Si vous entrez un nombre, A prendra la valeur de ce nombre. Si vous entrez "–", A = 11. Si vous entrez "=", A = 12. Si vous entrez "/", il y a arrêt. Si vous entrez toute autre touche, A prendra la valeur 0 et vous aurez la note LA, la ligne 70 transformant 0 en 10 car l'instruction MUSIC n'accepte pas la valeur 0.

Voici le détail des instructions sonores :

1 SOUND (Canal, Période, Volume)

Tous les paramètres doivent être numériques. Les erreurs seront décelées.

Canal : 1, 2 ou 3 pour les sons
4, 5 ou 6 pour les bruits.

Notez qu'il n'y a qu'un canal pour les bruits : le 4, 5 ou 6 ne font que spécifier quel canal son doit être mélangé ~~au canal 4~~. au bruit.

Période : Ce paramètre fixe la hauteur du son. Plus il est grand, plus le son est grave.

Volume : 1 à 15

0 passe la main à l'instruction PLAY.

L'instruction SOUND peut être utilisée pour produire une grande variété de sons musicaux aussi bien que non musicaux. Les canaux 1, 2 et 3 produisent des sons purs, les

canaux 4, 5 et 6 ajoutent des bruits à chaque son. La valeur de la période commande la hauteur du son, (Il s'agit de la période de la vibration, inverse de la fréquence sonore, ne pas confondre avec la durée du son). A moins que vous ne soyez branché sur un amplificateur extérieur, vous trouverez sûrement que le volume à 6 ou 7 est suffisamment puissant.

2. MUSIC (Canal, Octave, Note, Volume)

Canal : 1, 2 ou 3.

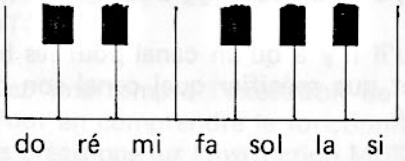
Octave : 0 à 6, 0 étant le plus grave.

Note :

1	2	3	4	5	6	7	8	9	10	11	12
do	do#	ré	ré#	mi	fa	fa#	sol	sol#	la	la#	si

Tout autre nombre tenté provoquera une erreur, assortie du message correspondant.

MUSIC a été créée pour les sons purs, et la structure a été conçue pour entrer facilement des notes, par exemple à partir d'une partition. Il y a 3 canaux et des notes sur 7 octaves et le choix du volume va de 1 à 15.



Si le volume est à 0, sélectionné par SOUD ou MUSIC, alors la sortie est pilotée par le paramètre "enveloppe" (ou modulation) de l'instruction PLAY. Les instructions SOUD et MUSIC sont toutes deux activées par PLAY. A noter que la durée peut être contrôlée par WAIT et que pour arrêter l'émission d'un son on utilise PLAY 0, 0, 0, 0.

3. PLAY (Son, Bruit, Enveloppe, Durée)

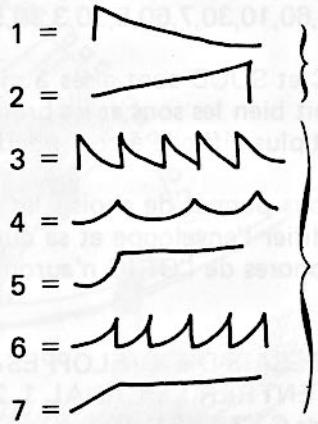
Son :

0	rien
1	canal 1
2	canal 2
3	canaux 1 et 2
4	canal 3
5	canaux 3 et 1
6	canaux 3 et 2
7	canaux 3 et 2 et 1

} actifs

Bruit : semblable à la commande timbre, mais agit sur les bruits.

Enveloppe : c'est la façon dont le son ou le bruit est modulé.



L'enveloppe agit sur la "forme" du son : répétitif, montant puis baissant, etc ... (crescendo, ...).

durée = 0 à 32767

influe sur le temps que met le bruit ou la note à commencer et à s'arrêter, en relation avec la commande enveloppe.

Quand vous utilisez les instructions sonores d'ORIC, vous pouvez avoir envie d'utiliser le déclic du clavier en appuyant sur CTRL et F simultanément. Après cela lorsqu'on enfonce une touche cela modifie les sons qui sont produits à

cet instant.

Ce programme illustre une façon de mettre en DATA la hauteur des notes et leur durée. Un accord est produit par l'ouverture simultanée des canaux 1 et 2 dans l'instruction PLAY.

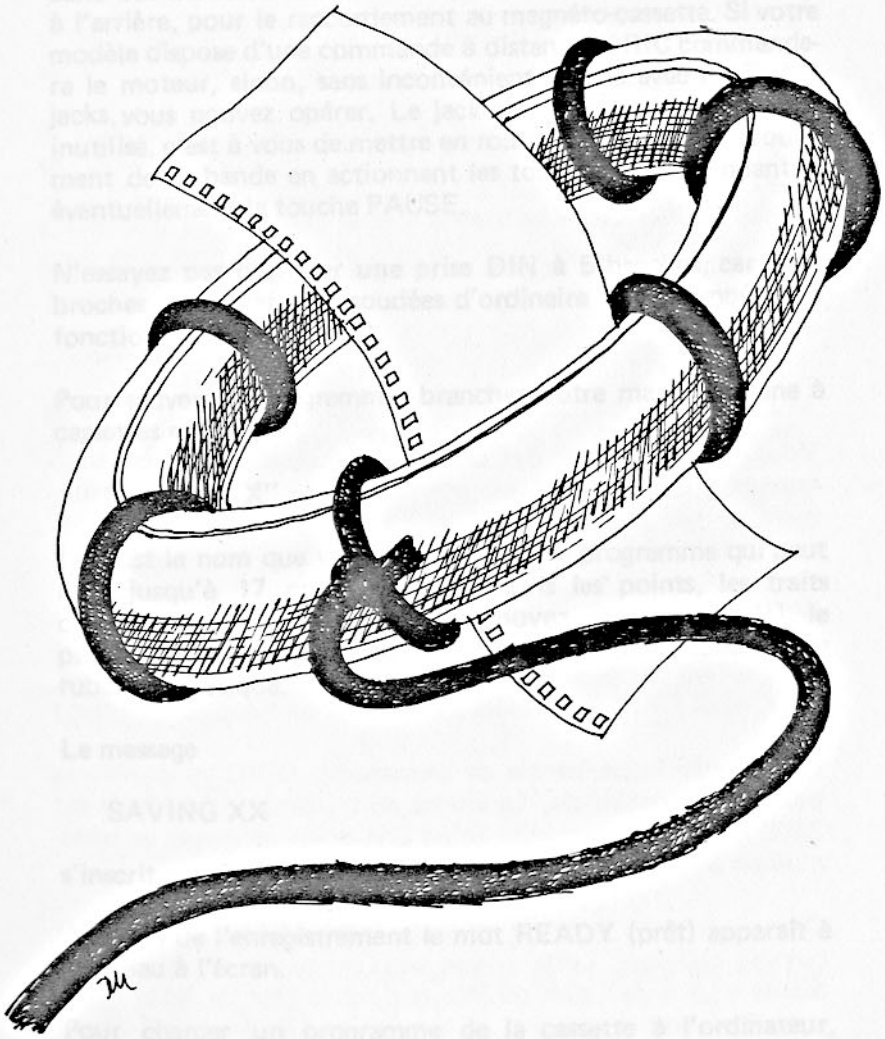
```
 10 REM**ACCORDS**
 20 FOR N=1 TO 11
 30 READ A,B
 40 MUSIC 2,3,A,0
 45 PLAY 3,0,7,2000
 50 WAIT B
 60 PLAY 0,0,0,0
 80 NEXT N
100 DATA 5,30,5,30,7,30,8,75,5,75
110 DATA 8,60,10,30,7,60,5,30,3,30,5,180
```

Alors que MUSIC et SOUD sont aisés à mettre en oeuvre et que l'on conçoit bien les sons et les bruits que l'on peut obtenir, PLAY est plus difficile à comprendre.

Ce programme vous permet de choisir les canaux 1, 2 ou 3 et aussi de modifier l'enveloppe et sa durée. Après cela, les commandes sonores de l'ORIC n'auront plus de secret pour vous.

```
  5 REM **ESSAIS D'ENVELOPPES**
 10 INPUT "ENTRER LE CANAL 1, 2 ou 3 " ; C
 20 IF C < 1 or C > 3 THEN 10
 30 INPUT "ENTRER LE NO DE L'ENVELOPPE,
    0 A 7 " ; N
 40 IF N < 0 OR N > 7 THEN 30
 50 INPUT "ENTRER LA DURÉE DE L'ENVELOP-
    PE, 0 A 32767 " ; D
 60 IF D < 0 OR D > 32767 THEN 50
 70 CLS
 80 ? "CANAL " C
 90 ? "ENVELOPPE NO " N
100 ? "DURÉE " D
110 MUSIC C,3,4,0
120 PLAY C,0,N,D
130 ? APPUYER SUR RETURN SI LE SON NE S'AR-
    RETE PAS".
```


CHAPITRE 11
Sauvegarde des Programmes sur Cassettes



11. SAUVEGARDE DES PROGRAMMES SUR CASSETTES

Quand vous avez passé un long moment à taper un programme, il est bon de savoir comment le sauvegarder et le charger ultérieurement sur votre ORIC ou un autre éventuellement.

Il vous faut un magnétophone à cassette et un câble de raccordement adéquat. Comme il a déjà été dit les fiches dépendent des modèles. ORIC a une prise DIN à 7 broches, située à l'arrière, pour le raccordement au magnéto-cassette. Si votre modèle dispose d'une commande à distance, ORIC commandera le moteur, sinon, sans inconvénient, même avec un fil à 3 jacks, vous pouvez opérer. Le jack de commande moteur est inutilisé, c'est à vous de mettre en route et d'arrêter le déroulement de la bande en actionnant les touches correspondantes, éventuellement la touche PAUSE.

N'essayez pas d'utiliser une prise DIN à 5 broches, car les 2 broches de sorties sont soudées d'ordinaire et cela empêche le fonctionnement sur ORIC.

Pour sauver un programme, branchez votre magnétophone à cassettes et taper :

CSAVE "XX"

(XX est le nom que vous donnez à votre programme qui peut aller jusqu'à 17 caractères y compris les points, les traits d'union, etc ...). Quand vous appuyez sur [RETURN], le programme est envoyé sous forme de signaux sonores sur le ruban magnétique.

Le message

SAVING XX

s'inscrit en haut à droite sur l'écran.

A la fin de l'enregistrement le mot READY (prêt) apparaît à nouveau à l'écran.

Pour charger un programme de la cassette à l'ordinateur,

commencer par s'assurer que les diverses connexions sont correctement établies, taper alors :

CLOAD "XX"

ORIC va explorer la bande jusqu'à ce qu'il trouve le programme nommé "XX" et le charger en mémoire vive dès qu'il l'a trouvé.

Pendant la recherche vous lirez

Searching ... sur la ligne du haut de l'écran.

Dès que le programme est trouvé le message devient

Loading XX

Si vous avez oublié le nom du programme, ou que vous voulez tout simplement obtenir le prochain programme inscrit sur le ruban, taper :

CLOAD " "

Vous pouvez acheter des cassettes spéciales pour micro-ordinateurs (C10 ou C15) ou utiliser des cassettes d'enregistrement sonore de bonne qualité. Les modèles de courte durée sont préférables car cela facilite la recherche des programmes.

Un dernier conseil, n'essayez pas d'enregistrer sur l'amorce en plastique de la cassette. Quand vous écoutez de la musique vous n'êtes pas à quelques notes prêt au début du morceau, vous, mais ORIC se plaindra s'il manque ne fusse qu'un octet !

Outre cette façon banale de sauvegarde, ORIC vous offre sa palette de possibilités. La vitesse de transmission normale est 2400 bauds (bits/seconde) (c'est une unité de vitesse de transmission de données).

C'est la vitesse normale utilisées avec CSAVE et CLOAD comme ci-dessus. Cette vitesse peut permettre des enregistrements tout à fait sûrs pourvu que votre tête de lecture soit propre et bien à sa place, et que vos bandes soient de bonne

qualité.

S'il y a une erreur d'enregistrement, vous le saurez en cours de chargement, le message suivant vous sera adressé :

FILE ERROR — LOAD ABORTED

ce qui signifie ERREUR D'ENREGISTREMENT — CHARGEMENT NON RÉUSSI.

Si vous souhaitez absolument être sûr que votre chef d'oeuvre est sauvé sur cassette (CSAVE) et ceci pour la postérité, alors vous n'avez qu'à ajouter S après CSAVE comme il est indiqué ci-dessous. Le transfert se fera alors à la vitesse de 300 bauds, c'est plus lent, mais bien plus sûr. (Qui va piano, va sano, va lontano). (dicton italien).

CSAVE "PROG 1", S

Si vous omettez la virgule et la lettre S, l'enregistrement se fera à 2400 bauds. Faire CLOAD "PROG 1", S au chargement.

- Si vous voulez que le programme s'exécute aussitôt la fin du chargement, ajoutez l'instruction AUTO à l'instruction CSAVE.

  CSAVE "PROG 1", AUTO

  Le CLOAD correspondant n'a pas besoin de contenir autre chose que :

  CLOAD "PROG 1"

  car l'instruction AUTO est mémorisée sur la bande magnétique en fin de programme.

- Pour sauver des blocs de mémoire, il vous faut connaître l'Adresse ou commence le bloc, et l'adresse de Fin de bloc, comme suit : (End = Fin).

  CSAVE "PROGMEM", A#400, E#499

vous sauvegarderez sur cassette le contenu des mémoires de #400 à #499.

Pour charger ces blocs, tapez

CLOAD "PROGMEM", A#400, E#499

Comme le reste de la mémoire vive n'est pas touchée, il est ainsi possible de charger une collection de caractères nouveaux, des instructions en langage machine, etc ... sans détruire le programme BASIC.

Vous pouvez aussi utiliser cette méthode pour sauver des écrans et les récupérer ultérieurement.

Assurez-vous quand vous faites cela que vous êtes dans un mode compatible sinon d'étranges choses peuvent arriver !

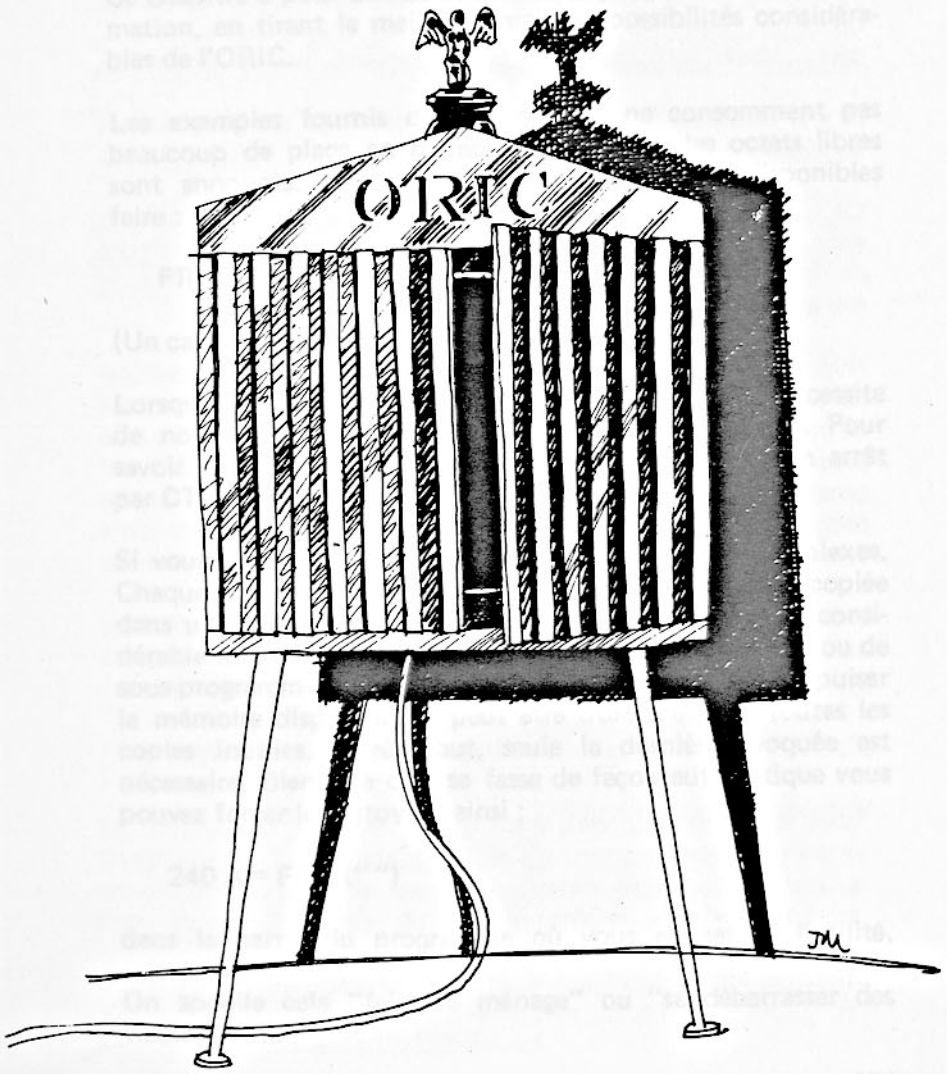
Pour sauver l'écran texte ou l'écran graphique, faire :

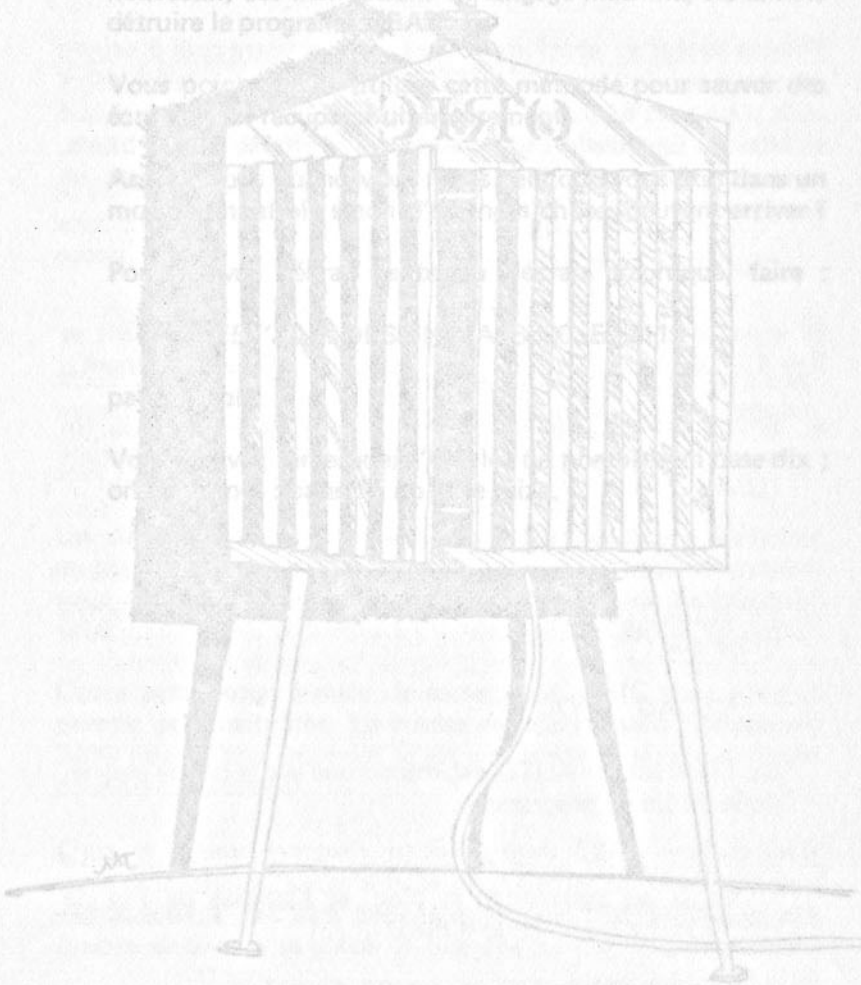
CSAVE "JOLI DESSIN", A48000, E49119

par exemple

Vous pouvez remarquer l'emploi de nombres en base dix ; on a le choix : base dix ou base seize.

CHAPITRE 12
Basic Approfondi





12. BASIC APPROFONDI

Jusqu'ici vous avez pu vous sentir parfaitement maître de l'utilisation de l'ORIC. Vous avez pu utiliser des programmes trouvés dans des périodiques ou dans des livres, et probablement aussi vous avez commencé à écrire vos propres programmes.

Ce chapitre a pour but de vous aider à progresser en programmation, en tirant le meilleur parti des possibilités considérables de l'ORIC.

Les exemples fournis dans ce manuel ne consomment pas beaucoup de place en mémoire. Au départ les octets libres sont annoncés. Pour savoir le nombre d'octets disponibles faire :

```
PRINT FRE (0)        (FREE = LIBRE)
```

(Un caractère tapé au clavier prend un octet)

Lorsque le programme s'exécute le mode TEXTE nécessite de nombreux octets et le mode HIRES encore plus. Pour savoir où vous en êtes, demander ? FRE(0) après un arrêt par CTRL C par exemple.

Si vous écrivez des programmes plus longs et plus complexes. Chaque fois qu'une variable chaîne est utilisée elle est copiée dans une zone libre de la mémoire. L'accumulation est considérable lors de l'emploi de longues boucles FOR/NEXT ou de sous-programmes emboîtés. Si vous êtes sur le point d'épuiser la mémoire disponible, il peut être utile d'effacer toutes les copies inutiles. Après tout, seule la dernière évoquée est nécessaire. Bien que cela se fasse de façon automatique vous pouvez forcer le nettoyage ainsi :

```
240 A = FRE ("")
```

dans la partie du programme où vous en sentez l'utilité.

On appelle cela "faire le ménage" ou "se débarrasser des vieilleries" ...

Quand vous écrivez de courts programmes, il est aisé de les écrire directement au clavier. Quand un problème surgit il est aisé de le résoudre, et vous pouvez probablement savoir ce que fait le programme rien qu'en examinant la liste.

Avec des programmes qui dépassent les 20 lignes environ, cela commence à devenir plus difficile, et si vous interrompez votre travail une semaine, vous pouvez en le retrouvant vous demander comment vous vous y êtes pris la première fois, quant à l'effet qu'il peut produire sur quelqu'un d'autre ... cela risque fort de lui être totalement incompréhensible.

Il y a plusieurs façons de rendre plus clairs vos programmes aussi bien pour vous que pour les autres.

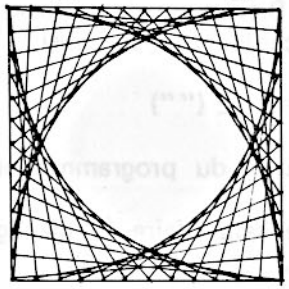
Ce ne sont pas des règles difficiles, ni trop strictes, mais elles sont de nature à améliorer indubitablement votre maîtrise de la programmation. De plus elles vous faciliteront l'accès à d'autres langages comme le Pascal ou le Forth.

Avant tout, notez sur une feuille vos idées sur la conception du programme plutôt que de vous précipiter au clavier pour tenter de les exploiter directement. Il n'est pas nécessaire non plus de vous mettre à dessiner un organigramme. En fait, un organigramme ne s'impose nullement pour mettre au point des programmes bien faits. Il suffit simplement de décrire dans l'ordre ce que l'on veut obtenir, et ceci en français ordinaire.

A titre d'exemple, imaginez que l'on vous a chargé d'écrire un programme qui montre comment des lignes droites peuvent apparaître comme des courbes régulières.

L'effet obtenu est souvent appelé "enveloppe de courbes".

Voici ce que cela donne :



L'analyse du problème peut être la suivante :

Exécution → Initialisation → Commentaires
→ Programme principal → Possibilité de répétition

Des commentaires ont été introduits dans le programme afin que n'importe qui, à l'utilisation, sache comment il est conçu et ce qu'il fait. Partir ainsi des besoins de l'utilisateur est souvent appelé approche "de l'extérieur vers l'intérieur" ("outside-in").

Chaque partie du programme sera écrite séparément, de façon modulaire. Chaque module aura un nom selon l'ordre indiqué plus haut. Au départ on aura un module de commande qui appellera les sous-programmes. C'est celui que l'on commence à écrire.

```
2000 REM* MODULE DE COMMANDE*
2010 GOSUB 3000 ' Initialisation
2020 GOSUB 4000 ' Commentaires
2030 GOSUB 5000 ' Programme principal
2040 STOP
```

Le programme est écrit ! Il ne reste plus qu'à écrire les sous-programmes. En pratique, il vous arrivera d'avoir en réserve un certain nombre de sous-programmes déjà écrits et utilisables. Il est très utile de s'en constituer une bibliothèque.

A ce stade, vous pouvez considérer que l'utilisation de l'ordinateur vous aide à programmer les modules. Etudiez les séparément.

INITIALISATION

Quand vous allumez l'ORIC, toutes les variables sont à zéro ou vides. Cela signifie qu'il n'est nul besoin d'initialiser une variable dont la valeur initiale est 0 pour les besoins du programme. Pour désigner les variables il est intéressant d'utiliser des mots et des lettres qui rappellent à quoi elles servent : par exemple :

```
LØNG = 160        (longueur du côté du carré en pixels)
```

Attention, surtout à ne pas employer 2 variables commençant par les 2 mêmes lettres. Attention aussi à ne pas utiliser les mots réservés du basic comme ON, OR ou des mots qui commencent ainsi, ou même qui les contiennent.

Si la variable est une lettre, une indication voisine grâce à un REM renseigne l'utilisateur sur son rôle. Exemple :

```
I = RND(1)*7 : REM*CHOIX DE LA COULEUR DE
    L'ENCRE, AU HASARD*
```

xxxxxxxxxx

PROGRAMME PRINCIPAL

Quand vous atteignez la partie essentielle du programme, et s'il s'agit de graphismes, il est très utile de faire une maquette sur papier et de calculer les valeurs qui empêcheront le dessin de dépasser les limites de l'écran.

Comment programmer le dessin ? Il est possible de définir chaque trait séparément, mais c'est un gaspillage de mémoire et cela manque totalement d'élégance.

Si vous connaissez les points de départ des lignes et les espacements, il vaut mieux utiliser des boucles FOR/NEXT.

Dans ce programme, il est possible d'utiliser 2 boucles FOR/NEXT imbriquées :


```
REPEAT
    FOR INCR = 31 TO 3 STEP – 2
        FOR COMPTE = 0 TO LØNG STEP INCR
            Routine de tracé
        NEXT COMPTE
    NEXT INCR
UNTIL KEY S( ) " "
```

Ces boucles sont à l'intérieur de REPEAT . . . UNTIL qui est une boucle qui assure la répétition du processus.

Observez bien que les boucles ne se chevauchent pas, et que pour entrer ou sortir d'une boucle il n'y a qu'une seule voie.

Il y a une autre particularité d'ORIC qui va nous servir beaucoup ici. C'est l'indentation de l'écriture d'un programme Usuellement les espaces inutiles sont supprimés par ORIC, aussi si vous entrez :

```
10      PRINT A
```

et que vous listez, vous aurez

```
10 PRINT A
```

Cependant si, juste après le n° de ligne vous tapez deux-points (:), l'espace après les deux-points va subsister.

Grâce à cela, on peut indenter les boucles emboîtées. Bien que cela demande initialement plus de temps, le programme y gagnera en clarté.

Plusieurs programmes dans ce livre sont indentés afin de les rendre plus lisibles, de faire ressortir leur structure. Si vous supprimez les deux-points et les indentations leur fonctionnement n'en sera, bien sûr nullement affecté.

Comme il a déjà été dit, il est prudent et habituel de numéroter les lignes de 10 en 10 pour permettre l'introduction de lignes supplémentaires ultérieurement.

Pour compléter votre programme, il lui faut un titre, votre nom, et la date de sa finition, éventuellement un numéro à changer au fur et à mesure que vous l'améliorez. C'est utile pour vous et les autres, avec qui vous serez amené à échanger des idées.

Quand vous achèterez une imprimante, la liste sur papier sera plus agréable à consulter, et vos programmes plus faciles à déchiffrer s'ils sont indentés.

Décider ainsi d'un programme et le construire de la sorte, c'est faire de la "programmation descendante". Bâtir un programme ligne par ligne est évidemment la "programmation ascendante".

La programmation descendante permet l'écriture de structures claires et nettes. Cela signifie aussi que l'instruction GOTO sera rarement utilisée. Bien que, à première vue, le programmeur trouve cela bien commode, l'usage abusif et irréfléchi des GOTO, aux quatre coins des programmes les rend difficile à comprendre et conduit au déplaisant syndrome de la "programmation spaghetti".

```
10 ? "DONNER UN NOMBRE"
20 INPUT A
30 IF A = 30 THEN GOTO 50
40 GOTO 10
50 GOTO 70
60 ?  A" EST SUPÉRIEUR A 100" : GOTO 10
70 ? A" EST LE DERNIER NOMBRE"
80 IF A ) 100 THEN GOTO 60
90 END
```

Cet exemple est un peu tiré par les cheveux, mais il montre la confusion qui peut résulter d'un emploi inconsidéré des GOTO.

L'emploi de REPEAT / UNTIL
FOR / NEXT
IF / THEN ... ELSE

vous aidera à ne pas sombrer dans une marmite de spaghetti !

Et voici maintenant le programme complet, traçant une courbe en l'enveloppant de tangentes. Il n'est pas proposé comme étant le plus merveilleux programme qui ait jamais été écrit, mais simplement comme un exemple d'utilisation des boucles, des REM et des indentations pour améliorer la lecture d'un programme.

```
1000  REM**ENVELOPPE DE COURBE**
1010  REM
1020  : REM**COPYRIGHT A.J.S.**
1030  : REM
1040  : REM**1/1/83**
1050  : REM
2000  : REM**MODULE DE COMMANDE**
2010  : GOSUB 3000 ' INITIALISATION
2020  : GOSUB 4000 ' COMMENTAIRES
2030  : GOSUB 5000 ' PROGRAMME PRINCIPAL
2040  : STOP
2050  : REM
3000  : REM**INITIALISATION**
3010  : REM
3020  : INK1:PAPER 4
3030  : LØNG = 160
3040  : TEXT
3050  : RETURN
3060  : REM
4000  : REM**COMMENTAIRES**
4010  : REM
4020  : CLS
4025  : B$="L  ****ENVELOPPE DE COURBE *****"
4030  : PRINT CHR$(27);BS
4032  : PRINT
4034  : PRINT
4035  : PRINT
4036  : PRINT "Ce programme trace des droites joignant"
4040  : PRINT "des points placés sur les côtés consécutifs"
```

```
4045  :PRINT  "d'un carré. Le "STEP" correspond à"
4050  : PRINT "l'espace entre 2 points successifs sur le"
4055  :PRINT "côté du carré"
4060  :PRINT
4065  :PRINT  "Après chaque exécution, le dessin est"
4070  :PRINT  " repris en changeant le pas (STEP). Il"
4080  :PRINT "diminue de 2"
4090  :PRINT  "Appuyer sur [RETURN] pour com-
                mencer"
4100  :GET A$
4110  :RETURN
4120  :REM
5000  :REM
5010  :REM
5020  :REPEAT
5030  :  FOR INCR = 31 TO 3 STEP – 2
5040  :  HIRES
5050  :   PRINT " PAS " INCR
5060  :   INK (INCR/6) + 1
5070  :   FOR COMPTE = 0 TO LØNG STEP INCR
5080  :    CURSET 180 – COMPTE, 10,3
5090  :    DRAW COMPTE, LØNG – COMPTE, 1
5100  :    CURSET 20, COMPTE + 10,3
5110  :    DRAW COMPTE, LØNG – COMPTE, 1
5120  :    CURSET COMPTE + 20, 170,3
5130  :    DRAW LØNG – COMPTE, – COMPTE, 1
5140  :    CURSET 20, COMPTE + 10,3
5150  :    DRAW LØNG – COMPTE, – COMPTE, 1
5160  :   NEXT COMPTE
5170  :   WAIT 100
5180  :  NEXT INCR
5190  :PRINT  "ENFONCER UNE TOUCHE POUR
        ARRETER" : WAIT 500
5200  :UNTIL KEY$ ⟨ ⟩ " "
5210  :RETURN
```

Les programmes structurés ont aussi leurs inconvénients, il faudra voir s'ils ne consomment pas exagérément de la place en mémoire. Il est d'usage de conserver un programme entièrement renseigné et d'en faire des copies débarrassées des REM et des indentations. Il est toujours fort utile de consulter le modèle mis de côté chaque fois que l'on veut procéder à une

modification du programme.

Avantages de la programmation structurée :

1) La suite des instructions est facile à lire.

2) La structure apparaît nettement, en modules distincts.

3) Le risque d'erreur est limité, et les fautes dont la recherche est crispante sont évitées ou faciles à détecter.

4) Les habitudes ainsi prises permettent d'aborder plus facilement l'étude des autres langages et leur utilisation.

Inconvénients :

1) Les programmes structurés nécessitent davantage de place en mémoire.

2) La vitesse d'exécution peut être ralentie.

3) Les possibilités de l'ordinateur peuvent ne pas être exploitées au mieux.

4) Il est plus difficile d'écrire de bons programmes structurés que de sombrer dans la négligence !

FILTRER LES ERREURS D'INTRODUCTION

Ecrire des programmes structurés fera de vous un bon programmeur, mais il n'est pas sûr que votre programme soit aussi bon qu'il devrait l'être. Quand vous allez expérimenter votre "chef-d'oeuvre", et s'il est destiné à n'importe qui, il est important d'envisager toutes les choses bizarres qu'ils vont tenter d'introduire.

Si vous demandez un nombre, qu'arrivera-t-il si l'utilisateur tape "deux" au lieu de "2" ? Que va-t-il se passer s'il appuie sur [RETURN] sans avoir rien entré ?

Heureusement ORIC n'est pas méchant pour ceux qui se trompent. Dans le premier cas, une chaîne entrée alors qu'un

nombre est attendu provoque le message "REDO FROM START" tant qu'un nombre n'est pas fourni, avec retour à l'INPUT. Dans le second cas, ORIC attendra jusqu'à ce que quelque chose soit effectivement fourni. Si l'on maintient [RETURN] enfoncé, le point d'interrogation va être sans cesse ré-écrit.

Vous pouvez faciliter la compréhension de ce que vous attendez, en faisant apparaître à l'écran un message clair. Si une erreur est commise, une deuxième indication plus précise mettra les points sur les "i". Exemple :

```
10 INPUT "INDIQUER L'ANNÉE "; A$
20 IF VAL (A$) <1900 OR VAL(A$)> 1985 THEN
   PRINT "ENTRE 1900 ET 1985, SVP" :
   GOTO 10
30 PRINT "MERCI !"
```

x x x x x x x x x x

Si vous voulez tracer une ligne ou déplacer un caractère, il est indispensable de rester à l'intérieur des limites de l'écran, il ne faut pas aller dans les colonnes réservées.

Si A est la position horizontale et B la position verticale en écran graphique (HIRES) alors quelque chose comme ceci peut aider :

```
IF A>238  THEN A=238
IF A<1    THEN A=1
IF B>198  THEN B=198
IF B<1    THEN B=1
```

C'est à vous de prévoir cela dans le programme.

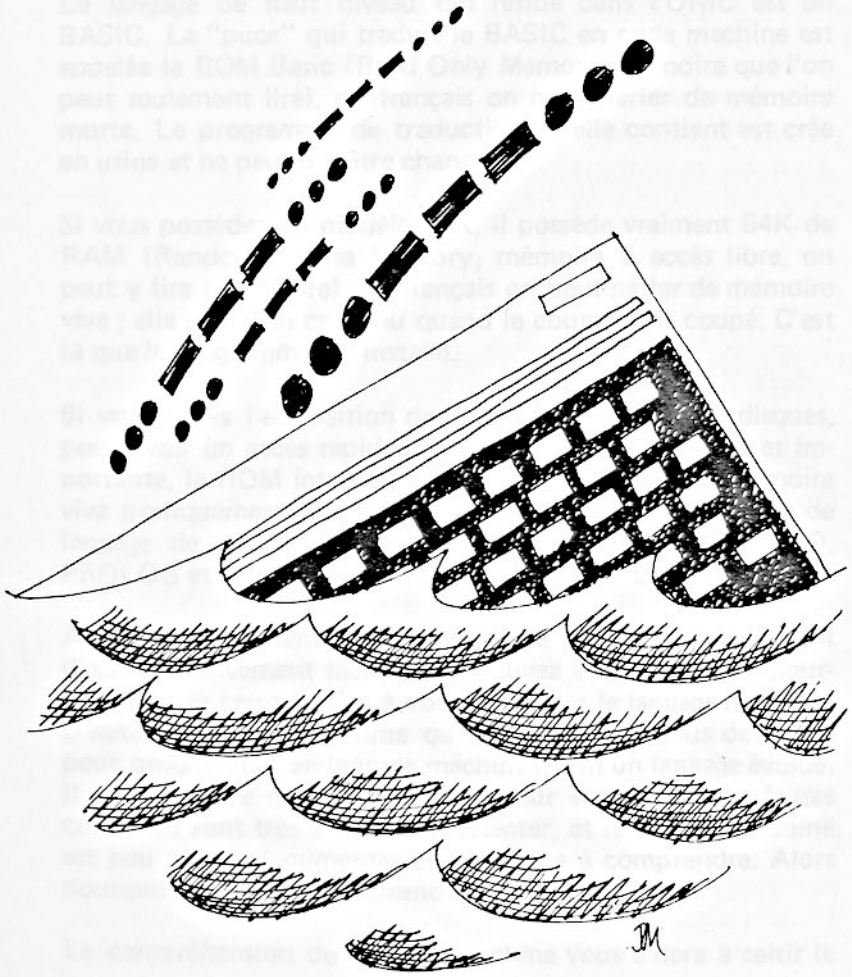
Cela peut suffire si le caractère n'est pas trop grand (pas plus de 2 pixels). Modifier les valeurs selon les cas.

Vous devez penser aux pires choses qu'un utilisateur peut faire en utilisant votre programme. Vous devez déjouer toutes les ruses qu'il ne manquera pas de tenter pour en faire ressortir les imperfections.

Un programme parfait n'existe pas, cependant il est possible de pousser très loin les tests d'erreurs d'utilisation, en pensant aux erreurs involontaires, à l'incompréhension de certains, à la coquinerie des autres. Pour vous aider demandez à vos camarades de jouer les espiègles avec votre programme : cela les amusera et vous permettra de le perfectionner.

Vous aurez alors un programme "en béton".

CHAPITRE 13
Langage Machine



13. LANGAGE MACHINE

Il a déjà été dit que les ordinateurs ne comprenaient pas le langage usuel, même l'anglais ! Le mieux qu'ils sachent faire est de déchiffrer un langage comme le BASIC qui n'est qu'une toute petite partie du vocabulaire anglais.

Les langages des ordinateurs peuvent être rangés, de ceux, de haut niveau, qui s'approchent du langage humain, jusqu'à ceux, de bas niveau, langages machines en numération binaire.

Le langage de haut niveau qui réside dans l'ORIC est un BASIC. La "puce" qui traduit le BASIC en code machine est appelée la ROM Basic (Read Only Memory, mémoire que l'on peut seulement lire), en français on peut parler de mémoire morte. Le programme de traduction qu'elle contient est crée en usine et ne peut pas être changé.

Si vous possédez un modèle 48K, il possède vraiment 64K de RAM (Random Access Memory, mémoire à accès libre, on peut y lire et y écrire), en français on peut parler de mémoire vive ; elle perd son contenu quand le courant est coupé. C'est là que le programme est installé.

Si vous faites l'acquisition de disques, de lecteurs de disques, pour avoir un accès rapide à une mémoire permanente et importante, la ROM interne est masquée et les 64K de mémoire vive pratiquement disponibles, lors de l'usage par exemple de langage de haut niveau comme FORTH, PASCAL, LOGO, PROLOG et LISP qui peuvent être utilisés sur ORIC.

Avec tout ce potentiel de langages de haut niveau, qui sont d'accès relativement facile, vous pouvez vous demander pourquoi l'on se casse la tête à vouloir utiliser le langage machine. D'autant plus qu'on estime qu'il faut dix fois plus de temps pour programmer en langage machine qu'en un langage évolué. Il n'y a pas de messages d'erreur pour vous aider, les fautes commises sont très difficiles à détecter, et le langage machine est peu aisé à documenter et peu facile à comprendre. Alors pourquoi se mettre à l'apprendre ?

La compréhension du langage machine vous aidera à saisir le

fonctionnement des ordinateurs. Des programmes réussis en langage machine sont exécutés bien plus vite qu'en aucun des langages évolués. Imaginez que vous demandez à un allemand d'expliquer à un italien comment faire un travail, vous allez vite comprendre qu'il vaut mieux s'adresser directement à l'italien dans sa langue !

Il existe différents moyens de rendre accessible le langage machine. D'abord, on l'écrit usuellement en base seize, car une page de nombres binaires ressemble rapidement à un embrouillamini de zéros et de uns, et les nombres en base dix ne montrent pas de façon claire ce qui se passe dans un octet.

Voilà pourquoi sur ORIC il est possible d'entrer les nombres soit en notation décimale, soit en notation hexadécimale. C'est plus facile à lire.

Au cœur de l'ORIC, est placée l'unité centrale, le micro-processeur : c'est lui qui joue le plus grand rôle.

(Central Processeur Unit, C.P.U. dit-on en langue anglaise.)

Tous les micro-ordinateurs sont bâtis autour d'un micro-processeur. Ceux qui utilisent le même qu'ORIC, peuvent dans les limites des possibilités de l'ordinateur utiliser le même langage machine.

ORIC utilise un micro-processeur 6502 fabriqué par Rockwell International Corporation. Il en existe d'autres, comme le 6800, le 6809, le Z80 et le 8080.

Ils fonctionnent de façons diverses et utilisent chacun un jeu d'instructions particulier.

En son sein, le micro-processeur manipule des nombres, stockés sous forme de 8 chiffres binaires. Ces nombres sont chargés en diverses adresses mémoires et manipulés soit comme des instructions soit comme des données.

Par exemple, si le 6502 reçoit le nombre 10101001, il le comprend en tant qu'instruction : charger un registre, une case mémoire spéciale appelée accumulateur, avec le prochain

nombre reçu. Le 6502 ne comprend que les nombres binaires de 8 bits. ORIC peut accepter les nombres en base dix ou seize, il les convertit en base deux avant de les envoyer au 6502.

Pour aider leur mémorisation une abréviation est retenue pour chaque instruction du 6502, ainsi pour l'exemple ci-dessus, 10101001, ou 169 (en base 10) ou # A9 (en base 16) est désigné par LDA (LoaD Accumulateur, charger l'accumulateur).

Ces aides mémoires (ou pense-bête) sont désignés sous le nom de "mnémoniques". Pour vous aider vous en trouverez la liste en appendice K.

Si vous chargez #A9 dans l'ORIC, il va comprendre, mais vous peut être pas, et pour vous éclairer vous pouvez ajouter à la façon d'un commentaire REM le sens de cette instruction sous forme mnémonique :

```
100 DATA # A9, # 20 ' LDA, #20
```

Pour vous aider à charger vos programmes en langage machine, il est possible d'écrire un programme qui vous permette d'entrer directement les mnémoniques, les variables, les données, les adresses, etc ... et qui les décodera pour les écrire dans le binaire que la machine absorbe. L'utilisateur tape les mnémoniques (le programme source) et l'assembleur le traduit en code machine (le programme objet). Un désassembleur fait le contraire.

x x x x x x x x x x

PEEK et POKE

Comment savoir le contenu de chaque cas mémoire ?

Nous avons déjà vu PEEK et POKE dans ce qui précède.

Tapez

```
PRINT PEEK (48225)
```

ceci permet d'explorer une partie de la mémoire d'écran texte. Le nombre obtenu est en base dix ; la case mémoire n° 48225 contient l'octet en base deux équivalent. Si l'écran comporte un caractère à cet endroit, le nombre en est le code ASCII, s'il n'y a rien on recevra 32, code ASCII de "blanc" ou "espace".

Pour modifier la valeur introduire :

```
POKE 48225, 128
```

Vous voyez l'endroit de l'écran concerné ; un carré plein s'y inscrit, correspondant au code ASCII 128.

Si vous jetez un coup d'oeil sur la carte des mémoires en appendice A, vous pouvez savoir ce qui est stocké aux divers endroits. Entraînez-vous à l'emploi de POKE pour faire apparaître des caractères à l'écran en mode texte ou en mode graphique.

On a déjà mis en œuvre cette technique au chapitre 4. Ce ne serait pas malin d'aller modifier les mémoires par l'instruction POKE dans les pages de 0 à 3 (adresses de 0 à 1024 en base dix). Vous pouvez cependant essayer, pour voir ; c'est sans risque pour l'ORIC quoi que vous fassiez. Un RESET ou la mise hors tension est alors utile.

ADRESSES

Vous vous êtes peut être demandé comment ORIC mémorise les nombres au-delà de 255, surtout qu'il y a 65536 adresses différentes.

Voici la méthode : les numéros des adresses sont stockés sur 2 octets.

	1er octet	2ème octet
128 donne	10000000	00000000

Si le nombre est supérieur à 255, le second octet contient le

nombre de fois que 256 est contenu dans ce nombre. Exemple :

258 donne	00000010		00000001
	2	+	(1 x 256)

Il peut paraître étrange que les choses se fassent dans cet ordre; c'est ainsi! Aussi, si vous savez qu'un nombre est stocké sur les deux adresses 20345 et 20346, pour calculer le nombre ainsi mémorisé vous devez faire :

```
PRINT PEEK(20 345) + (PEEK(20 246)*256)
```

ORIC vous épargne la peine d'écrire tout ceci grâce à l'instruction DEEK.

```
PRINT DEEK (20 345)
```

suffit

DEEK est une abréviation de Double PEEK.

Si vous voulez changer le nombre contenu à cette double adresse faites :

```
DOKE 20 345,N
```

et N sera écrit sous forme de 2 octets en 20 345 et 20 346.

x x x x x x x x x x

Parfois il est utile de disposer d'un court programme en langage machine, bien que le reste soit écrit en BASIC. Il peut être entièrement indépendant ou être écrit de façon plus laborieuse à coups de "POKE".

Plusieurs instructions sont installés sur l'ORIC.

CALL X, ou X est une adresse en mémoire, branche l'exécution en langage machine à partir de l'adresse spécifiée. Le retour au BASIC se fait quand le mnémonique RTS (ReTurn From Subroutine, revenir du sous-programme) est atteint, ou plutôt son équivalent binaire.

Une autre manière d'obtenir un résultat à partir d'un sous-programme codé en assembleur est l'utilisation de l'instruction USR : DEF USR et PRINT USR(0).

Le branchement s'obtient par

DEF USR = adresse de départ

Si l'on écrit alors PRINT USR (0), le résultat produit par le sous programme en langage machine est extrait de l'accumulateur numérique et affiché à l'écran.

Voici un exemple d'utilisation :

```
  5 REM*** RAD/DEG CONVERSION***
 10 FOR DISP=0 TO 12
 20 : READ DTA
 30 : POKE # 400 + DISP,DTA
 40 NEXT DISP
100 DATA #A9, #07      'LDA = CON57: Le moins signifi-
                       catif
110  DATA #A0, #04     'LDA = CON57: Le plus significatif
120 DATA #4C, #73, # DE
                       'JMP MOVFM : Virgule flottante
130 DATA #86, #65, # 2E, # E0, # D8
                       'CONSTANTE 180/PI
```

Faire RUN, puis taper DEF USR # 400. N'importe quand, dès lors, PRINT USR (0) fournira le coefficient de conversion des radians en degrés.

Voici d'une façon détaillée ce qui se passe : La valeur entre parenthèse de la fonction USR est envoyée dans l'accumulateur et un JSR à l'adresse #21 est exécuté.

Les adresses #21 à #23 doivent contenir un JMP à l'adresse de début du sous programme en langage machine.

La valeur à renvoyer est placée dans l'accumulateur en notation décimale à virgule flottante.

Pour obtenir un entier codé sur 2 octets à partir de la valeur

en virgule flottante contenue dans l'accumulateur, le sous-programme devra exécuter un JSR à l'adresse #D867. Au retour la valeur entière sera en #34(octet le plus significatif) et #33 (octet le moins significatif).

Si vous voulez convertir ce résultat entier en sa valeur décimale à virgule flottante, afin que la fonction puisse restituer la valeur qu'elle a calculé, les 2 octets ci-dessus cités doivent être transférés, l'un dans le registre A (octet le plus significatif) l'autre dans le registre Y (octet le moins significatif). Ainsi lorsqu'on a l'exécution d'un JSR à l'adresse #D8D5, au retour, la valeur décimale a été rangée dans l'accumulateur.

Il existe 2 autres opérations qu'ORIC peut accomplir :

! peut être définie comme une nouvelle instruction qui n'existe pas déjà dans le BASIC de l'ORIC.

&(X) (ou X = 0 à FFFF)

peut être défini comme une fonction qui n'est pas déjà prévue dans le BASIC de l'ORIC.

Les sous-programmes de création doivent être écrits en langage machine et chargés dans une zone particulière de la mémoire. L'adresse de départ est chargée ainsi :

```
DOKE #2F5, adresse     (pour !)
DOKE #2FC, adresse     (pour &(X))
```

- Pour définir ! comme signifiant PRINT AT (AT = à) introduire ceci (@est appelé "à commercial" et se lit "at").

```
  5 REM*****PROGRAMME CRÉANT L'INSTRUC-
    TION***** «PRINT @ X, Y ; "JJJJ»
 10 REPEAT
 20 READ DTA
 30 POKE #400 + CL,DTA
 40 CL = CL + 1
 50 UNTIL DTA = #FF      : REM FIN DE PROG.
100 DATA #20, #96, #D9   : REM JSR GTVALS
```

```
110 DATA #AC, #F8, #02    : REM LDY GCOL
120 DATA #C8              : REM INY
130 DATA # 8C, #69, #02   : REM STY CURCOL
140 DATA #A5, #1F         : REM LDA GCL
150 DATA #A4, #20         : REM LDY GHC
160 DATA #85, #12         : REM STA CURBAS
170 DATA #84, #13         : REM STY CURBAS + 1
180 DATA #A9, #3B         : REM LDA #';'
190 DATA #20, #DB, #CF    : REM JSR SYNCHR
200 DATA #4C, #61, #CB    : REM JMP PRINT
210 DATA #FF
220 DOKE #2F5, #400
```

Si vous faites

```
! X, Y ; "ORIC"
```

le mot ORIC apparaîtra à l'écran, la lettre O ayant pour coordonnées X, Y.

- Pour définir & comme la fonction qui fournit la position verticale du curseur, voici le sous-programme :

```
 5 REM**EXTENSION CMD VERT/CURS/POS
10 FOR N = 0 TO 5
20 READ DTA
30 POKE #400 + N, DTA
40 NEXT
50 DOKE # 2FC, #400
60 DATA # AC, #68, #02
70 DATA #4C, #FD, #D3
```

Il est très utile de savoir en quelle ligne est le curseur lorsqu'on veut utiliser les lettres en double hauteur, afin d'éviter d'avoir le bas des lettres au dessus de leur moitié supérieure.

Une ligne comme celle-ci, ajoutée à un programme, empêchera ce phénomène désagréable de se produire :

```
500 IF &(0) / 2 < > INT (&(0)/2) THEN PRINT
```


cela fera descendre le curseur d'une ligne le cas échéant.

- Quelques adresses de la page 4 (#0400 à #0420) ont été réservées pour vos sous-programmes en langage machine. Toute autre zone de la mémoire peut servir également mais le programme BASIC peut s'écrire par dessus le sous-programme et le détruire.

  Pour réserver de la mémoire pour des sous-programmes en langage machine, le plafond de la zone mémoire peut être abaissé. Pour connaître sa position actuelle demander :

  ```
  PRINT DEEK (#A6)
  ```

  Calculez le nombre d'octets nécessaires pour loger votre sous-programme binaire, ajouter un petit nombre, par sécurité (à moins que vous ne soyez à court de mémoire) et soustrayez le total du nombre que vous avez obtenu avec DEEK.

  Entrez alors :

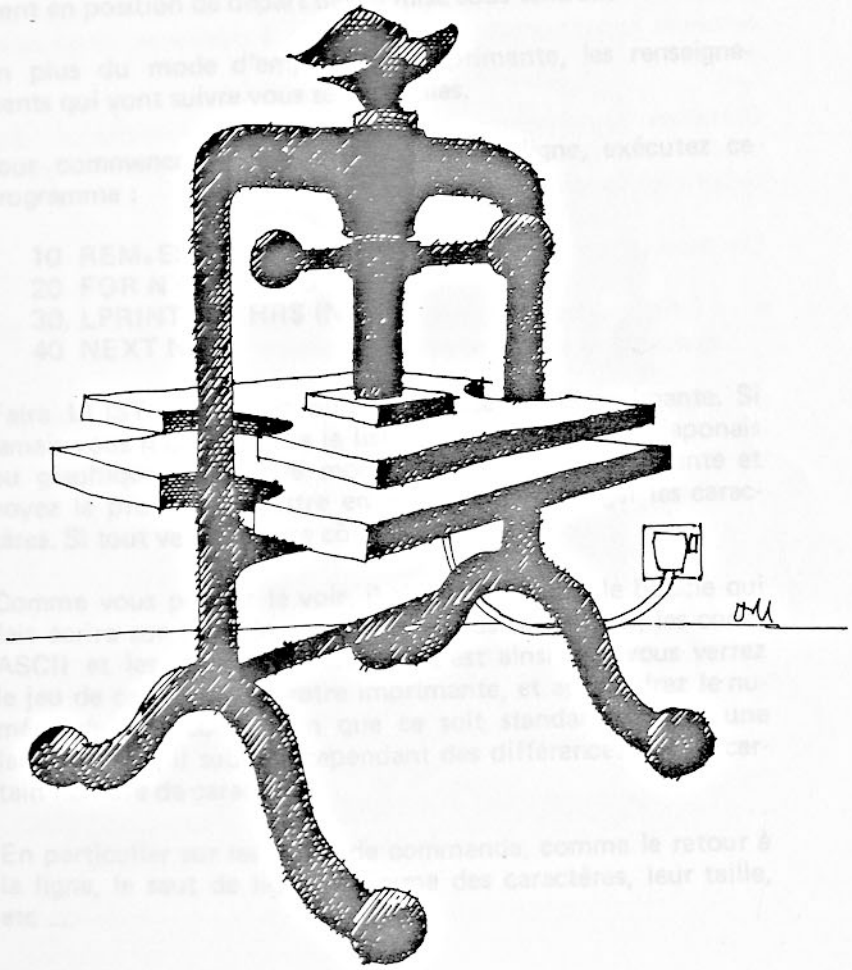
  HIMEM X ou X est le nouveau plafond de la mémoire que vous venez de calculer.

  Si vous voulez en savoir plus sur l'utilisation du 6502, et sa programmation vous trouverez divers livres qui traitent le sujet de façon approfondie.

  Deux des plus utiles sont ceux de Rodnay Zaks chez SYBEX et Lance Leventhal aux Editions Radio.

  On peut aussi s'adresser à la Société Rockwell International qui a créé le 6502.

CHAPITRE 14
Utilisation d'une Imprimante



14. UTILISATION D'UNE IMPRIMANTE

ORIC peut être utilisée avec n'importe quelle imprimante munie d'une interface CENTRONICS. Il vous faudra le câble de raccordement en même temps que l'imprimante. La prise est placée à l'arrière à côté du port d'extension.

Il faut raccorder l'imprimante avant de l'allumer.

Si tout est en ordre la tête d'écriture se place automatiquement en position de départ dès sa mise sous tension.

En plus du mode d'emploi de l'imprimante, les renseignements qui vont suivre vous seront utiles.

Pour commencer, l'imprimante étant en ligne, exécutez ce programme :

```
10 REM*ESSAI D'IMPRIMANTE**
20 FOR N = 0 TO 255
30 LPRINT N, CHR$ (N)
40 NEXT N
```

Faire LLIST, vous allez obtenir la liste sur l'imprimante. Si jamais vous n'obtenez pas la liste mais des caractères japonais ou graphiques, relisez le mode d'emploi de l'imprimante et voyez le procédé à mettre en oeuvre pour changer les caractères. Si tout va bien de ce côté-là, faire RUN.

Comme vous pouvez le voir, il s'agit d'une simple boucle qui fait écrire sur l'imprimante, et non plus sur l'écran, les codes ASCII et les caractères associés. C'est ainsi que vous verrez le jeu de caractères de votre imprimante, et apprendrez le numéro de leur code. Bien que ce soit standardisé dans une large mesure, il subsiste cependant des différences sur un certain nombre de caractères.

En particulier sur les codes de commande, comme le retour à la ligne, le saut de ligne, la forme des caractères, leur taille, etc ...

Par exemple, les imprimantes Microline proposent des caractères standards, serrés ou élargis. Si vous tapez :

```
LPRINT CHR$ (31)
```

les prochains caractères sortiront 2 fois plus gros, ce qui est très utile pour les titres, ...

```
LPRINT CHR$ (30)
```

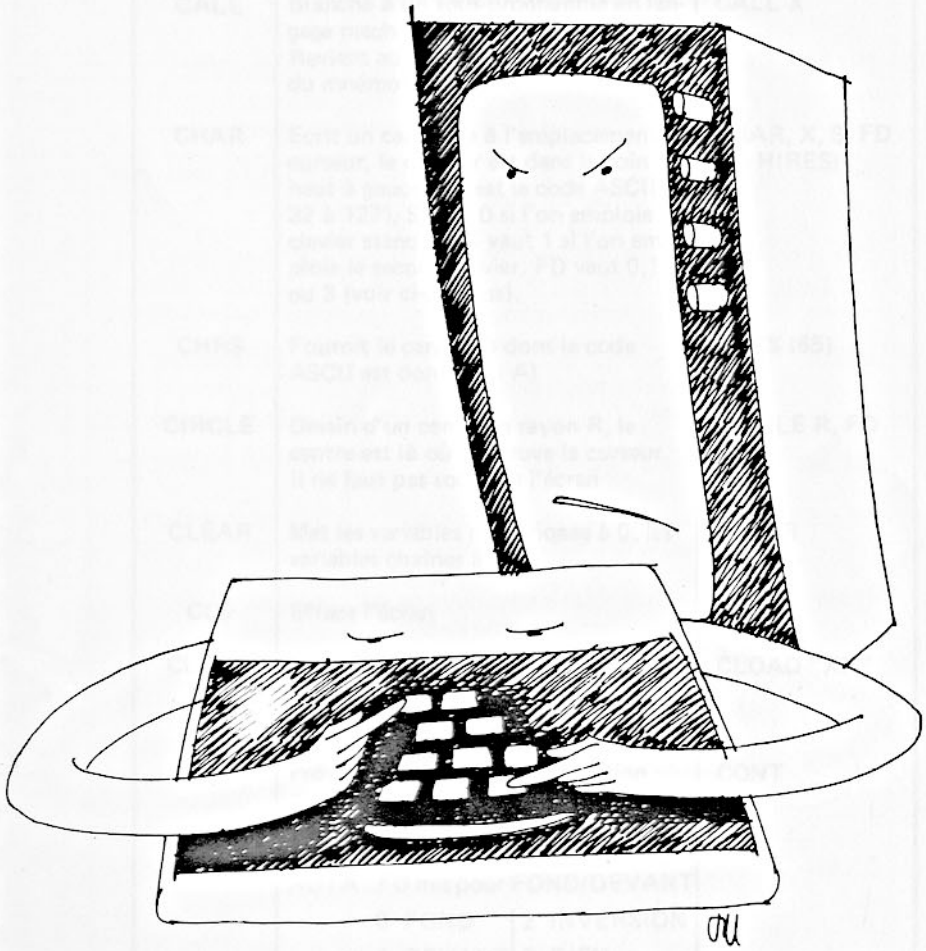
provoque une avance du papier jusqu'en bas d'une page sur certaines imprimantes. Ces codes sont à connaître, ils peuvent être écrits dans les programmes.

Certaines imprimantes offrent la possibilité d'obtenir sur papier la réplique de n'importe quel dessin présent à l'écran, ce sont les imprimantes graphiques.

Les deux usages principaux des imprimantes sont :

- l'Edition des programmes sous forme de liste pour en examiner la structure et les travailler.
- L'impression de courrier préalablement saisi à l'écran.

CHAPITRE 15
Le Basic de l'Oric



15. LE BASIC DE L'ORIC

Instruction	Commentaire	Exemple
ABS	Valeur absolue. (ici la réponse est 4)	?ABS (–4)
AND	ET logique	? A AND – B
ASC	Fournit le code ASCII du premier caractère d'une chaîne	C = ASC(N$) D = ASC ("A")
ATN	Fournit arc tangente en radians	Z = ATN (Y/4)
CALL	Branche à un sous-programme en langage machine, à partir de l'adresse X. Revient au BASIC après la rencontre du mnémonique RTS.	CALL X
CHAR	Ecrit un caractère à l'emplacement du curseur, le curseur est dans le coin en haut à gauche, X est le code ASCII (de 32 à 127), S vaut 0 si l'on emploie le clavier standard, S vaut 1 si l'on emploie le second clavier. FD vaut 0,1,2 ou 3 (voir ci-dessous).	CHAR, X, S, FD (en HIRES)
CHR$	Fournit le caractère dont le code ASCII est donné (ici A)	CHR$ (65)
CIRCLE	Dessin d'un cercle de rayon R, le centre est là où se trouve le curseur. Il ne faut pas sortir de l'écran	CIRCLE R, FD
CLEAR	Met les variables numériques à 0, les variables chaînes à vide.	CLEAR
CLS	Efface l'écran	CLS
CLOAD	Transfert d'un programme d'une cassette à la mémoire vive. XX est le nom du programme.	CLOAD "XX"
CONT	Provoque la reprise de l'exécution d'un programme après un BREAK (provoqué par STOP ou CTRL C)	CONT
	NOTA : FD mis pour FOND/DEVANT 0 FOND | 2 INVERSION 1 DEVANT | 3 RIEN	

Instruction	Commentaire	Exemple
COS	Fournit le cosinus de l'angle N. (N en radians)	A = COS (N)
CURMOV	Déplacement relatif depuis la position précédente du curseur. Ne trace pas de trait. Marque éventuellement le point d'arrivée selon FD.	CURMOV X, Y, FD
CURSET	Déplacement absolu à la position X,Y. X entre 0 et 239, Y entre 0 et 199. Marque éventuellement le point selon FD.	CURSET, X, Y, FD
CSAVE	Transfert d'un programme de la mémoire vive à la cassette, sous le nom XX.	CSAVE "XX"
DATA	Conserve une liste de données qui peuvent être lues par l'instruction DATA. Peut être mis n'importe où, peut contenir des nombres ou des chaînes, les espaces sont ignorés sauf entre " "	DATA 1,2, oui, "non"
DEEK	Fournit le contenu d'un octet augmenté de 256 fois le contenu de l'octet suivant.	?DEEK(45610)
DEF FN	Définit une fonction numérique	DEF FN A(Z)=Z+4
DEF USR	Fixe l'adresse de départ d'un sous-programme en langage machine.	DEF USR = #400
DIM	Réserve de la place en mémoire pour les tableaux : indispensable au-delà de l'indice 10. L'exemple réserve 11x6 = 66 places de 255 caractères.	DIM A$ (10,5)
DOKE	Place la valeur V dans les mémoires X et (X + 1). INT (V/256) dans (X + 1), V – INT(V/256) dans X.	DOKE X, V
DRAW	Le curseur étant en A, B. Draw trace un trait de A, B à A + X, B + Y. FD est entre 0 et 3.	DRAW, X, Y, FD
END	Ordre d'arrêt de l'exécution d'un programme.	END

Instruction	Commentaire	Exemple
EXP	Fonction exponentielle. Base e = 2,71828...	A = EXP (N)
EXPLODE	Produit un bruit d'explosion.	EXPLODE
FALSE	Booléen. Vaut 0. (correspond à FAUX).	FALSE
FILL	Remplit un certain nombre de zones A, sur B rangées, selon la commande préfixée N. Il y a 200 rangées de 40 cellules, chaque cellule ayant 6 pixels. (les mots zone, segment , cellule, sont proposés dans le texte pour exprimer la même notion).	FILL B, A, N
FN	Fournit le résultat calculé par une fonction prédéfinie.	? FNA (X)
FOR . . . TO STEP NEXT	Crée une boucle qui commence à FOR et se termine à NEXT, et qui exécute les instructions intermédiaires un certain nombre de fois selon le pas (STEP) depuis une valeur (ici 1) jusqu'à une autre (ici 11) incrémentée (de 3 ici) à chaque fois. STEP est optionnel et vaut 1 par défaut. STEP peut être négatif. Toute boucle s'exécute au moins une fois.	FOR N = 1 TO 11 STEP 3 : NEXT N
FRE	Fournit le nombre d'octets non utilisés en mémoire vive au moment de l'interrogation. Fait le ménage dans la mémoire. (voir le texte).	? FRE (0) A = FRE ("")
GET	Arrête l'exécution jusqu'à ce qu'une touche soit actionnée. Ne prend qu'un caractère, l'affecte à la variable (ici A$).	GET A$
GOSUB	Ordre de branchement à l'adresse indiquée. Le retour est commandé par RETURN	GOSUB 1000
GOTO	Ordre de branchement sans condition à l'adresse indiquée et sans retour.	GOTO 4000

Instruction	Commentaire	Exemple
GRAB	Permet l'utilisation de la zone mémoire entre #9800 et #B 400 (48K) ou entre #1800 et #3400 (16K). (Voir la carte des mémoires).	GRAB
HEX$	Fournit la valeur en base seize du nombre V écrit en base dix	? HEX$ (V)
HIMEM	Abaisse le plafond de la mémoire disponible pour les programmes en BASIC. Libère au-dessus une zone qui peut servir à des sous-programmes en langage machine.	HIMEM #8700
HIRES	Fait passer en mode haute définition avec fond noir et avant plan blanc, le curseur est mis en 0,0. 3 lignes de texte subsistent en bas de l'écran sans changement de couleurs.	HIRES
IF ... THEN ELSE	Si l'expression suivant IF est vraie, alors la ou les instructions après THEN sont exécutées. Si l'expression est fausse le programme exécute le (ou les) instruction(s) après ELSE. ELSE peut être omis, dans ce cas le programme saute à la ligne suivante si l'expression est fausse.	IF A 〉10 THEN ? "OK" ELSE ? "NIET"
INK	Change la couleur de l'avant plan pour tout l'écran. N est un entier de 0 à 7.	INK N
INPUT	Arrête l'exécution d'un programme Attend un nombre ou une chaîne au clavier. La transfère dans la variable spécifiée après action sur la touche [RETURN]. Ici le message "AGE" apparaîtra à l'écran. Le nombre introduit ira en variable A. (;obligatoire). Un point d'interrogation signale l'attente.	INPUT N INPUT N $ INPUT "AGE";A
INT	Fournit le plus grand entier inférieur ou égal à la valeur entre parenthèse. INT (−2.5) vaut −3.	X = INT (Y+0.5)

Instruction	Commentaire	Exemple
KEY$	Lit le clavier. Si à ce moment précis une touche est enfoncée, en charge la valeur dans la variable X$, sinon passe à l'instruction suivante.	X $ = KEY$
LEFT$	Sous chaîne gauche. Met dans L$ les N caractères de gauche de A$.	L$ = LEFT$ (A$,N)
LEN	Fournit le nombre de caractères de la chaîne.	L = LEN (A$)
LET	Affecte une valeur, un contenu à une variable LET est optionnel, attention au sens de = !	LET A = 4
LIST	Affiche tout le programme (à l'écran) Affiche la ligne 100 Affiche toutes les lignes jusqu'à la ligne 50. Affiche la fin du programme depuis la ligne 1000. Affiche toutes les lignes de 50 à 70.	LIST LIST 100 LIST − 50 LIST 1000 − LIST 50 − 70
LLIST	Comme LIST mais sort sur l'imprimante (attention si elle n'est pas en ligne ...).	LLIST LLIST 50 − 70
LN	Fonction logarithme à base e, ou népérien	A = LN (X −2)
LOG	Fonction logarithme à base dix, ou vulgaire.	B = LOG (Y+1)
LORES	Fait passer en mode basse définition (40 x 25 + 3) LORES 0 utilise le clavier standard. LORES 1 utilise le deuxième clavier. Le fond devient noir, l'encre blanche.	LORES 0 LORES 1
LPRINT	Ecrit des nombres ou des chaînes sur l'imprimante.	LPRINT N LPRINT A$
MID$	Sous-chaîne intérieure. Met dans D$ les N caractères de A$ à partir de celui de rang I. Met dans K$ tous les caractères de B$ à partir de celui de rang I.	D$ = MID$ (A$,I,N,) K$ = MID$ (B$,J,)

Instruction	Commentaire	Exemple
MUSIC	Donne une note pure. Canal, Octave, Note, Volume.	MUSIC C,O,N,V
NEW	Efface le programme, met les variables à 0 ou à vide.	NEW
NOT	NON logique	? NOT (4 = 5)
ON...GOSUB	Branchement à divers sous-programmes selon une valeur. Si A = 1 on va en 100, 2 en 200, 3 en 300.	ON A GOSUB 100, 200, 300
ON...GOTO	Branchement sans condition. Si X = 1 on va en 500, 2 en 520, 3 en 530.	ON X GOTO 500, 520, 530.
OR	OU logique	? A OR (B = 5) IF A < 5 OR B > 7 THEN GOTO 100
PAPER	Change la couleur du fond sur tout l'écran. N est un entier de 0 à 7.	PAPER N
PATTERN	Modifie le motif tracé par DRAW X = 255 au départ et DRAW trace en continu. En changement X on obtient des traits discontinus (de 1 à 255), avec l'instruction DRAW.	PATTERN X
PEEK	Fonction fournissant le contenu d'une mémoire ici d'adresse X.	A = PEEK (X)
PI	Constante, vaut 3. 14159265 (valeur approchée de π)	? 2 * PI
PING	Produit un son de clochette.	PING
PLAY	Pour les sons, agit après MUSIC. Son, Bruit, Enveloppe, Durée de l'enveloppe. PLAY 0,0,0,0 arrête la musique, ou le bruit.	PLAY S,B,E,D
PLOT	Ecrit un caractère, ou une chaîne de caractères à partir de l'endroit de coordonnées X, Y en TEXT.	PLOT X, Y, "C" PLOT X, Y, A$

Instruction	Commentaire	Exemple
POINT	Renvoie 0 si le pixel indiqué est couleur du fond. Renvoie −1 s'il est de la couleur de l'encre (de l'avant-plan) X va de 0 à 239, Y va de 0 à 199	? POINT (X,Y)
POKE	Place le nombre V dans la mémoire d'adresse N V est un entier entre 0 et 255. N est un entier entre 0 et 65535 (ou #0 et #FFFF)	POKE N,V
POP	Fait oublier à l'ORIC l'adresse de retour pour l'instruction GOSUB la plus récente. Le prochain RETURN rencontré provoquera un branchement juste après l'instruction GOSUB précédant celle dont l'adresse est perdue.	POP
POS	Donne la position horizontale du curseur	POS Ø
PRINT	Ecrit des nombres ou des chaînes à l'écran.	PRINT "SALUT"
	? est une abréviation de PRINT. La virgule provoque un décalage, le " ; " une juxtaposition.	? N, A$? A$; B$
PULL	Décale d'un cran la pile d'adresse dans les boucles REPEAT. (analogie avec POP).	PULL
READ	Lit la donnée indiquée par le pointeur en DATA, l'affecte à une variable.	READ A$, N
RELEASE	Alloue la zone qui a été décrite à l'instruction GRAB à l'écran graphique.	RELEASE
REM	Permet d'écrire des commentaires dans les programmes. Tout ce qui suit REM n'est pas lu. ' (apostrophe) est une abréviation possible mais pas juste après un numéro d'instruction.	REM TITRE 10 A = 5: 'Début
REPEAT	Les instructions incluses entre REPEAT et UNTIL sont répétées TANT QUE le test qui suit UNTIL est faux. Quand il est vrai, le programme continue à l'instruction qui suit UNTIL. S'exécute au moins une fois.	REPEAT UNTIL

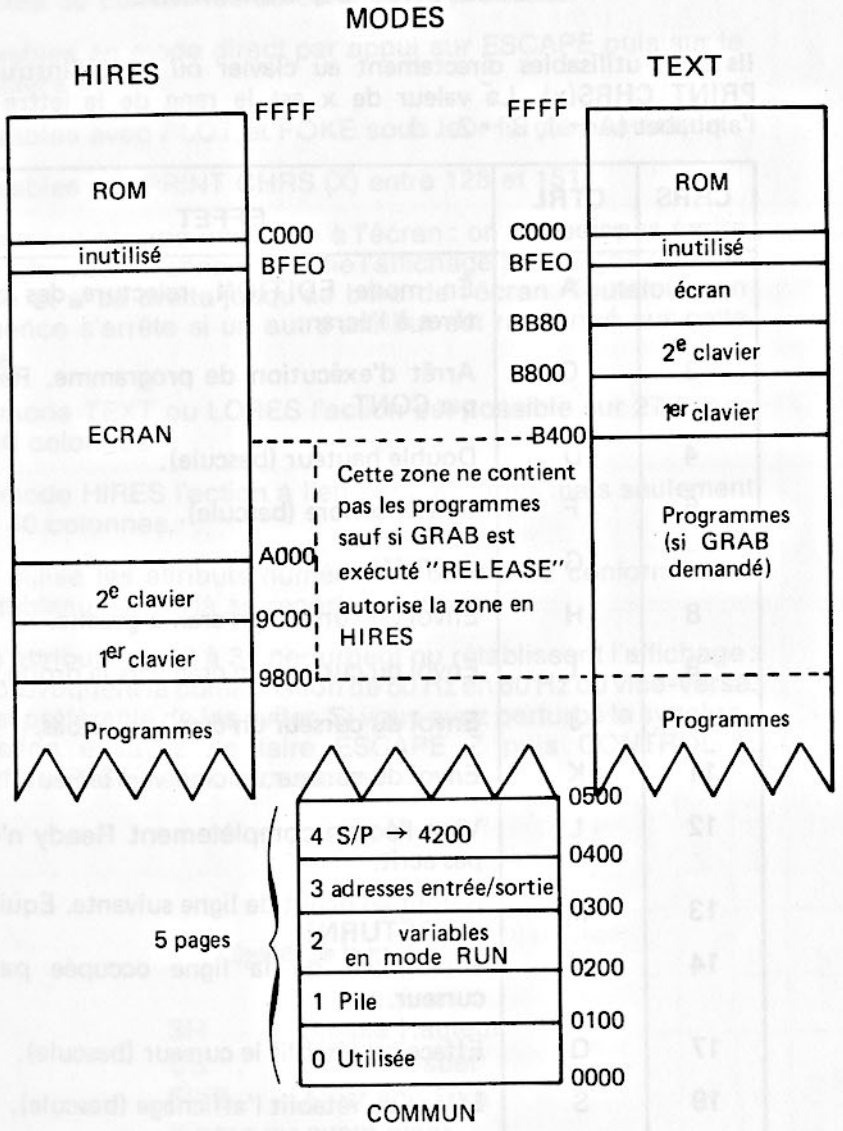
Instruction	Commentaire	Exemple
RESTORE	Ramène le pointeur de données en DATA sur la première donnée.	RESTORE
RETURN	A ne pas confondre avec la touche de même nom. Ecrit en fin de sous-programme, renvoie à l'instruction qui suit le GOSUB ayant provoqué le branchement.	RETURN
RIGHT$	Sous chaîne droite. Met dans D$ les N caractères de droite de A$.	D$ = RIGHT$(A$, N)
RND	Fournit un nombre pseudo-aléatoire. (Une imitation du hasard). Si X ⟩=1, A entre **0** et 1 (1 exclus) Si X =0, A est le même qu'au précédent tirage. Si X ⟨0, A prend toujours la même valeur pour un X donné.	A = RND(1)*6 A = RND(X)
RUN	Lance l'exécution du programme depuis le début. Lance l'exécution du programme à partir de la ligne 200 Met les variables à 0 ou à vide.	RUN RUN 200
SCRN	Fournit le code ASCII du caractère présent à l'écran, à la position X, Y et ceci en TEXT ou LORES.	SCRN (X,Y)
SGN	Donne −1 si l'argument est négatif 0 s'il est nul, 1 s'il est positif. (ici X − Y est l'argument).	Z = SGN (X−Y)
SHOOT	Produit le bruit d'un coup de feu.	SHOOT
SIN	Fournit le sinus de l'angle N. (N en radians).	A = SIN (N)
SOUND	Produit un son ou un bruit, Canal Période, Volume.	SOUND C, P, V
SPC	Dans une instruction PRINT, espace de N, deux écritures successives. (N de 0 à 255).	? "EH" SPC (N) "HUM"
SQR	Donne la racine carrée de N.	A = SQR (N)

Instruction	Commentaire	Exemple
STOP	Interrompt l'exécution d'un programme. Reprise possible avec CONT.	STOP
STR$	Convertit un nombre en une chaine	N$ = STR$ (N)
TAB	Dans une instruction PRINT, bouge le curseur de N espaces à compter du bord gauche de l'écran.	? TAB (N+12)
TAN	Fournit la tangente de N. (N en radians).	A = TAN(N)
TEXT	Efface l'écran. Remet en mode texte.	TEXT
TROFF	Interrompt l'effet de TRON. (mode programme).	TROFF
TRON	En mode programme, permet la recherche des erreurs, provoque l'écriture à l'écran de chaque numéro de ligne en cours d'exécution.	TRON
TRUE	Booléen. Vaut −1 (correspond à VRAI).	TRUE
UNTIL	Fin de boucle commencée par REPEAT, est suivi d'un test. (Voir REPEAT).	UNTIL
USR	Transmet la valeur N à une fonction écrite en langage machine. (Voir chapitre 13).	USR (N)
VAL	Convertit une chaîne en sa valeur numérique. Si le premier caractère est alphabétique on obtient 0.	A = VAL (N$)
WAIT	Pause. La durée de l'arrêt est N fois 10 ms. WAIT 50 arrête 500 millisecondes soit 1/2s.	WAIT N WAIT 50

Instruction	Commentaire	Exemple
ZAP	Produit un son de fiction (arme à laser).	ZAP
	Les instructions doivent être écrites en majuscules. En outre on dispose des signes usuels : + – * / ↑ ⟨ ⟩ = ( ) ; ,	
	: (séparateur d'instruction et non signe de division)	
	Deux autres signes ! et & sont prévus (voir chapitre 13).	
	En mode édition :	
EDIT	Ecrit la ligne 20. Met le curseur devant 20.. On peut alors recopier, par CTRLA, modifier en écrivant, insérer avec la flèche ←, ↑, ou ↓, éviter avec la flèche →, effacer avec DEL, enfin valider par [RETURN]	EDIT 20
POS	Donne la position horizontale du curseur	POS (O)

Appendice A

CARTE de la MEMOIRE d'ORIC 1. (48K)



N.B. : 1) A part la ROM pour les modèles 16K, les adresses s'obtiennent en soustrayant 8000 (hex) à celles du modèle 48K.

2) Adresses indiquées en hexadécimal.

APPENDICE B

CARACTERES DE COMMANDE

Ils sont utilisables directement au clavier ou avec l'instruction PRINT CHRS(x). La valeur de x est le rang de la lettre dans l'alphabet (A → 1, B → 2. . .).

CHRS x	CTRL caractère	EFFET
	A	En mode EDITION, relecture des caractères à l'écran.
3	C	Arrêt d'exécution de programme. Reprise par CONT.
4	D	Double hauteur (bascule).
6	F	Clavier sonore (bascule).
7	G	Sonnette.
8	H	Envoi du curseur un cran à gauche.
9	I	Envoi du curseur un cran vers la droite.
10	J	Envoi du curseur un cran vers le bas.
11	K	Envoi du curseur un cran vers le haut.
12	L	Vide l'écran complètement. Ready n'est pas écrit.
13	M	Retour en début de ligne suivante. Equivalent de [RETURN]
14	N	Effacement de la ligne occupée par le curseur.
17	Q	Efface ou rétablit le curseur (bascule).
19	S	Efface ou rétablit l'affichage (bascule).
20	T	Commutation du clavier : MAJUSCULES / minuscules (bascule).

Appendice C

ATTRIBUTS

Préfixes de commande ou commande préfixées.

Utilisables en mode direct par appui sur ESCAPE puis sur le caractère indiqué.

Utilisables avec PLOT et POKE sous leur forme numérique.

Utilisables par PRINT CHRS (X) entre 128 et 151.

Un attribut occupe une case à l'écran : on ne peut pas écrire sur cette case. L'attribut modifie l'affichage sur la ligne où il est placé et à sa droite jusqu'au bord de l'écran. Toutefois son influence s'arrête si un autre attribut est rencontré sur cette ligne.

En mode TEXT ou LORES l'action est possible sur 27 lignes et 40 colonnes.

En mode HIRES l'action a lieu sur 200 lignes mais seulement sur 40 colonnes.

Fill utilise les attributs numériques de 0 à 22 conformément au tableau : au-delà se reporter au texte.

Les attributs de 23 à 31 perturbent ou rétablissent l'affichage : ils provoquent la commutation de 50 Hz en 60 Hz ou vice-versa. Il est préférable de les éviter. Si vous avez perturbé la synchronisation essayez de faire ESCAPE Z puis CONTROL L. Parfois RESET sera nécessaire.

Légende de la page 164

SH Simple Hauteur
US Clavier Usuel
FIXE Affichage Fixe
(Encre ou avant plan)

DH Double Hauteur
SG Clavier semi-graphique
CLIGN. Affichage Clignotant
(Papier ou arrière plan)

PRINT CHR$	POKE PLOT	ESC	EFFET	PRINT CRHS	POKE PLOT	ESC	EFFET
128	0	@	encre noire	144	16	P	papier noir
129	1	A	encre rouge	145	17	Q	papier rouge
130	2	B	encre verte	146	18	R	papier vert
131	3	C	encre jaune	147	19	S	papier jaune
132	4	D	encre bleu foncé	148	20	T	papier bleu foncé
133	5	E	encre mauve	149	21	U	papier mauve
134	6	F	encre bleu ciel	150	22	V	papier bleu ciel
135	7	G	encre blanche	151	23	W	papier blanc
136	8	H	SH/US/FIXE	152	24	X	TEXT 60 Hz
137	9	I	SH/SG/FIXE	153	25	Y	TEXT 60 Hz
138	10	J	DH/US/FIXE	154	26	Z	TEXT 50 Hz
139	11	K	DH/SG/FIXE	155	27	[	TEXT 50 Hz
140	12	L	SH/US/CLIGN.	156	28	\	GRA 60 Hz
141	13	M	SH/SG/CLIGN.	157	29	]	GRA 60 Hz
142	14	N	DH/US/CLIGN.	158	30	↑	GRA 50 Hz
143	15	O	DH/SG/CLIGN.	159	31	←	GRA 50 Hz

APPENDICE D

CODES A.S.C.I.I.

Code	Caractère	Code	Caractère	Code	Caractère	Code	Caractère
32	espace	64	@	79	O	96	©
33	!	65	A	80	P	97	a
34	"	66	B	81	Q	98	b
35	#	67	C	82	R	99	c
36	$	68	D	83	S	100	d
37	%	69	E	84	T	101	e
38	&	70	F	85	U	102	f
39	'	71	G	86	V	103	g
40	(	72	H	87	W	104	h
41	)	73	I	88	X	105	i
42	*	74	J	89	Y	106	j
43	+	75	K	90	Z	107	k
44	,	76	L	91	[	108	l
45	-	77	M	92	\	109	m
46	.	78	N	93	]	110	n
47	/			94	↑	111	o
48	0			95	←	112	p
49	1			96	£	113	q
50	2			97	©	114	r
51	3			98	a	115	s
52	4			99	b	116	t
53	5			100	c	117	u
54	6			101	d	118	v
55	7			102	e	119	w
56	8			103	f	120	x
57	9			104	g	121	y
58	:			105	h	122	z
59	;			106	i	123	f
60	<			107	j	124	—
61	=			108	k	125	]
62	>			109	l		
63	?			110	m		
				111	n		
				112	o		
				113	p		
				114	q		
				115	r		
				116	s		
				117	t		
				118	u		
				119	v		
				120	w		
				121	x		
				122	y		
				123	z		
				124	f		
				125	—		

Appendice E

TABLE DE CONVERSION

Décimal / Binaire / Hexadécimal

DEC	BINAIRE	HEX
0	00000000	00
1	00000001	01
2	00000010	02
3	00000011	03
4	00000100	04
5	00000101	05
6	00000110	06
7	00000111	07
8	00001000	08
9	00001001	09
10	00001010	0A
11	00001011	0B
12	00001100	0C
13	00001101	0D
14	00001110	0E
15	00001111	0F
16	00010000	10
17	00010001	11
18	00010010	12
19	00010011	13
20	00010100	14
21	00010101	15
22	00010110	16
23	00010111	17
24	00011000	18
25	00011001	19
26	00011010	1A
27	00011011	1B
28	00011100	1C
29	00011101	1D

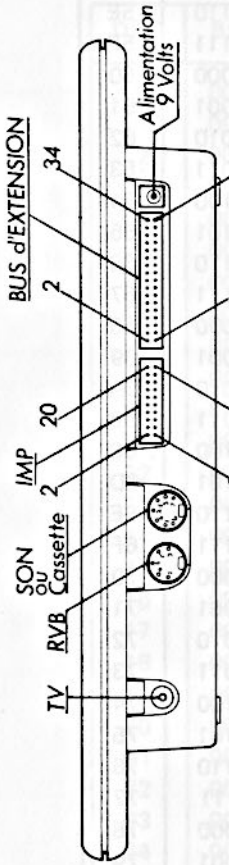
DEC	BINAIRE	HEX
30	00011110	1E
31	00011111	1F
32	00100000	20
33	00100001	21
34	00100010	22
35	00100011	23
36	00100100	24
37	00100101	25
38	00100110	26
39	00100111	27
40	00101000	28
41	00101001	29
42	00101010	2A
43	00101011	2B
44	00101100	2C
45	00101101	2D
46	00101110	2E
47	00101111	2F
48	00110000	30
49	00110001	31
50	00110010	32
51	00110011	33
52	00110100	34
53	00110101	35
54	00110110	36
55	00110111	37
56	00111000	38
57	00111001	39
58	00111010	3A
59	00111011	3B

DEC	BINAIRE	HEX
60	00111100	3C
61	00111101	3D
62	00111110	3E
63	00111111	3F
64	01000000	40
65	01000001	41
66	01000010	42
67	01000011	43
68	01000100	44
69	01000101	45
70	01000110	46
71	01000111	47
72	01001000	48
73	01001001	49
74	01001010	4A
75	01001011	4B
76	01001100	4C
77	01001101	4D
78	01001110	4E
79	01001111	4F
80	01010000	50
81	01010001	51
82	01010010	52
83	01010011	53
84	01010100	54
85	01010101	55
86	01010110	56
87	01010111	57
88	01011000	58
89	01011001	59
90	01011010	5A
91	01011011	5B
92	01011100	5C
93	01011101	5D

DEC	BINAIRE	HEX
94	01011110	5E
95	01011111	5F
96	01100000	60
97	01100001	61
98	01100010	62
99	01100011	63
100	01100100	64
101	01100101	65
102	01100110	66
103	01100111	67
104	01101000	68
105	01101001	69
106	01101010	6A
107	01101011	6B
108	01101100	6C
109	01101101	6D
110	01101110	6E
111	01101111	6F
112	01110000	70
113	01110001	71
114	01110010	72
115	01110011	73
116	01110100	74
117	01110101	75
118	01110110	76
119	01110111	77
120	01111000	78
121	01111001	79
122	01111010	7A
123	01111011	7B
124	01111100	7C
125	01111101	7D
126	01111110	7E
127	01111111	7F

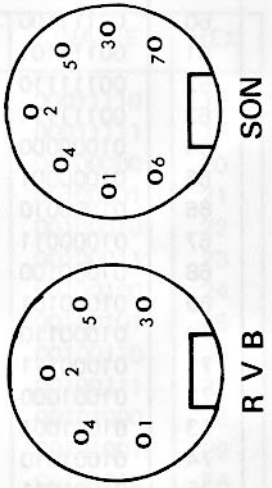
Appendice F

LES SORTIES DE L'ORIC



BUS d'EXTENSION • **IMPRIMANTE** **R V B** **SON ou CASSETTE**

Carte	1	2	ROMDIS	STB	1	2		1 Rouge	1 Sortie
Φ2	3	4	Reset	D0	3	4		2 Vert	2 Masse
I/O	5	6	I/O Control	D1	5	6		3 Bleu	3 Entrée
R/W	7	8	IRQ	D2	7	8		4 Synchro	4 Son } 8912
D2	9	10	D0	D3	9	10	Masse	5 Masse	5 Son } 8912
A3	11	12	D1	D4	11	12			6 Contact du relais
A0	13	14	D6	D5	13	14			7 Contact du relais
A1	15	16	D3	D6	15	16			
A2	17	18	D4	D7	17	18			
D5	19	20	A4	ADK	19	20			
A5	21	22	D7						
A6	23	24	A15						
A7	25	26	A14						
A8	27	28	A13						
A9	29	30	A12						
A10	31	32	A11						
+5V	33	34	Terre ou masse						



Appendice G

FONCTIONS COMPOSÉES

Ces fonctions n'existent pas, au départ, sur l'ORIC.

Elles peuvent être créées par l'instruction DEF FN.

Exemple :

```
DEF  COT(x) = 1 / TAN(x)        pour la cotangente.
```

```
ARC SIN(X)= ATN (X/SQR(– X*X + 1))
ARC COS(X)= – ATN (X/SQR(– X* X + 1)) + 1.5708
ARC COT(X)= – ATN(x) + 1.5708
```

```
SINH(X)= (EXP(X)–EXP(–X))/2                    sinus hyperbolique
COS H(X)= (EXP(X)+EXP(–X))/2                   cosinus  "   "
TANH(X)= –EXP(–X)/(EXP(X)–EXP(–X))             *2 + 1
                                               tangente "   "
COTANH(X) = EXP(–X)/(EXP(X)–EXP(–X))           *2 + 1
                                               cotangente "  "
```

```
ARGSINH(X)=LOG(X + SQR(X*X+1))
ARGCOSH(X)=LOG(X+SQR(X*X –1))
ARGTANH(X)=LOG(1 + X)/1 – X))/2
ARGCOTANH(X)=LOG((X + 1)/(X – 1)/2
```

} Réciproques des 4 fonctions précédentes.

A modulo B s'obtient ainsi :

```
MOD(A) = INT ((A/B – INT (A/B))*B+0.05)*SGN(A/B)
```

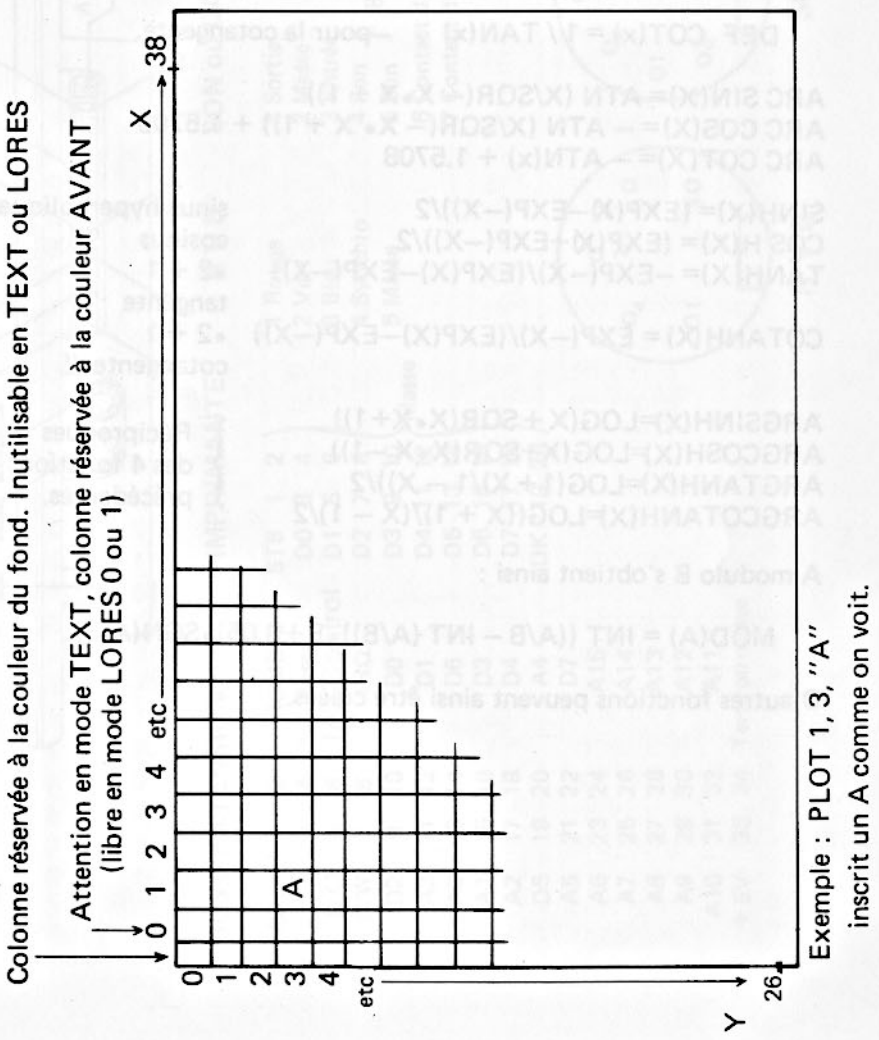
D'autres fonctions peuvent ainsi être créées.

Appendice H

Grille d'écran Texte : **TEXT**

LORES 0

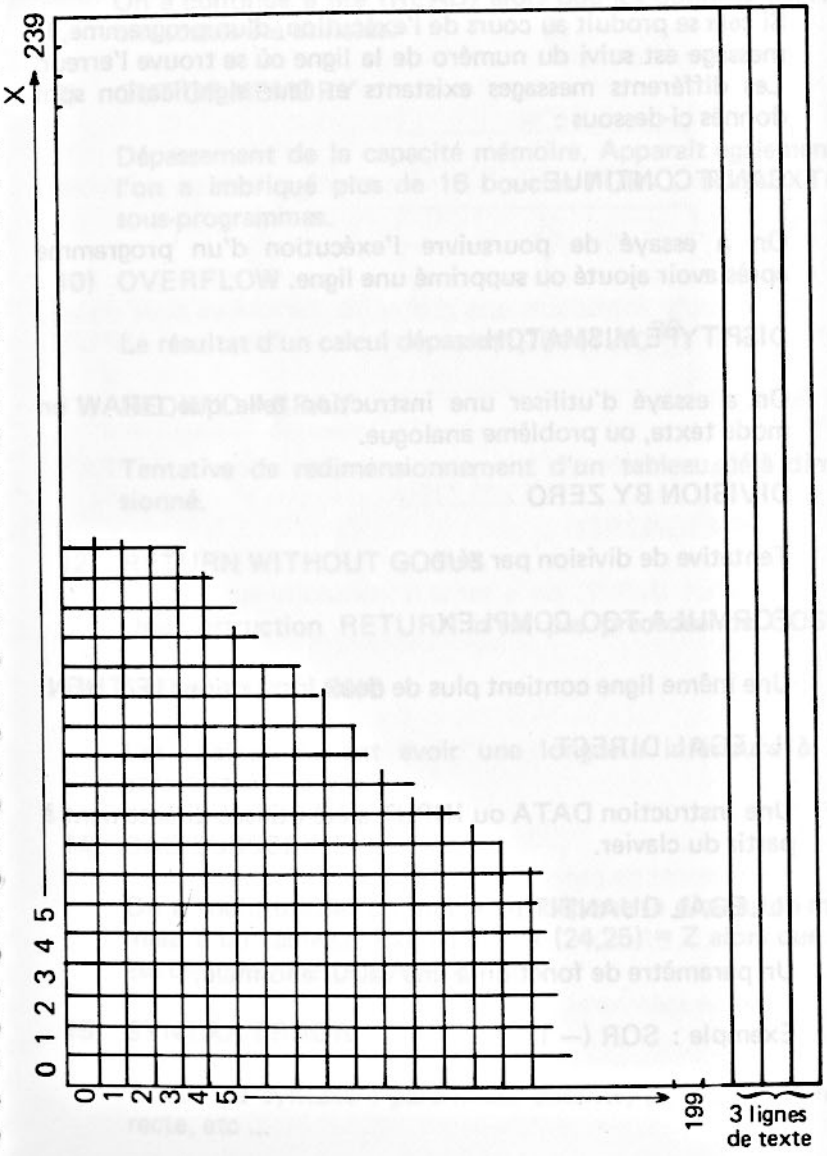
LORES 1



Appendice I

GRILLE D'ÉCRAN GRAPHIQUE, HAUTE DÉFINITION

HIRES



Appendice J

Messages d'erreur :

Si ORIC ne peut pas traiter un ordre ou une information, il fait apparaître un message d'erreur.

Si cela se produit au cours de l'exécution, d'un programme, le message est suivi du numéro de la ligne où se trouve l'erreur. Les différents messages existants et leur signification sont donnés ci-dessous :

1) CAN'T CONTINUE

On a essayé de poursuivre l'exécution d'un programme après avoir ajouté ou supprimé une ligne.

2) DISP TYPE MISMATCH

On a essayé d'utiliser une instruction telle que DRAW en mode texte, ou problème analogue.

3) DIVISION BY ZERO

Tentative de division par zéro.

4) FORMULA TOO COMPLEX

Une même ligne contient plus de deux instructions IF/THEN.

5) ILLEGAL DIRECT

Une instruction DATA ou INPUT a été utilisée directement à partir du clavier.

6) ILLEGAL QUANTITY

Un paramètre de fonction à une valeur anormale.

Exemple : SQR (– 1)

7) NEXT WITHOUT FOR

Une instruction NEXT n'est pas précédée de FOR.

8) OUT OF DATA

On a continué à lire (READ) alors que les données (DATA) ont toutes été utilisées.

9) OUT OF MEMORY

Dépassement de la capacité mémoire. Apparaît également si l'on a imbriqué plus de 16 boucles FOR ... TO/NEXT ou sous-programmes.

10) OVERFLOW

Le résultat d'un calcul dépasse $|1.70141 * 10^{38}|$

11) REDIM'D ARRAY

Tentative de redimensionnement d'un tableau déjà dimensionné.

12) RETURN WITHOUT GOSUB

Une instruction RETURN n'est pas précédée de GOSUB.

13) STRING TOO LONG

Les chaînes doivent avoir une longueur inférieure à 256 caractères.

14) BAD SUBSCRIPT

On a voulu utiliser un indice qui dépasse la dimension maximale d'un tableau. Exemple : A (24,25) = Z alors que A a été dimensionné DIM A (4,4).

15) SYNTAX ERROR

Erreur de syntaxe : parenthèse absente, ponctuation incorrecte, etc ...

16) TYPE MISMATCH

On a voulu donner une valeur numérique à une chaîne de caractères ou inversement.

17) UNDEF'D STATEMENT

Une instruction GOTO, THEN ou GOSUB renvoie à une ligne qui n'existe pas.

18) UNDEF'D FUNCTION

On appelle une fonction qui n'a pas été définie au préalable.

19) REDO FROM START

On a voulu introduire une chaîne de caractères alors qu'un nombre est demandé. Retourne à l'instruction INPUT.

20) BAD UNTIL

Une instruction UNTIL n'est pas précédée de REPEAT.

21) EXTRA IGNORED

Lors d'un INPUT, on a tenté d'introduire après une virgule une donnée de plus que prévu.

22) PRINTER ERROR

Erreur dans la liaison avec l'imprimante.

23) TOO LARGE

Dépassement de capacité de mémoire programme.

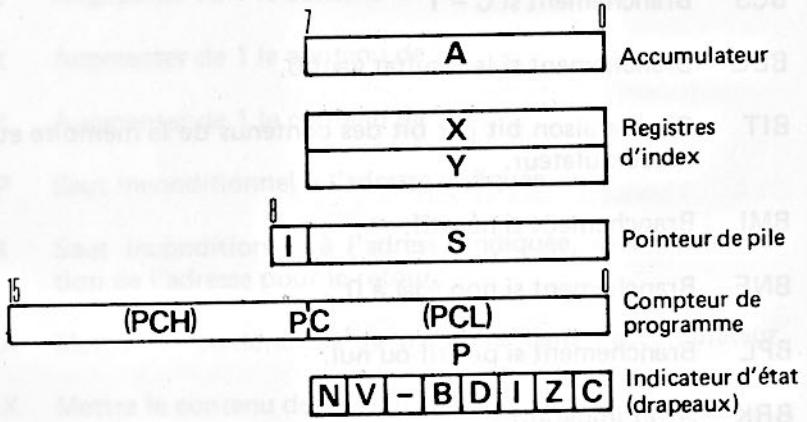
24) MEMORY ERROR

Erreur en mémoire.

Appendice K

LE 6502

Registre du microprocesseur 6502



N	Signe moins (nombre négatif)
V	Débordement
–	
B	Break (interruption)
D	Décimal
I	Inhibition d'interruption
Z	Indicateur de zéro
C	Indicateur de retenue

MNEMONIQUES

ADC Addition du contenu de la mémoire au contenu de l'accumulateur, avec retenue.

AND ET logique (entre mémoire et accumulateur).

ASL Décaler d'un cran à gauche.

BCC Branchement si C = 0

BCS Branchement si C = 1

BEQ Branchement si le résultat vaut 0.

BIT Comparaison bit par bit des contenus de la mémoire et de l'accumulateur.

BMI Branchement si négatif.

BNE Branchement si non égal à 0.

BPL Branchement si positif ou nul.

BRK Arrêt impératif.

BVC Branchement si non débordement.

BVS Branchement en cas de débordement.

CLC Annulation de retenue.

CLD Annulation de mode décimal.

CLI Autoriser les interruptions.

CLV Annuler le drapeau de débordement.

CMP Comparer mémoire et accumulateur.

CPX Comparer mémoire et X.

CPY Comparer mémoire et Y.

DEC Diminuer de 1 le contenu de la mémoire.

DEX Diminuer de 1 le contenu de X

DEY Diminuer de 1 le contenu de Y

EOR OU exclusif (entre mémoire et accumulateur).

INC Augmenter de 1 le contenu de la mémoire.

INX Augmenter de 1 le contenu de X.

INY Augmenter de 1 le contenu de Y.

JMP Saut inconditionnel à l'adresse indiquée.

JSR Saut inconditionnel à l'adresse indiquée, avec mémorisation de l'adresse pour le retour.

LDA Mettre le contenu de la mémoire dans l'accumulateur.

LDX Mettre le contenu de la mémoire dans X.

LDY Mettre le contenu de la mémoire dans Y.

LSR Décaler d'un cran à droite.

NOP Pas d'opération.

ORA OU inclusif (entre mémoire et accumulateur).

PHA Empiler A.

PHP Empiler P.

PLA Retirer A de la pile.

PLP Retirer P de la pile.

ROL Permutation circulaire d'un cran à gauche.

ROR Permutation circulaire d'un cran à droite.

RTI Retour après interruption.

RTS Retour après JSR (saut à un sous-programme).

SBC Soustraction du contenu de la mémoire au contenu de l'accumulateur, avec retenue.

SEC Mettre la retenue à 1.

SED Mettre en mode décimal.

SEI Interdire les interruptions.

STA Transfère dans la mémoire le contenu de l'accumulateur.

STX Transfère dans la mémoire le contenu de X.

STY Transfère dans la mémoire le contenu de Y.

TAX Transfère A dans X.

TAY Transfère A dans Y.

TSX Transfère S dans X.

TXA Transfère X dans A.

TXS Transfère X dans S.

TYA Transfère Y dans A.

LANGAGE MACHINE

Mnémonique	Fonction	Langage d'assemblage	Mode d'adressage	Nombre d'octets	Code HEX	Drapeaux
ADC Addition du contenu mémoire à l'accumulateur avec retenue	A ← A+M+C	ADC #opé ADC opér ADC opér, X ADC opér ADC opér, X ADC opér, Y ADC (opér, X) ADC (opér), Y	immédiat zéro-page zéro-page, X absolu absolu, X absolu, Y (indirect, X) (indirect), Y	2 2 2 3 3 3 2 2	69 65 75 6D 7D 79 61 71	NVZC
AND ET logique entre accumulateur et mémoire	A ← A∧M	AND #opér AND opér AND opér, X AND opér AND opér, X AND opér, Y AND (opér, X) AND (opér), Y	immédiat zéro-page zéro-page, X absolu absolu, X absolu, Y (indirect, X) (indirect), Y	2 2 2 3 3 3 2 2	29 25 35 2D 3D 39 31 31	NZ
ASL Décalage d'un cran à gauche (0 dernier bit à droite) (le bit de gauche va dans C)		ASL A ASL opér ASL opér, X ASL opér ASL opér, X	accumulateur zéro-page zéro-page, X absolu absolu, X	1 2 2 3 3	0A 06 16 0E 1E	NZC
BCC Branchement si C = 0	Br. si C = 0	BCC opér	relatif	2	90	
BCS Branchement si C = 1	Br. si C = 1	BCS opér	relatif	2	B0	
BEQ Branchement si Z = 1	Br. si Z = 1	BEQ opér	relatif	2	F0	
BIT Comparaison bit par bit	$Z \leftarrow \overline{A \wedge M}$ $N \leftarrow M_7$ $V \leftarrow M_6$	BIT * opér BIT * opér	zéro-page absolu	2 3	24 2C	NVZ
BMI Branchement si N = 1	Br. si N = 1	BMI opér	relatif	2	30	
BNE Branchement si Z = 0 (résultat non nul)	Br. si Z = 0	BNE opér	relatif	2	D0	
BPL Branchement si positif ou nul	Br. si N = 0	BPL opér	relatif	2	10	
BRK Arrêt forcé	empile PC + 2 et P	BRK*	implicite	1	00	B I ← 1
BVC Branchement si non débordement	Br. si V = 0	BVC opér	relatif	2	50	
BVS Branchement si débordement	Br. si V = 1	BVS opér	relatif	2	70	
CLC Annulation de retenue	C ← 0	CLC	implicite	1	18	C ← 0

Mnémonique	Fonction	Langage d'assemblage	Mode d'adressage	Nombre d'octets	Code HEX	Drapeaux
CLD Annulation du mode décimal	D ←0	CLD	implicite	1	D8	D ←0
CLI Autorisation des interruptions.	I ←0	CLI	implicite	1	58	I ←0
CLV Annuler le drapeau de débordement	V ←0	CLV	implicite	1	B8	V ←0

CMP
Comparer mémoire et accumulateur
	A – M	CMP #opér
		CMP opér
		CMP opér, X
		CMP opér
		CMP opér, X
		CMP opér, Y
		CMP (opér, X)
		CMP (opér), Y
			immédiat
			zéro-page
			zéro-page, X
			absolu
			absolu, X
			absolu, Y
			(indirect, X)
			(indirect), Y
				2
				2
				2
				3
				3
				3
				2
				2
					C9
					C5
					D5
					CD
					DD
					D9
					C1
					D1
						N Z C

CPX Comparer mémoire et registre X	X – M	CPX #opér CPX opér CPX opér	immédiat zéro-page absolu	2 2 3	E0 E4 EC	N Z C
CPY Comparer mémoire et registre Y	Y – M	CPY #opér CPY opér CPY opér	immédiat zéro-page absolu	2 2 3	C0 C4 CC	N Z C
DEC Diminuer de 1 le contenu de la mémoire	M ←M – 1	DEC opér DEC opér, X DEC opér, DEC opér, X	zéro-page zéro-page, X absolu absolu, X	2 2 3 3	C6 D6 CE DE	N Z
DEX Diminuer de 1 le contenu de X.	X ←X – 1	DEX	implicite	1	CA	N Z
DEY Diminuer de 1 le contenu de Y	Y ←Y – 1	DEY	implicite	1	88	N Z
EOR Ou exclusif entre mémoire et accumulateur	A ←A V M	EOR #opér EOR opér EOR opér, X EOR opér, EOR opér, X EOR opér, Y EOR (opér, X) EOR (opér), Y	immédiat zéro-page zéro-page, X absolu absolu, X absolu, Y (indirect, X) (indirect), Y	2 2 2 3 3 3 2 2	49 45 55 4D 5D 59 41 51	N Z
INC Incrémenter de 1 la mémoire	M ←M + 1	INC opér INC opér, X INC opér INC opér, X	zéro-page zéro-page, X absolu absolu, X	2 2 3 3	E6 F6 EE FE	N Z
INX Incrémenter X de 1	X ←X + 1	INX	implicite	1	E8	N Z

Mnémonique	Fonction	Langage d'assemblage	Mode d'adressage	Nombre d'octets	Code HEX	Drapeaux
INY Incrémenter Y de 1	Y ← Y + 1	INY	implicite	1	C8	N Z
JMP Saut inconditionnel à une adresse	PCL ← (PC+1) PCH ← (PC+2)	JMP opér JMP (opér)	absolu indirect	3 3	4C 6C	
JSR Saut inconditionnel à un sous programme	Empile PC+2 PCL ← (PC+1) PCH ← (PC+2)	JSR opér	absolu	3	20	

LDA
Charge l'accumulateur
avec la mémoire
	A ← M	LDA #opér
		LDA opér
		LDA opér, X
		LDA opér,
		LDA opér, X
		LDA opér, Y
		LDA (opér, X)
		LDA (opér), Y
			immédiat
			zéro-page
			zéro-page, X
			absolu
			absolu, X
			absolu, Y
			(indirect, X)
			(indirect), Y
				2
				2
				2
				3
				3
				3
				2
				2
					A9
					A5
					B5
					AD
					BD
					B9
					A1
					B1
						N Z

LDX Charger x avec la mémoire	X ← M	LDX #opér LDX opér LDX opér, Y LDX opér LDX opér, Y	immédiat zéro-page zéro-page, Y absolu absolu, Y	2 2 2 3 3	A2 A6 B6 AE BE	N Z
LDY Charger Y avec la mémoire	Y ← M	LDY #opér LDY opér LDY opér, X LDY opér LDY opér, X	immédiat zéro-page zéro-page, X absolu absolu, X	2 2 2 3 3	A0 A4 B4 AC BC	Z
LSR Décalage d'un cran à droite (0 entre à gauche) (le bit de droite va dans C)		LSR A LSR opér LSR opér, X LSR opér LSR opér, X	accumulateur zéro-page zéro-page, X absolu absolu, X	1 2 2 3 3	4A 46 56 4E 5E	Z C N ← 0
NOP Instruction muette	aucune	NOP	implicite	1	EA	
ORA OU inclusif entre mémoire et accumulateur	A ← AVM	ORA #opér ORA opér, ORA opér, X ORA opér ORA opér, X ORA opér, Y ORA (opér, X) ORA (opér), Y	immédiat zéro-page zéro-page, X absolu absolu, X absolu, Y (indirect, X) (indirect), Y	2 2 2 3 3 3 2 2	09 05 15 0D 1D 19 01 11	N Z
PHA Empiler A	empile A S ← S − 1	PHA	implicite	1	48	
PHP Empiler P	Empile P S ← S − 1	PHP	implicite	1	08	

Mnémonique	Fonction	Langage d'assemblage	Mode d'adressage	Nombre d'octets	Code HEX	Drapeaux
PLA Dépiler A	A ←(Pile) S ←S + 1	PLA	implicite	1	68	N Z
PLP Dépiler P	P ←(Pile) S ←S − 1	PLP	implicite	1	28	rétablis
ROL Permutation circulaire d'un cran à gauche (le bit de C entre à droite) (le bit de gauche va dans C)		ROL A ROL opér ROL opér, X ROL opér, ROL opér, X	accumulateur zéro-page zéro-page, X absolu absolu, X	1 2 2 3 3	2A 26 36 2E 3E	N Z C
ROR Permutation circulaire d'un cran à droite (le bit de C entre à gauche) (le bit de droite va dans C)		ROR A ROR opér ROR opér, X ROR opér ROR opér, X	accumulateur zéro-page zéro-page, X absolu absolu, X	1 2 2 3 3	6A 66 76 6E 7E	N Z C
RTI Retour après interruption	P ←(Pile) S ←S + 1 PCL ←(Pile) S ←S + 1 PCH ←(Pile) S ←S + 1	RTI	implicite	1	40	rétablis
RTS Retour après un sous-programme	PC ←(Pile) PC ←PC + 1 S ←S + 2	RTS	implicite	1	60	

SBC Soustraction du contenu de la mémoire à l'accumulateur avec retenue	A←A−M−C	SBC# opér SBC opér SBC opér, X SBC opér SBC opér, X SBC opér, Y SBC (opér, X) SBC (opér), Y	immédiat zéro-page zéro-page, X absolu absolu, X absolu, Y (indirect, X) (indirect), Y	2 2 2 3 3 3 2 2	E9 E5 F5 ED FD F9 E1 F1	N V Z C
SEC Mettre à 1 la retenue	C ←1	SEC	implicite	1	38	C ←1
SED Mettre en mode décimal	D ←1	SED	implicite	1	F8	D ←1
SEI Interdiction des interruptions	I ←1	SEI	implicite	1	78	I ←1
STA Recopier dans la mémoire le contenu de l'accumulateur	M ←A	STA opér STA opér, X STA opér STA opér, X STA opér, Y STA (opér, X) STA (opér), Y	zéro-page zéro-page, X absolu absolu, X absolu, Y (indirect, X) (indirect), Y	2 2 3 3 3 2 2	85 95 8D 9D 99 81 91	
STX Recopier en mémoire le contenu de X	M ←X	STX opér STX opér STX opér, Y	absolu zéro-page zéro-page, Y	3 2 2	8E 86 96	

Mnémonique	Fonction	Langage d'assemblage	Mode d'adressage	Nombre d'octets	Code HEX	Drapeaux
STY Recopier en mémoire le contenu de Y	M ← Y	STY opér STY opér STY opér, X	absolu zéro-page zéro-page, X	3 2 2	8C 84 94	
TAX Transférer A dans X	X ← A	TAX	implicite	1	AA	N Z
TAY Transférer A dans Y	Y ← A	TAY	implicite	1	A8	N Z
TSX Transférer S dans X	X ← S	TSX	implicite	1	BA	N Z
TXA Transférer X dans l'accumulateur.	A ← X	TXA	implicite	1	8A	N Z
TXS Transférer X dans S	S ← X	TXS	implicite	1	9A	
TYA Transférer Y dans A	A ← Y	TYA	implicite	1	98	N Z

APPENDICE L

QUELQUES IDÉES DE PROGRAMMES

Kaléidoscope

```
10 TEXT : CLS : PRINT CHR$ (17)
20 FOR J = 0 TO 27
30 FOR I = 0 TO 39
40 POKE # BB80+J*40+I, 16
50 NEXT I, J
60 A=–1 : B=A : F=16 : X=14 : Y=21
70 A=A+1 : IF A = 15 THEN A = 0
80 B=B+1 : IF B = 19 THEN B = 0
90 G=G+1 : IFG(10 GOTO 90
95 G =0 : F = F+1 : IF F = 24 THEN F = 16
100 POKE#BB80 + (X – A)* 40+Y–B, F
110 POKE#BB80 + (X – A) *40 +Y+B, F
120 POKE#BB80 + (X + A)* 40 +Y–B, F
130 POKE#BB80 + (X + A)* 40+Y+B, F
140 GOTO 70
```

Gammes

```
10 X = 1 : I = 1
20 PLAY 1, 0, 0, 0
30 MUSIC 1, K, I, 6
40 I = I+X : IF I = 13 THEN I = 1 : K = K + X
50 IF K = 7 THEN X = – X : K = 6
60 IF I = 0 THEN I = 12 : K = K + X
70 IF I = 2 OR I = 4 OR I = 7 OR I = 9 OR I = 11 THEN 40
80 IF K = –1 THEN RUN
100 A$ = KEY$ : IF A$ (    )  "    " THEN END
120 GOTO 20
```

Variante : supprimer la ligne 70 ; ajouter WAIT 50 en 105.

NOTES PERSONNELLES

NOTES PERSONNELLES

NOTES PERSONNELLES