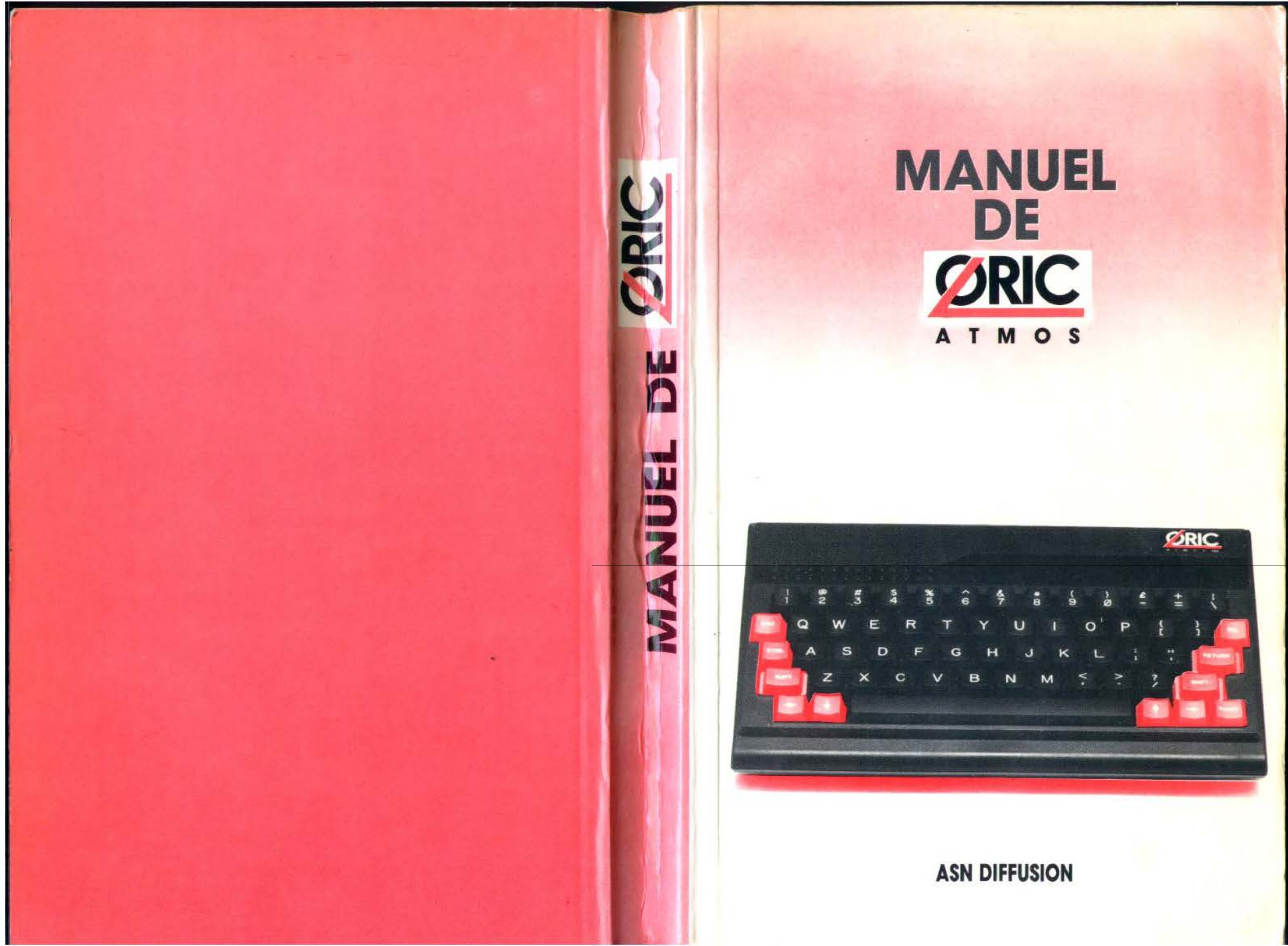
Numérisation http://wsc.n3.net



William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr

MANUEL DE L'ORIC ATMOS

Ian Adamson
Traduction et adaptation française
Jean Pascal DUCLOS

Propriété de ASN DIFFUSION
21, la Haie Griselle
94470 BOISSY-SAINT-LÉGER

Numérisation http://wsc.n3.net

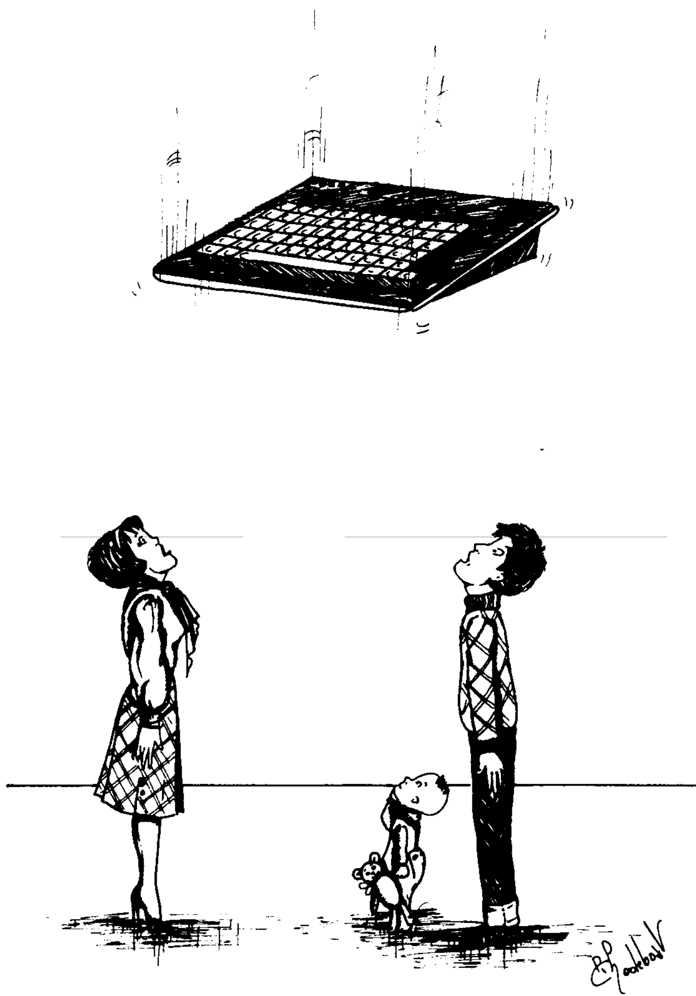
L'édition originale de ce livre est parue sous le titre
The ORIC ATMOS MANUEL
© Ian Adamson, 1984
PAN BOOKS LTD
ISBN 0330 28482 7
L'édition française est parue sous le titre
MANUEL DE L'ORIC ATMOS
enregistré sous le numéro 7124
© ASN DIFFUSION, 1984
Tous droits de reproduction interdits pour tous pays y compris l'URSS

INTRODUCTION

Chapitre 1 - Prise de contact
Chapitre 2 - Initiation au langage
Chapitre 3 - Programme en Basic
Chapitre 4 - Des boucles sans comparaison
Chapitre 5 - Suivons le parcours des mémoires
Chapitre 6 - Enregistrement et données
Chapitre 7 - Graphique et couleur
Chapitre 8 - Musique et sons
Chapitre 9 - Les mots-clés du BASIC
Chapitre 10 - Initiation au code machine
Chapitre 11 - Les entrées/sorties
Annexes **1** - Tables des codes ASCII
2 - Code "Escape" (changement de code)
3 - Messages d'erreurs
4 - Grilles d'écran
5 - Plan d'implantation de la mémoire
6 - Table de conversion
7 - Utilisation de l'imprimante ORIC RCP - 40
8 - Registre du microprocesseur 65Ø2
9 - Adresses et les routines de la ROM
10 - Circuits d'entrées/sorties
11 - Connecteurs de l'ATMOS
12 - Mots réservés au BASIC ainsi que leur codes

William Saint-Cricq / wsaintcr@wanadoo.fr Page 3 sur 163 15/08/2000

Numérisation http://wsc.n3.net



Introduction

Bienvenue dans l'univers du micro-ordinateur ORIC ATMOS ! Si vous lisez ces lignes, c'est sans doute que vous êtes l'heureux possesseur d'un ATMOS et ce livre va vous permettre de l'utiliser au mieux de toutes ses possibilités. Haute résolution graphique, sons, sur trois canaux, couleurs et quantité d'autres effets font de l'ORIC ATMOS un des micro-ordinateurs les plus évolués à ce jour.

Ce livre contient tout ce dont vous avez besoin pour tirer le meilleur parti de votre ATMOS, apprendre le langage machine, vous initier à la programmation en langage BASIC, et découvrir la fascination de l'informatique. Si vous partez de zéro, souvenez-vous lorsque vous rencontrerez quelques difficultés que cet appareil est très puissant et d'un emploi complexe. Vous ne pouvez pas espérer en avoir fait le tour en quelques heures.

Avec un peu de patience, de l'application, en vous exerçant avec les exemples que nous donnons, vous serez bientôt un expert du clavier. Nous présenterons peu à peu le jargon des informaticiens. Il est indispensable d'en passer par là, car un ordinateur est plein de ressources techniques auxquelles il faut être tant soit peu initié pour mieux comprendre la programmation. De prime abord entendre parler de bits, d'octets, de K, de puce etc. est troublant mais ce sont des termes indispensables pour notre exposé.

Ce manuel commence par vous expliquer clairement la mise en œuvre et les caractéristiques essentielles. Il vous indique les possibilités d'extension. Ensuite vos premiers pas sont guidés, faisant d'une pierre deux coups, vous apprenez le maniement du clavier et des rudiments de programmation. Puis le BASIC est approfondi. Un chapitre est consacré aux opérations de sauvegarde sur cassette. Les possibilités graphiques et sonores sont décrites dans des chapitres spéciaux car elles sont complexes. La totalité des termes BASIC est proposée en ordre alphabétique avec leur écriture, leur code, un exemple d'utilisation facilitant la compréhension. Un important chapitre est consacré au langage machine, suivi de renseignements techniques sur les entrées/sorties.

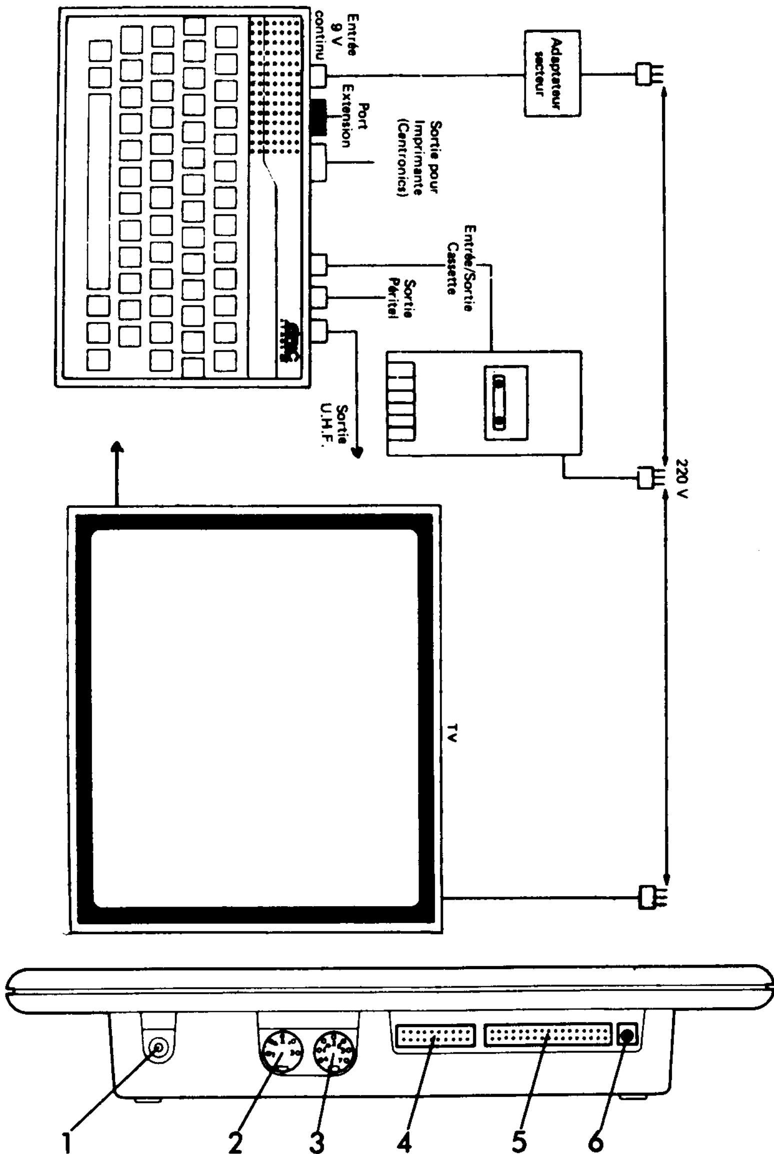
En annexe vous trouverez quantité de renseignements sur les adresses, les routines, les caractéristiques techniques.

Nous sommes certains que l'univers que vous offre l'ATMOS vous procurera un plaisir intarissable, de grandes satisfactions jusqu'à la fascination même.

Entrons dans le monde féerique de l'ATMOS !

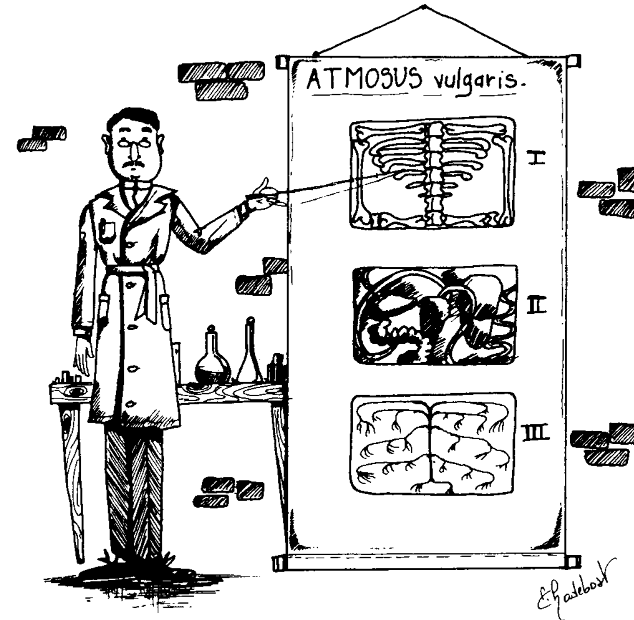
William Saint-Cricq / wsaintcr@wanadoo.fr Page 4 sur 163 15/08/2000

Numérisation http://wsc.n3.net



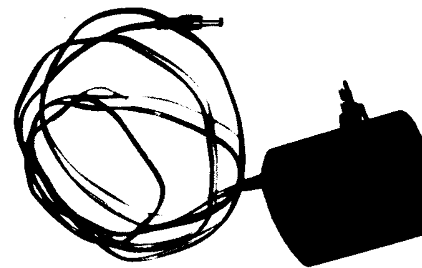
ATTENTION :
Ne pas brancher l'alimentation de l'ORIC sur le bus d'extension, cela provoque une grave panne.

Chapitre 1
Prise de contact



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

La plus à gauche **(POWER)** recevra l'alimentation en courant électrique 9 volts continu délivré par l'adaptateur fourni (A).



15/08/2000
Page 5 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

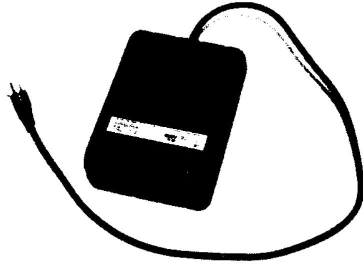
Deux précautions sont à prendre :

• Veillez à ne pas toucher les broches voisines (5) ce qui pourrait endommager gravement votre ORIC. Il est bon de protéger par un cache ces contacts : ruban adhésif ou autre.
• Brancher d'abord l'adaptateur (A) sur le secteur et ensuite seulement introduire le jack dans l'ORIC (en 6) : cela fait office d'interrupteur de mise en marche. En cas de non fonctionnement, sortir le jack, attendre quelques secondes et le rebrancher.

Pour relier l'ORIC à l'écran distinguons les cas :

1) Vous disposez d'un téléviseur couleur ancien, non muni de la prise **PERITÉLÉVISION**.

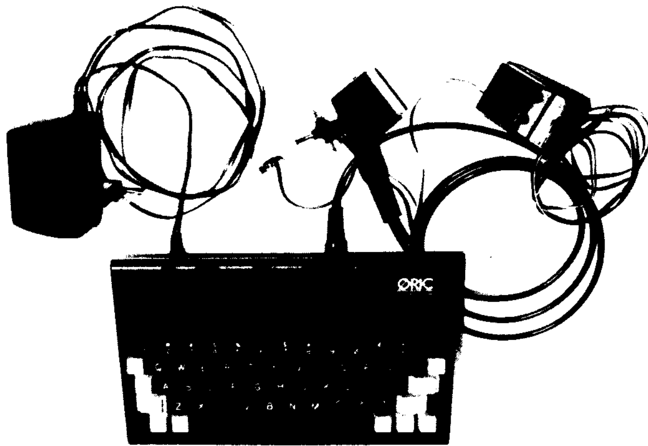
Il vous faut un adaptateur particulier UHF (option) (B).



Précaution : brancher d'abord dans la prise, ensuite relier au secteur, sinon les étincelles peuvent se produire sur la fiche au moment de l'introduction et l'adaptateur peut griller.

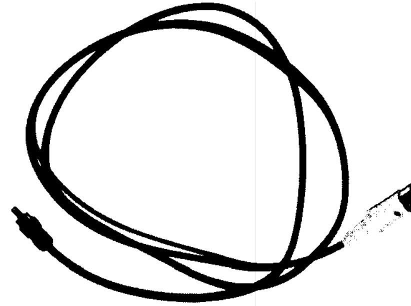
Parfois, il convient de ne pas enfoncer le jack à fond.

VOICI L'ENSEMBLE :

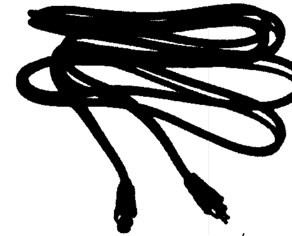


2) Vous disposez d'un moniteur monochrome.

Il vous faut un câble spécial (F) peu onéreux (option). Il est muni à une extrémité d'une fiche **DIN** à introduire dans la prise **RGB** de l'ORIC (3). A l'autre extrémité une fiche spéciale entrée vidéo convient pour votre moniteur.

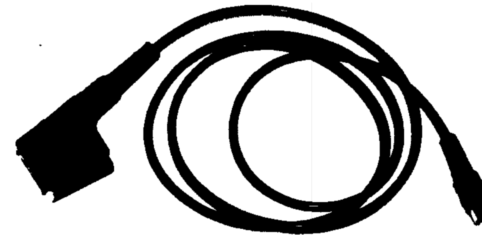


3) Vous disposez d'une télévision **PAL** ou d'une télévision Multistandard. Un câble d'antenne (G) relie l'ORIC (sortie à droite (**UHF**)) à l'entrée antenne de votre téléviseur.



La sortie de l'ORIC se fait par la prise **DIN** (2) repérée par **RGB** (initiales de RED, GREEN, BLUE c'est-à-dire Rouge, Vert, Bleu = **RVB**).

Il vous faut un câble spécial **PÉRITEL** (3) (fourni à part).



15/08/2000
Page 6 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr

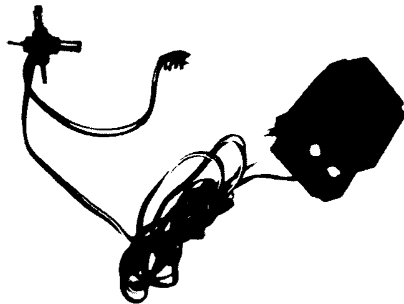
Numérisation http://wsc.n3.net

Vous entrez dans votre poste de télévision avec le câble d'antenne relié à l'adaptateur, le câble **PERITEL** se branche sur l'adapteur **UHF**.

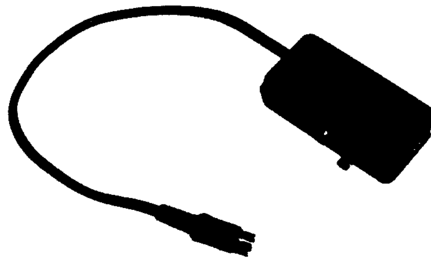
4) Vous disposez d'un moniteur couleur muni d'une prise péritel ou d'une télévision couleur récente (donc munie d'une entrée **PÉRITÉLÉVISION**).

Le câble **PÉRITÉLÉVISION** (C) employé seul se branche d'une part sur la prise (2) marquée **RGB,** d'autre part à l'entrée **PERITEL** de votre appareil.
Deux cas sont alors possibles :

Votre image se forme, cela signifie que votre prise péritel est alimentée en 12 Volts.
b) Vous avez des zébrures, une image instable, il vous faut fournir du 12 Volts continu dans la prise prévue à cet effet. Une alimentation spéciale (D) (option) vous est alors nécessaire.



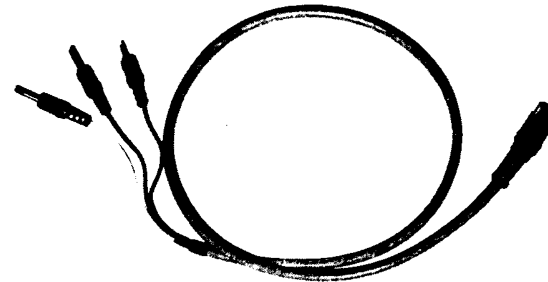
5) Vous n'avez qu'une télévision N et B système **SECAM.**
Connecter la prise **DIN** de l'adaptateur (H) (option) à la sortie (2) de l'ORIC.
Brancher le cordon **UHF** (G) entre le boitier (H) et l'entrée antenne de votre téléviseur.



L'ORIC fournit une image nette et des couleurs franches. Sur un écran monochrome on obtient des tons dégradés ou des effets différents quand on utilise les changements de couleurs.
Si votre écran est allumé, une image se forme et le message suivant apparaît :

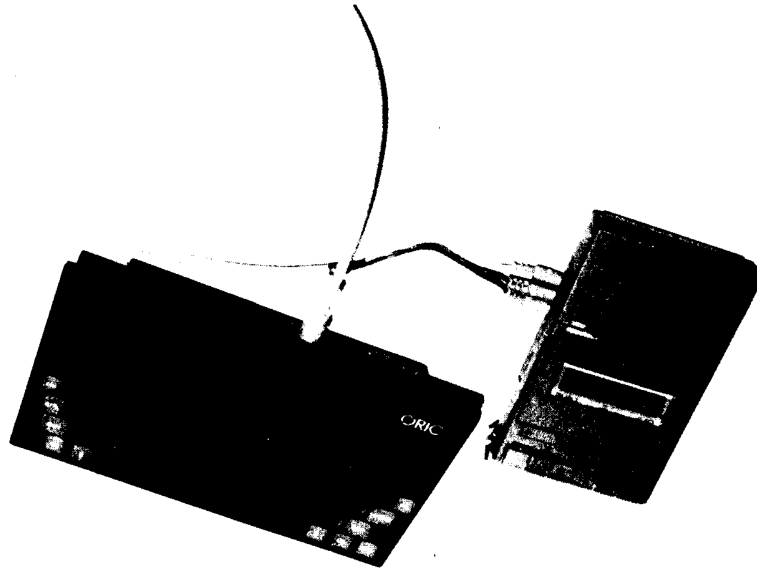
ORIC EXTENDED BASIC V1-1
C 1983 TANGERINE
37631 BYTES FREE

Ready
37631 est le nombre d'octets libres pour un 48K. Si vous avez un 16K, vous disposez de 4863 octets seulement. L'écart entre 16 et 48 kilo-octets est de 32 × 1024 × 32 768 octets. (voir **GRAB**).
Indiquons tout de suite que chaque lettre ou chiffre ou signe consomme un octet.
Le mot Ready (prêt) signifie que l'ORIC est utilisable. A la ligne suivante un carré noir s'affiche et s'éteint : c'est le curseur. Quand on actionne une touche du clavier, c'est là que s'inscrit le caractère correspondant. Nous verrons plus loin comment converser avec ORIC.
A côté de la sortie **RGB** vous lisez **TAPE :** ce qui signifie **RUBAN** (comprenez ruban magnétique). C'est la sortie magnétophone (3). C'est une prise **DIN 7** broches. Il vous faut un cordon adapté à votre magnétophone (**DIN — DIN** ou **DIN** — 3 jacks). Demander conseil à votre point d'approvisionnement.



15/08/2000
Page 7 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr

Cette sortie vous permettra de sauvegarder vos programmes (voir le chapitre 6). Elle permet aussi de charger les programmes que vous avez sur cassette.

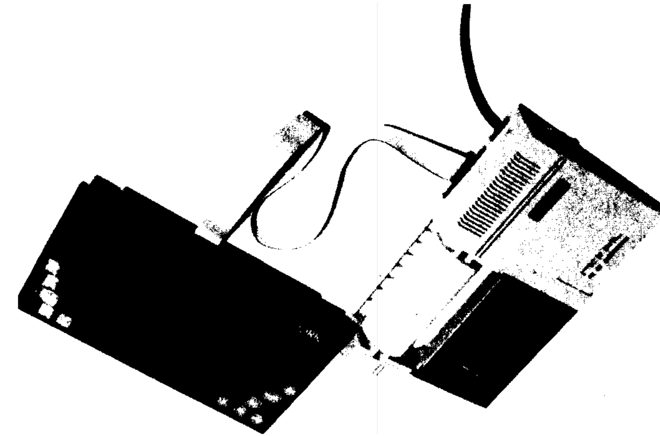


La sortie comporte deux bornes (4 et 5 voir page 168) qui permettent de relayer le haut-parleur de l'**ORIC** par un amplificateur extérieur.

A côté, vous disposez d'une sortie vers une imprimante (4). Vous pouvez vous raccorder à toute imprimante munie d'une interface parallèle Centronics, un câble spécial sera nécessaire. Veiller à ne pas le monter à l'envers, en général il est muni d'un détrompeur. L'imprimante **MCP — 40 ORIC** est non seulement une imprimante en 4 couleurs mais aussi une petite table traçante. (Voir l'Annexe 7 pour son utilisation et le chapitre 11 pour l'utilisation **ENTRÉE/SORTIE** de la broche de l'imprimante).

La grande sortie (5) est destinée aux extensions variées. Le lecteur de disque par exemple, ou un **MODEM** ou des poignées de jeux, etc. Sur le boitier vous lisez : **EXPANSION.**

Enfin, sous le boitier avec la pointe d'un crayon vous pouvez presser un bouton caché dans un trou rectangulaire : c'est le **RESET** : il permet de reprendre les commandes sans perdre le programme.

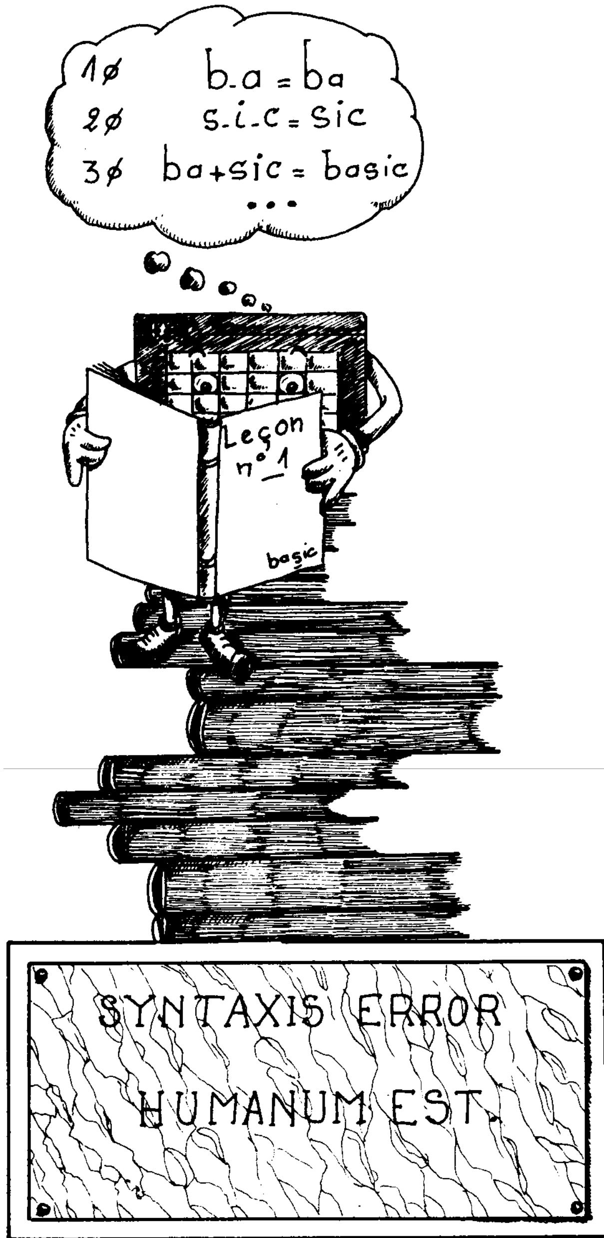


Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Numérisation http://wsc.n3.net



Chapitre 2
Initiation au langage

Voici votre ORIC installé, mais à part le message inscrit sur l'écran il n'est pas très actif ! C'est un être muet qui, bien que doté d'un "quasi cerveau", attend des ordres pour agir. Il nous faut un langage compréhensible par ORIC et par nous-mêmes, le clavier servant à lui parler, l'écran affichant nos demandes et ses réponses où le résultat des actions commandées. Nous allons dans ce chapitre aider quelque peu les débutants. Ceux qui ont déjà de l'expérience peuvent le survoler. Il faut de la persévérance, de la patience. Nous allons vous initier à une nouvelle façon de penser, vous indiquer quantité de petits détails, rien n'est inutile : apprenez chaque chose en manipulant ; c'est votre propre expérience qui compte. N'hésitez pas à tester vos idées : ORIC ne peut pas être endommagé à partir du clavier.

Ce qui se passe dans la puce est expliqué dans le chapitre consacré au langage machine qui intéressera certains d'entre vous tôt ou tard. Un langage évolué, c'est-à-dire plus proche du langage humain est disponible sur votre ORIC : c'est le BASIC.

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

Ce langage écrit en 1964 est le plus répandu et permet d'écrire aisément des programmes. Celui dont vous disposez a été mis au point par la société MICROSOFT, mondialement réputée et comporte un nombre appréciable de mots. C'est de l'anglais, mais ceux qui ne connaissent pas la langue ne seront pas longtemps gênés : une centaine de mots cela se retient assez vite, surtout ceux qui sont souvent utilisés.

Lorsqu'on converse avec ORIC, un interprète traduit les mots BASIC, pour les rendre exécutables et les effets sont instantanément visibles sur l'écran.

Observons le clavier. Nous y reconnaissons les touches d'une machine à écrire : elles sont noires. Celles qui sont rouges sont un peu particulières. Lorsque vous appuyez sur une touche noire le caractère correspondant s'inscrit à l'écran. Le carré clignotant occupe la place où s'écrira le prochain caractère : c'est le curseur.

15/08/2000 Page 9 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

La touche comportant le chiffre **5** vous permet aussi d'écrire **%** : l'une des 2 touches marquées **SHIFT** (majuscule) vous donne accès au caractère **%** par appui simultané sur **SHIFT** et **5**. Les flèches vous permettent de déplacer le curseur. La large barre noire écrit un espace. La touche **FUNCT** n'est pas utilisée en version BASIC 1.1. Toutefois en annexe les "experts" trouveront comment l'exploiter parfois. La touche **DEL** efface le dernier caractère inscrit : quel progrès comparé à la gomme ! On peut bien sûr "gommer" plusieurs caractères. Laissez le doigt appuyé sur une touche : le caractère se répète rapidement. L'effacement se fait tout aussi rapidement. (**DEL** est l'abréviation de **DELETE** = EFFACER).

Pour les touches ne comportant qu'une lettre majuscule l'action sur **SHIFT** est sans effet pour l'instant.

Il se pourrait qu'en tapant n'importe quoi sur le clavier l'image à l'écran se dérègle. N'ayez aucune crainte, vous n'avez rien détérioré. Deux méthodes s'offrent à vous pour reprendre l'utilisation correcte. La plus radicale consiste à retirer la prise d'alimentation, à attendre quelques secondes et à rebrancher. Une autre façon consiste à actionner un bouton caché au fond d'un orifice rectangulaire sous l'ORIC. Appuyer avec la pointe d'un crayon ou un objet de votre choix. Ce bouton s'appelle **RESET** : il peut (pas toujours) redonner la main sans détruire l'état des mémoires. Vous apprécierez cette commodité.

Ce qui apparaît à l'écran est le reflet de la mémoire écran. Quand on actionne les touches noires on alimente la mémoire provisoire du clavier. L'action sur la touche rouge marquée **RETURN** provoque la lecture de cette mémoire clavier, son interprétation, et l'exécution quand le texte a été compris, enfin la mémoire clavier est vidée.

Si vous avez écrit n'importe quoi, vous ne serez pas étonné de voir s'inscrire à l'écran :

? SYNTAX ERROR

Ce qui signifie que l'interpréteur n'a pas compris : cela peut venir d'une simple faute d'orthographe, ou d'un petit détail non conforme comme une virgule là où il faut un point-virgule. Plus l'erreur est minime plus il vous sera difficile de la déceler.

Écrivez à nouveau l'ordre en faisant bien attention.

La mémoire tampon du clavier accepte 75 caractères puis une sonnette vous avertit que vous n'avez plus beaucoup de place. Vous avez ainsi 3 rappels jusqu'au 78e et dernier caractère possible. Si vous allez jusqu'à 79 vous voyez s'afficher \ (signe appelé antislash) et le curseur se retrouve en début de ligne suivante. La mémoire du clavier a débordé, elle s'est vidée... son contenu est perdu. Il est toutefois en mémoire écran et nous verrons plus loin comment récupérer du texte sans le réécrire, en recopiant la mémoire écran dans la mémoire du clavier.

Cependant le message **Ready** est à l'écran et le curseur vous invite à exprimer vos désirs que l'ORIC va s'empresser de satisfaire dès l'appui sur **RETURN,** en abrégé **RTN**. Essayez :

PRINT "MESSAGE"

puis appuyez sur **RTN.** Si vous n'avez pas confondu les guillemets avec une double apostrophe vous obtenez le mot MESSAGE écrit à la ligne suivante et à nouveau le mot Ready et le curseur. ORIC est prêt pour autre chose. Notez qu'ouvrir et fermer les guillemets c'est le même signe. Notez aussi qu'il n'est pas nécessaire de les fermer dans la mesure où rien d'autre n'est écrit à côté.

On continue avec

PRINT "4 + 5" (appuyez sur **RTN**)

Vous obtenez 4 + 5 à la ligne suivante. Maintenant tapez :

PRINT 4 + 5 (puis **RTN**)

Cette fois ORIC répond 9. Il a compris qu'il fallait écrire le résultat de l'addition de 4 et de 5, alors qu'avant il n'interprétait pas le message entre guillemets. Vous avez compris que **PRINT,** qui signifie IMPRIME, provoque l'affichage à l'écran.

Combinons les deux :

PRINT "4 + 5 ="; 4 + 5

Le point virgule demande l'impression côte à côte. Ici on peut l'omettre, le résultat est le même. Changeons d'opération :
* au-dessus du 8 est le signe de multiplication.

PRINT 4*3

Nous pouvons employer la lettre X pour indiquer une multiplication dans la partie non interprétée :

PRINT "4 × 3 = "4*3

L'écriture est conforme à nos habitudes.

Vous avez vu qu'un espace sépare le signe = du 1 de 12. C'est qu'ORIC inscrit les nombres positifs à l'écran avec un espace au début et à la fin. Disons tout de suite qu'il inscrit les nombres négatifs avec un signe moins devant et un espace derrière. Ainsi, lorsque des nombres sont envoyés successivement à l'écran, sur la même ligne, ils sont séparés et lisibles. C'est intéressant pour la disposition de tableaux de nombres.

Voici la liste complète des signes opératoires disponibles :

+ , *, —, /, ↑

dans l'ordre addition, multiplication, soustraction, division, élévation à une puissance : ce dernier signe est avec le **6** sur le clavier ; à l'écran il correspond à ↑. On comprend qu'il indique que le nombre

Numérisation http://wsc.n3.net
15/08/2000 Page 10 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

suivant devrait être écrit plus haut. La division donne soit le résultat exact quand c'est possible, soit un résultat approché avec 9 chiffres en tout. Pour l'exponentiation (↑) ne soyez pas surpris des résultats approchés fournis, même dans certains cas simples comme **7↑2**, cela tient à la façon de procéder. Avec **7*7** le résultat est meilleur. C'est ainsi sur la plupart des ordinateurs de cette catégorie.

La meilleure façon pour vous d'apprendre comment ORIC se livre aux calculs est d'expérimenter. Pour plus de commodité apprenez que ? est une abréviation de **PRINT**. Ainsi
? "AH" donne le même résultat que **PRINT "AH"**.

Essayez : (n'oubliez pas **RTN** à chaque fois)
?6/2 **?2↑2** (2 au carré) **?11 – 19**

Vous voyez qu'ORIC connaît les nombres négatifs.

Essayez maintenant

?13/2 **?1/2**

Observez que la virgule est remplacée par un point et n'oubliez pas d'écrire un point (et non une virgule) pour séparer les chiffres décimaux. Observez aussi que "le zéro-avant-la-virgule" n'est pas écrit : c'est la notation anglaise.

Toutefois si vous écrivez **?Ø.7/2**, ORIC comprend.

Attention ne confondez pas le zéro **Ø** qui est barré et la lettre O. De même, attention à ne pas confondre I et 1. Un être humain est capable de rectifier de lui-même ce genre d'imprécision, pas un ordinateur.

? **SYNTAX ERROR** vous rappelera l'incompréhension de l'interpréteur.

Si vous essayez des expressions plus complexes comme **2/3 + 4** ou **4*3 + 5/3** vous pouvez parfois vous étonner des résultats. Il faut savoir les règles de priorité.

Une expression entre parenthèses est toujours calculée d'abord.

Ensuite la (ou les) élévation(s) à une puissance (↑).

Puis à égalité * et /, enfin + et –.

Ainsi **?2 + 3*4** vous donne **14** puisque la multiplication 3*4 est effectuée d'abord, et 2 est ensuite ajouté à 12.

Avec **?4*3 + 2** on obtient aussi **14**, bien sûr.

Si l'on veut calculer 3 + 2 et le résultat de cette addition multiplié par 4 il faut écrire :

? (3 + 2)*4 ou ? 4*(3 + 2)

Les opérations de même priorité sont exécutées dans l'ordre de l'écriture, de gauche à droite.

Ainsi **?4 – 2 + 4 + 5** donne **11** mais **?4 – (2 + 4 + 5)** donne **– 7**, puisque l'expression entre parenthèses donne 11 comme vous le savez sûrement sans le demander à ORIC.

?4*6/7 donne **3.42857143** avec 9 chiffres en tout et le dernier arrondi automatiquement. Ici aussi il y a un espace au début et à la fin.

Insistons sur la différence d'affichage entre :

?"1"; "2" et ?1; 2

Le point-virgule, après l'instruction **PRINT** provoque l'écriture côte à côte. Dans le 1er cas on obtient 12, le 2 est tout contre le 1, dans le 2e cas on obtient :

(espace) 1 (espace) (espace) 2 (espace)

Notion de variable.

Avant d'aborder la notion de programme, il nous faut examiner quelques autres idées.

En écrivant **PRINT "ABC"** nous obtenons l'écriture des 3 lettres ABC. Ceci est dû, nous l'avons dit, à la présence des guillemets. Si l'on omet les guillemets, **PRINT ABC** est compris et l'on obtient un **Ø**. Comment se fait-il ? ABC a été interprété comme nom de variable numérique. La valeur **Ø** est choisie par défaut. Si nous indiquons **LET ABC = 93** ou tout simplement **ABC = 93** sans oublier **RTN** et que nous demandons **?ABC** nous obtenons **93**. Dans une zone de sa mémoire ORIC a choisi un emplacement, y a mis le nombre **93** et a retenu que **ABC** était le nom de cette zone mémoire. On peut en changer le contenu en écrivant

ABC = 15 par exemple,

c'est pourquoi ABC s'appelle une variable.

Lorsqu'on écrit **PRINT ABC**, un espace permet une lecture plus aisée mais **PRINTABC** est compris avec autant de facilité. Ceci est valable pour de nombreuses instructions.

Essayons maintenant

ABC = 1ØØ
ABC = 13 + 1Ø suivi de RTN
ABC = ABC + 1Ø et de ?ABC
ABC = ABC*2 à chaque fois
ABC = ABC/2.3

Comme vous l'avez vu, on peut non seulement affecter une valeur à une variable mais aussi utiliser la variable dans les calculs. Ces écritures ont de quoi surprendre car il est mathématiquement faux que ABC soit égal à ABC augmenté de 1Ø. Mais en langage BASIC le signe = est ici utilisé pour signifier : "mettre dans la case ABC le contenu de la case ABC augmenté de 1Ø". Avec le mot **LET** c'est plus clair : **LET ABC = ABC + 1Ø**. LET pourrait se traduire par FAIS EN SORTE QUE.

William Saint-Cricq / wsaintcr@wanadoo.fr Page 11 sur 163 15/08/2000

Numérisation http://wsc.n3.net

A l'inverse, **LET 100 = ABC** est refusé car 100 n'est pas un nom de variable acceptable. Seul les noms commençant par une lettre majuscule sont valables. Le message **?SYNTAX ERROR** est envoyé à l'écran.

Après la première lettre on peut employer au choix lettres et chiffres sans autre limitation de longueur que celle due à la capacité de la mémoire tampon du clavier. Les espaces seront négligés : PRIX DU LITRE est assimilé à PRIXDULITRE. Le choix de variables comme PAYE, SOLDE, TVA favorise la clarté des programmes. Il faut savoir cependant que seuls les deux premiers caractères servent à différencier les variables. Essayez de demander **PRINT ABUS** et vous verrez que le contenu est celui de la dernière valeur attribuée à la variable ABC. Il est essentiel de noter que les mots, réservés (annexe 12) ne peuvent pas être utilisés comme noms de variables, ni même s'y trouver incorporés. Exemple GOLF contient GO. LONG contient ON. TOTAL contient TO. PORT contient OR. Ces 4 noms seront rejetés et signalés par **?SYNTAX ERROR**.

On commet facilement des erreurs de ce genre quand on est débutant.

Un mot réservé est un mot qu'ORIC reconnaît : il fait partie de son BASIC. Nous avons déjà vu **PRINT** et **LET** et nous allons voir les autres peu à peu. Le chapitre 9 vous donne la liste complète en ordre alphabétique des mots compréhensibles par l'ORIC ATMOS. Cette liste est limitative : aucun autre mot n'est compris. ORIC interprète chaque caractère et compare à son vocabulaire et transfère les mots reconnus sous une forme codée (numériquement) dans une zone de sa mémoire, ceci dès qu'on appuie sur **RTN**.

Tout ce qui n'est pas entre guillemets est pris soit pour une variable, soit pour une lettre d'un mot BASIC. Rappelons que **LET** est facultatif. **ABC = 100** est synonyme de **LET ABC = 100**. Vérifiez-le sur quelques exemples.

L'éditeur

Avant d'aborder la programmation, il nous faut étudier l'éditeur. Il s'agit d'utiliser le clavier en observant l'écran pour nourrir l'ordinateur d'informations qu'il aura à traiter.

Jusqu'ici nous avons surveillé attentivement ce que nous avons tapé. En cas d'erreur, nous avons utilisé la touche **DEL** pour effacer le dernier caractère entré, puis tapé la suite correctement. Nous avons pu aussi appuyer sur **RTN** et obtenir le fameux message ?SYNTAX ERROR ce qui nous a conduit à écrire à nouveau la ligne. Ceci nous a montré l'importance de la précision de la frappe, du respect des mots, des signes. Les négligences, les à-peu-près ne sont pas possibles avec un ordinateur. Heureusement ORIC vous offre des moyens de corriger aisément vos fautes, et mêmes de modifier des portions de phrase lorsque vous vous êtes trompés.

Les 4 touches fléchées et la touche **CTRL**

Vous avez remarqué que l'affichage à l'écran se fait de haut en bas, et que, lorsque l'écran est plein, la nouvelle ligne s'écrit en bas en repoussant toutes les autres d'un cran vers le haut, ce qui fait que la ligne supérieure est perdue.

Ce décalage systématique, ce défilement du texte, (ce déroulement) rappelle l'écriture sur parchemins. (En anglais c'est le scrolling). Notons au passage que sur ORIC ATMOS il est plus rapide que sur ORIC 1.

Pour ce débarrasser du contenu de l'écran on écrit **CLS** suivi de **RTN**. (Clear Screen = Nettoyer l'écran). Le curseur va en haut à gauche et le mot Ready s'inscrit. Un appui simultané sur **CTRL** et sur **L** nettoie l'écran, met le curseur en haut à gauche mais n'écrit pas le mot Ready et, plus important, ne vide pas la mémoire tampon du clavier. Supposons que vous ayiez commencé à écrire **ABC = 18** et que vous faites **CTRL L.** Vous écrivez alors **ABC = 28** puis actionnez **RTN** . Vous obtenez **?SYNTAX ERROR**.

Le contenu de la mémoire tampon était :

ABC = 18ABC = 28

ce qui n'a pas de sens pour ORIC.

Un appui *simultané* sur **CRTL** et sur un certain nombre de touches produit des effets fort utiles, en nous épargnant de nombreuses frappes.

Voici quelques autres usages de la touche **CTRL** :

CRTL A	recopie le caractère présent sous le curseur : ce caractère va dans la mémoire tampon du clavier.
CTRL F	Commutateur du BIP des touches.
CTRL Q	Commutateur du curseur.
CTRL X	Efface la mémoire tampon du clavier sans en interpréter le contenu.
CTRL T	Commutateur majuscules/minuscules.

Le commutateur **CTRL F** s'utilise, soit pour supprimer le bruit des touches, soit pour le rétablir. **CTRL Q** permet d'effacer ou de restituer le curseur clignotant. D'emblée, à la mise en route l'ORIC se met en mode majuscules, ce qui est signalé par l'inscription CAPS en haut à droite de l'écran. (CAPS est une abréviation de CAPITALES). **CTRL T** donne accès aux minuscules. Les touches alphabétiques sont seules concernées. Un appui simultané sur une touche **SHIFT** et sur une touche alphabétique permet l'écriture

15/08/2000 Page 12 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

d'une majuscule. Pour les signes, qu'on soit en mode majuscule ou non, on obtient usuellement celui qui est écrit en bas et par appui sur **SHIFT** celui qui est en haut. Il est important de s'apercevoir qu'en mode majuscules l'appuis sur **SHIFT** pour les lettres n'a pas d'effet. Cette particularité du clavier est due au fait que :

Les instructions et commandes en basic
doivent être écrites en majuscules sur oric

Nous allons maintenant écrire un petit programme en BASIC et l'utiliser. Cela nous servira à comprendre le fonctionnement de l'éditeur de l'ORIC.

Chaque fois que vous tapez sur une touche, assurez-vous que vous le faites sans erreur. En particulier surveillez la ponctuation, car virgule ou point-virgule, apostrophe ou guillemet, faciles à confondre, produisent des effets nettement différents.

Notre premier programme est un calcul de TVA, au taux de 18,6 %. Choisissons un article dont le prix hors taxe est 540 F. Il faut introduire cette donnée dans l'ordinateur. Nous ferons ensuite calculer le montant de la TVA (en multipliant par 0,186). Nous déterminerons le prix TTC par addition. Enfin nous demanderons l'affichage du total. Ce qui donne ceci :

```
1Ø REM......CALCUL DE TVA.......
2Ø HT=54Ø
3Ø TVA HT=HT*Ø.186
4Ø TTC=HT +TVA
5Ø PRINT "TOTAL=";TTC
```

Un programme est formé de lignes numérotées. Ici de 1Ø à 5Ø avec un intervalle de 1Ø. On peut choisir les numéros de ligne de Ø à 63999. L'intervalle est utile pour insérer des lignes en cas d'oubli ou pour perfectionner un programme sans avoir besoin de tout renuméroter. Pour introduire ce texte il est bon de commencer par effacer l'écran. Utiliser **CTRL L** ou CLS (un appui sur chacune des 3 touches C, L, S) suivi de **RTN**. Taper le programme indiqué ligne par ligne en appuyant sur **RTN** à la fin de chaque ligne. Pendant la frappe l'enregistrement est fait de façon provisoire dans la mémoire tampon du clavier. Simultanément la mémoire écran est utilisée ce qui vous permet de suivre votre travail. En appuyant sur **RTN**, il n'y a pas exécution immédiate car vos instructions sont précédées d'un n° de ligne. ORIC va installer cette ligne dans une zone spéciale appelée mémoire programme. En outre la mémoire tampon sera effacée.

Si vous commettez une erreur minime vous savez déjà que la touche **DEL** vous permet la correction. Si vous voulez annuler tout ce

que vous êtes en train d'écrire appuyez simultanément sur **CTRL** et **X** vous verrez apparaître le signe \ et le curseur ira en début de ligne suivante. **CTRL X** vide la mémoire tampon du clavier sans transférer la ligne en mémoire programme. La ligne 1Ø comporte le mot BASIC REM. C'est le début du mot REMarque. Ce qui suit n'est pas lu par l'ordinateur pendant l'exécution. C'est une indication pour le lecteur. Cette ligne sera mise avec les autres en mémoire programme. La ligne 1Ø comporte aussi des minuscules. Après avoir tapé le début jusqu'au C de Calcul faire **CTRL T** ce qui vous donne accès au clavier minuscule. Revenir par **CTRL T** au clavier majuscules ce qui se vérifie par l'apparition de CAPS en haut à droite.

Si vous avez hésité plusieurs fois et utilisé **CTRL X,** l'écran peut avoir un aspect comme celui-ci :

```
1Ø REM......CALCUL DE TVA.......
2Ø HT=57\
2Ø HT=54Ø
3Ø TVA HT=HT*Ø.176\
3Ø TVA HT=HT*Ø.186
4Ø TTC=HT +TVA
5Ø PRINT "TOTAL=';  \
5Ø PRINT "TOTAL=";TTC
```

Pour connaître le contenu actuel de la mémoire programme écrire **LIST**, en mode direct, puis **RTN.**

Le programme apparaît à l'écran, sans lignes surperflues.

La commande **LIST** recopie la mémoire programme dans la mémoire écran.

Écrivons délibérément une nouvelle ligne 3Ø erronée.

```
3Ø TV=HT*Ø.186
```

Validons par **RTN.** Effaçons l'écran par **CTRL L.** Demandons le listage avec **LIST RTN.** Nous voyons que cette nouvelle ligne a supplanté l'ancienne. Deux méthodes de correction sont possibles : écrire à nouveau toute la ligne 3Ø correctement ou utiliser une commande particulière : **EDIT.**

Écrivons

```
EDIT 3Ø  [RTN]
```

La ligne 3Ø s'affiche et le curseur clignote juste devant le 3 de 3Ø.

Numérisation http://wsc.n3.net
William Saint-Cricq / wsaintcr@wanadoo.fr Page 13 sur 163 15/08/2000

Numérisation http://wsc.n3.net

Si ce n'est pas le cas faire **CTRL Q** pour le faire apparaître.
Nous allons garnir la mémoire tampon du clavier à partir de la mémoire écran par **CTRL A.**
Maintenons appuyé *simultanément* **CTRL** et **A.** Le curseur se déplace vers la droite. Tout caractère dépassé est recopié. Lorsque le curseur est parvenu sur =, il nous faut insérer le A qui manque. Pour cela utilisons la flèche ↑ qui déplace le curseur sans autre effet. Écrivons alors A. Voici l'aspect de la ligne :

```
    A
30 TV=HT*0.186
```

La commande **EDIT** prévoit une ligne libre au-dessus pour permettre l'insertion de la sorte. Attention ! Le reste de la ligne n'a pas été recopié. Avec les flèches ↓ et ← revenir sur le signe =. Recopier jusqu'au bout de la ligne par **CTRL A** et faire **RTN** pour transférer la nouvelle ligne 30 de la mémoire tampon du clavier à la mémoire programme. Vérifier avec **LIST** et **RTN.**
Si le curseur n'est pas devant le numéro de la ligne à modifier, il est facile de le déplacer avec les flèches. On peut même se passer de la commande **EDIT**. Il suffit que l'écran affiche ce que l'on a à recopier pour que l'opération soit possible. Voici la méthode. S'assurer que la mémoire tampon du clavier est vide par **RTN** ou **CTRL X.** Écrire le n° de ligne désiré ou le relire par **CTRL A.** Ensuite recopier n'importe qu'elle partie de l'écran par **CTRL A.**
Se déplacer d'un endroit à l'autre, sans contrainte, par l'usage des flèches de déplacement du curseur.
Écrire des caractères quand on le désire, l'endroit où on les voit à l'écran n'a pas d'importance. Tout ce qui compte c'est la succession de leur introduction en mémoire tampon du clavier.
Ne pas dépasser toutefois 78 caractères, la sonnette signale qu'il faut s'arrêter. **RTN** transfère le tout à la mémoire écran. **LIST RTN** permet de vérifier. **LIST 30** vous donne seulement la ligne sur laquelle vous travaillez ce qui est souvent utile.
Le principal intérêt de cette méthode de relecture est la possibilité de création d'une nouvelle ligne. Souvent dans les programmes plusieurs lignes se ressemblent à un détail près.
Ainsi la ligne

```
240 PRINT "TOTAL ="; PRIX
```

peut être obtenue facilement à partir de

```
50 PRINT "TOTAL ="; TTC
```


Le curseur est d'abord amené devant le 5. On écrit 240 ; on recopie le début par **CTRL A.** Arrivés sur le T de TTC on écrit PRIX à la place de TTC. **RTN** et c'est fait. La ligne 240 existe, la ligne 50 n'est ni perdue, ni modifiée.

Suppression de caractères.

Supposons que la ligne 50 soit :

```
50 PRINT "TOTAL GENERAL="; TTC
```

et que nous voulions supprimer le mot "GÉNÉRAL".

On recopie le début de la ligne. Après avoir dépassé le L de TOTAL, on déplace le curseur avec → jusqu'au signe =. On reprend alors la recopie jusqu'au bout. On valide par **RTN.**
Pour se débarasser d'une ligne superflue, il suffit d'écrire son numéro suivi de **RTN.** ORIC l'efface alors de la mémoire programme.
Si l'on a oublié la ligne n° 20, il suffit de l'écrire : elle ira s'insérer entre 10 et 30. On peut créer entre 10 et 20 autant de lignes que de nombres entiers disponibles dans cet intervalle : 11, 12, 13... 19.
Pour obtenir l'exécution du programme préparé on écrit **RUN** suivi de **RTN.** L'ordinateur lit les ordres écrits dans sa mémoire programme. Ligne 10 une remarque, passage à la ligne 20. La variable HT prend la valeur 540. Ligne 30 la variable TVA prend la valeur 540 × 0,186. Ligne 40 la variable TTC est calculée par addition du contenu des variables HT et TVA. Ligne 50 sur l'écran s'affiche le résultat :

```
TOTAL = 640.44
```

Si vous n'avez pas obtenu ce résultat, c'est qu'il subsiste quelque erreur. Reprenez le travail de mise au point et essayez à nouveau.

Recherche d'erreurs

Modifions les lignes 40 et 50 avec l'idée de les rendre plus claires :

```
40 TOTAL = HT + TVA
50 PRINT "TOTAL =";TOTAL
```

A l'exécution de ce programme, demandée par la commande **RUN** suivie de **RTN** nous obtenons **?SYNTAX ERROR IN 40.** Pourquoi ?

William Saint-Cricq / wsaintcr@wanadoo.fr Page 14 sur 163 15/08/2000

Numérisation http://wsc.n3.net

Le mot TOTAL commence par TO qui est un mot réservé du BASIC. Aussi l'instruction est lue **40** TOTAL = HT + TVA et n'a pas de sens en BASIC. C'est le genre d'erreur capable de gêner un débutant, aussi surveillez bien les noms de vos variables en les confrontant avec la liste des mots réservés donnés en appendice.

Les variables HT, TVA et TTC choisies précédemment sont correctes TVA et TTC sont distinguées puisque 2 lettres sont retenues. Si nous remplaçons TTC par PRIX, le BASIC l'acceptera.

Reprenons le programme indiqué au début. Tout va bien, mais son utilité est très limitée. Il serait intéressant d'obtenir le prix TTC pour des prix HT variés. C'est très facile, il suffit de modifier la ligne **20**. Au lieu de fixer la valeur de la variable HT de façon immuable à **540**, nous allons demander à l'ordinateur de lire cette valeur au clavier.

L'instruction **INPUT** (de l'anglais TO PUT IN, mot à mot : mettre dans) va nous servir. Écrivons :

```
20 INPUT HT
```

Lorsqu'on exécute ce nouveau programme, un point d'interrogation apparaît à l'écran. Cela signifie que l'ordinateur attend le prix HT. Entrez un nombre puis **RTN** et vous obtenez le prix TTC correspondant. Le nombre que vous avez écrit a été affecté à la variable HT. Un point d'interrogation n'est pas une indication bien claire, aussi convient-il de renseigner la personne qui est au clavier sur ce qu'on attend d'elle. Écrivons :

```
20 INPUT "INDIQUER LE PRIX HT";HT
```

A l'exécution, le message entre guillemets, va s'afficher suivi par le point d'interrogation, facilitant l'utilisation du programme.

Faites plusieurs essais. Entrer autre chose qu'un nombre. Vous recevez alors le message

```
?REDO FROM START
```

car la variable HT doit recevoir une valeur numérique et tout ce qui est différent est rejeté. Ce message indique que ce qui vient d'avoir lieu est négligé et que l'on attend une nouvelle introduction.

A chaque essai vous avez dû écrire **RUN** et appuyer sur **RTN.** Il est possible d'enchaîner de nombreuses exécutions par programme. Le BASIC de l'ORIC nous offre plusieurs possibilités. La plus simple (mais pas la meilleure) est l'usage de **GOTO**. Cette instruction est suivie d'un n° de ligne. A la rencontre de la ligne contenant

GOTO le déroulement du programme cesse de se faire en suivant les numéros de lignes successifs. Le pointeur de ligne se branche inconditionnellement à la ligne indiquée. Si nous voulons reprendre à partir de la ligne **20** nous écrirons **GOTO 20**. Une remarque est utile pour aider la compréhension. Nous utilisons l'apostrophe qui, ici, est une abréviation de **REM**, de même que le point d'interrogation peut remplacer **PRINT**. Pour un si petit programme, mettre ainsi une remarque est sans doute excessif. Mais sur cet exemple nous exposons les méthodes à mettre en œuvre pour des programmes qui peuvent atteindre plusieurs centaines de lignes. Les remarques sont alors fort utiles : elles font ressortir la structure et précisent l'objet des diverses parties. Ceci à l'usage de l'auteur lui-même. Lorsqu'on revient travailler un sujet abandonné pendant des semaines, l'absence de renseignements peut rendre la compréhension très difficile, impossible même parfois. Pour d'autres personnes qui ont accès au programme, les remarques sont encore plus utiles.

```
10 REM.....CALCUL DE TVA.....
20 INPUT"INDIQUEZ LE PRIX HT ";HT
30 TVA = HT*0.186
40 TTC=HT+TVA
50 PRINT "TOTAL =";TTC
60 'renvoi pour un nouveau calcul
70 GOTO 20
```

Chaque fois que le programme arrive à la ligne **70**, le cours se poursuit à partir de la ligne **20**.

Pour interrompre un tel programme il faut utiliser **CTRL C.** Ceci rend la main au clavier. Le message **BREAK IN 20** par exemple est affiché. Cette façon de faire est nécessaire lorsque le programme boucle indéfiniment. Ici c'était volontaire. En d'autres occasions le programme pourra entrer en "boucle folle" sans que vous l'ayiez prévu.

Indépendamment des calculs effectués, le programme boucle comme celui-ci :

```
10 PRINT"BOUCLE"
20 GOTO 10
```

Il existe d'autres méthodes pour programmer des procédures répétitives. Elles seront étudiées en détail au chapitre 4.

En utilisant le programme de calcul de TVA dans la dernière forme que nous lui avons donnée, on commence à comprendre l'intérêt des micro-ordinateurs. Inlassablement l'ORIC répond dès que le prix HT lui a été indiqué. A moins que vous ne soyez com-

William Saint-Cricq / wsaintcr@wanadoo.fr   Page 15 sur 163   15/08/2000

merçant ou dans les affaires, ce programme ne vous facilitera pas la vie, mais là n'est pas notre sujet.

Pour l'instant nous nous familiarisons avec le langage BASIC et ce programme en vaut un autre pour l'initiation. Le jeu, les couleurs clignotantes, la musique électronique, ce sera pour plus tard : il nous faut d'abord apprendre l'essentiel du BASIC... dommage !

Que pourrions-nous faire d'autre maintenant avec ce programme tel qu'il est ? Vous avez vu que les lignes nouvelles s'inscrivent en bas de l'écran, produisant un décalage systématique des autres vers le haut. Ce serait plus agréable de n'avoir à l'écran qu'un calcul à la fois. Nous avons vu **CLS**. Nous l'avons utilisé en mode direct. On peut s'en servir aussi en mode programme. Écrivons à la place de la ligne 7Ø :

```
7Ø CLS:GOTO 2Ø
```

Voilà une ligne qui comporte 2 instructions. Pour les séparer il faut mettre la ponctuation (deux-points).

C'est comme en français : on met des points à la fin des phrases. Derrière un n° de ligne on peut mettre plusieurs instructions, à condition de les séparer par "deux-points" et ceci dans la limite de la longueur de la ligne.

Si vous faites **RUN**, vous voyez l'effet produit : l'écran est effacé chaque fois que la ligne 7Ø est rencontrée. Mais vous n'avez pas le temps de lire la réponse. Le BASIC de l'ORIC comporte une instruction qui arrête momentanément le déroulement d'un programme : c'est **WAIT** suivi d'un nombre qui indique les centièmes de seconde de la temporisation. Ainsi **WAIT 1ØØ** provoque un arrêt de 1 seconde. Écrivons ligne 7Ø :

```
7Ø WAIT 5ØØ :CLS:GOTO 2Ø
```

Nous avons maintenant un programme qui attend suffisamment pour qu'on ait le temps de voir un résultat avant de passer au calcul suivant.

Si l'on désire un temps variable, l'instruction **WAIT** accepte d'être suivie par une variable numérique.

Variables chaînes de caractères

Jusqu'ici nous avons utilisé des chaînes de caractères entre guillemets. Il est possible d'utiliser des variables pour les chaînes comme c'est le cas pour les variables numériques. Les mêmes restrictions s'imposent : pas de mots réservés au BASIC dans les noms des variables chaînes. Le premier caractère doit être une lettre majus-

cule. Les autres peuvent être au choix des chiffres ou des lettres majuscules. Le dernier caractère du nom de la variable chaîne doit être **$**. Cette désinence est impérative.

Exemple :

W$, NOM$, A12$ sont des noms de variables acceptables.

Ces variables s'emploient avec **PRINT, INPUT** et **LET**.

PRINT W$

INPUT W$

LET A$ = "ABCDEF" ou **A$ = "ABCDEF"**

LET est ici aussi optionnel. Le signe = ici est le signe d'affectation comme pour les variables numériques. Seuls les 2 premiers caractères du nom des variables sont retenues pour les différencier. Le programme ci-dessous peut être écrit sans qu'il soit nécessaire d'effacer le précédent.

```
11Ø A$="ORIC:"
12Ø LET A1$="LECON DE "
13Ø LET BA$="BASIC "
14Ø INPUT"DONNEZ VOTRE PRENOM,SVP";PR$
15Ø CLS:PRINTA1$" "BA$" POUR "PR$
16Ø PRINT:PRINT" SUR "A$
```

Nous pouvons obtenir l'exécution de ce programme en utilisant d'une autre façon la commande **RUN**. Écrivons **RUN11Ø**. **RTN** lance l'exécution à partir de la ligne 11Ø. Le mot ORIC est affecté à la variable **A$**. En 12Ø les mots LEÇON DE sont affectés (avec usage de **LET)** à la variable **A1$**. En 13Ø de même pour le mot BASIC dans la variable **BA$**. La ligne 14Ø montre un emploi de **INPUT** avec message facilitant la compréhension. Les lignes 15Ø et 16Ø provoquent l'affichage d'une phrase sur un écran préalablement vidé de son contenu. Bien observer le rôle des espaces à l'intérieur des guillemets. On aurait pu écrire la ligne 15Ø en utilisant des points virgules :

```
15Ø CLS:PRINTA1$;" ";BA$;" POUR ";PR$
```

Sur certains ordinateurs ces points virgules sont obligatoires.

Une autre façon d'obtenir l'exécution est de demander en mode direct **GOTO 11Ø** ou **GOTO11Ø**, les espaces entre **GOTO** et **11Ø** n'ont pas d'importance sur ORIC. Il n'y a pas de différence apparente. Il y en a une pourtant. Un **GOTO 16Ø** donne l'inscription ; "SUR ORIC" alors qu'un **RUN 16Ø** donne seulement : "**SUR**". C'est que la commande **RUN** efface les variables de la mémoire. Pour effacer ces variables en cours d'exécution de programme on dispose de l'instruction **CLEAR**. Les variables numériques sont mises à zéro, les variables chaînes à vide. Bien faire la différence

Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr Page 16 sur 163 15/08/2000

Numérisation http://wsc.n3.net

entre une chaîne d'espace, même réduite à un espace et la chaîne vide qui ne contient rien. Un **GOTO** pour reprendre l'exécution n'efface pas les mémoires. En faisant **GOTO 160 : CLEAR** on obtient le même effet qu'avec **RUN 160**.

Revenons à notre programme de TVA. Effaçons les lignes 110 à 160 tout simplement en écrivant les numéros des lignes suivi d'un appui sur les touches **RTN.** Les lignes sont effacées dans la mémoire programme mais pas de la mémoire écran. Avec la commande **LIST, RTN** vous pouvez vérifier. Puis vous entrez ces nouvelles lignes :

```
65 ' Lecture du clavier d'abord
66 GET A$
```

Nous savons désormais que **A$** est une variable chaîne. **GET** est une instruction qui demande à l'ORIC d'attendre qu'une touche soit enfoncée, puis de mettre l'unique caractère concerné dans la variable. Si l'on appuie sur la touche **X** alors **A$ = "X"**, par exemple. L'effet de la ligne 66 rend inutile l'attente de 5 s programmée en ligne 70. Écrivons **EDIT 70, RTN.**

Avec **CTRL A** recopions 70. Effaçons **WAIT 500** : soit en appuyant sur la barre d'espace, soit en déplaçant le curseur par la flèche →. Recopions par **CTRL A CLS : GOTO 20.** Un appui sur **RTN** pour terminer. **LIST, RTN** nous donne :

```
10 REM.....CALCUL DE TVA.......
20 INPUT"INDIQUER LE PRIX HT ";HT
30 TVA HT=HT*0.186
40 TTC= HT + TVA
50 PRINT "TOTAL =";TTC
60 'renvoi pour un nouveau calcul
65 ' Lecture du clavier d'abord
66 GET A$
70 CLS:GOTO 20
```

Vérifier que **GET** a bien l'effet indiqué en exécutant ce programme. Noter que le contenu de la variable **A$** n'est pas exploité ici. Seule la lecture du programme nous permet de savoir qu'il faut appuyer sur une touche à ce moment. Un utilisateur non prévenu aurait du mal à comprendre ce qu'il convient de faire. Aussi est-il d'usage de rendre les programmes conviviaux comme la version ci-dessous.

```
10 REM.....CALCUL DE TVA.......
15 CLS
20 INPUT"INDIQUER LE PRIX HT ";HT
```



```
30 TVA HT=HT*0.186
40 TTC= HT + TVA
50 PRINT "TOTAL =";TTC
60 'renvoi pour un nouveau calcul
65 ' Lecture du clavier d'abord
66 GET A$
70 CLS:GOTO 15
```

La ligne 15 a été ajoutée pour que l'écran soit vidé dès la première utilisation. En ligne 20, **PING** est l'une des commandes sonores prédéfinies. Cela attire l'attention de l'utilisateur qui va avoir à renseigner l'ordinateur. D'autres sons sont disponibles : **ZAP, EXPLODE** et **SHOOT**. Voir chapitres 8 et 9 pour en savoir plus.

Les lignes 25 et 35 ont été ajoutées pour renseigner l'utilisateur sur les opérations en cours. L'unité monétaire a été inscrite.

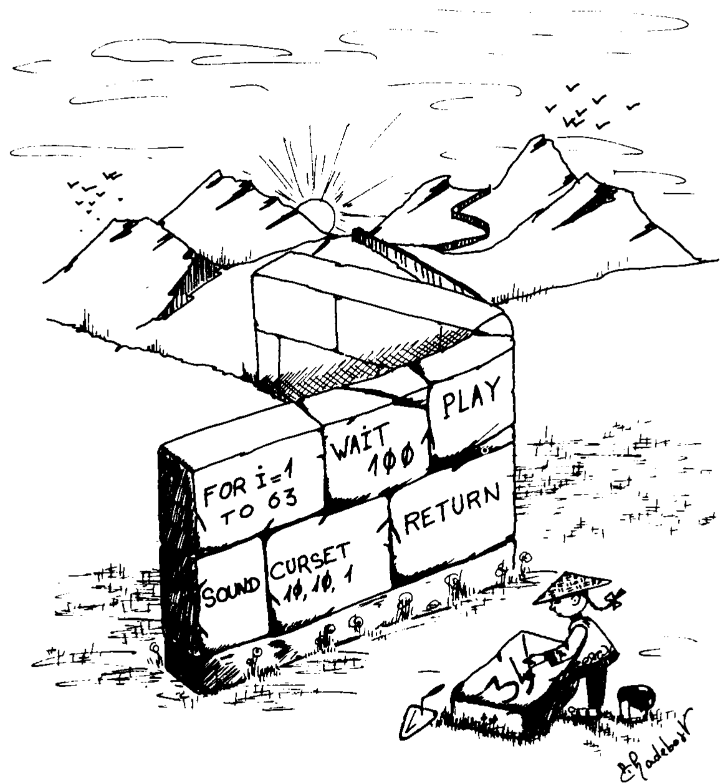
Après avoir utilisé ce programme vous pourriez souhaiter le conserver sur cassette. Le chapitre 6 vous indique la marche à suivre.

Avant de poursuivre assurez-vous de la compréhension des quelques notions de BASIC que nous venons d'exposer. Étudiez dans le chapitre 9 les instructions que nous avons rencontrées.

Nous poursuivrons cette introduction au BASIC en décrivant la manière dont les données sont stockées dans l'ORIC et la façon de les manipuler.

William Saint-Cricq / wsaintcr@wanadoo.fr Page 17 sur 163 15/08/2000

Numérisation http://wsc.n3.net



Chapitre 3
Programmer en BASIC

Nous avons maintenant vu quelques-unes des instructions du BASIC de l'ORIC, et comment elles se combinent dans un programme. Il y a trois éléments essentiels à n'importe quel programme, quelle qu'en soit la difficulté, que l'on veuille obtenir des combats palpitants de hordes ennemies ou des chiffres ennuyeux de TVA comme résultat final. L'ORIC, ou n'importe quel autre ordinateur, a besoin de données (qui peuvent être définies à l'intérieur du programme par l'instruction **INPUT**), plus un ensemble d'instructions sur la manière de traiter les dites données et afficher le résultat. Avant d'avancer vers l'introduction d'instructions BASIC plus complexes nous avons besoin d'en savoir plus sur la façon dont l'ORIC traite les données.

Nous avons les deux types de données — numérique et chaîne — que l'ORIC peut traiter. Les variables numériques que nous utilisons sont réelles ou à virgules flottantes (des nombres décimaux ordinaires en fait). Le plus grand nombre que l'ORIC peut contenir est 1,7014E+38 et le plus petit est 2,93874E−39. Ceux-ci sont exprimés en puissance (exponentielle), E ou notation scientifique, que l'ORIC lui-même utilise pour afficher les nombres hors de l'écart de 999999999 et 0,01. Essayez les deux programmes suivants pour voir comment l'ordinateur change son affichage selon que les nombres deviennent plus grands ou plus petits, et notez les résultats différents si l'on sort de l'écart disponible.

```
10 N=2
20 N=N*8
30 PRINT N
40 WAIT 10:GOTO 20
```

```
10 N=3
20 N=N/2
30 PRINT N
40 WAIT 10:GOTO 20
```

L'écriture exponentielle fournit une façon pour l'ORIC d'afficher de très grands ou de très petits nombres de manière concise. Les nombres en écriture exponentielle sont toujours affichés comme un décimal entre 1 et 9,99999999, suivis de E, suivis d'un nombre entre +38 et −39. La valeur de l'exposant indique le nom-

15/08/2000
Page 18 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

bre de chiffres dont il faut déplacer la virgule (vers la droite si l'exposant est positif, et vers la gauche s'il est négatif) pour obtenir le nombre correct. En d'autres termes, elle définit combien de fois la mantisse (la valeur décimale avant le E) doit être multipliée (E positif) ou divisée (E négatif) par 1Ø. Ainsi 1,23E+Ø9 donne 123ØØØØØØØ.Ø et 5,67E-Ø4 donne Ø,ØØØ567. L'ORIC a utilisé deux nombres à deux chiffres pour l'exposant, il comprend toujours le signe + pour les exposants positifs, mais il acceptera des nombres sans signe + ou qui ne commencent pas par Ø. Par exemple, 1,23E9 serait accepté comme une donnée valable. Essayez d'entrer des petits et des grands nombres, en écriture standard et exponentielle pour vous familiariser avec le système.

L'ORIC peut traiter des valeurs entières (nombre entier) entre 32767 et -32768, et il a un type séparé de variable pour stocker de tels nombres. Les variables entières sont identifiées en ajoutant **%** à un nom de variable autorisée. Ainsi **I%, N4%, WHOLE%** sont tous des noms autorisés pour des variables entières. On peut leur attribuer des valeurs non-entières, mais dans ce cas le nombre sera arrondi à l'entier inférieur (pas à l'entier le plus proche). Entrez le programme suivant pour voir ceci en pratique :

```
10 LET B%=6
20 B%=B%/2.23
30 PRINTB%
40 REEL=4.845
50 ENTIER%=REEL
60 PRINTENTIER%
70 B%=-3.4
80 PRINTB%
```

Notez particulièrement ce qui arrive à un nombre négatif. En arrondissant inférieurement des nombres négatifs, il faut se rappeler que -1, est plus petit que zéro, -2 plus petit que -1 etc.

D'où l'on déduit que le nombre entier inférieur le plus proche à -4,67 est -5 et ainsi de suite. Les variables entières sont traitées plus vite dans les calculs que les variables à virgule flottante, et prennent aussi moins de place en mémoire, dont il faut les utiliser aussi souvent que possible si la vitesse ou l'économie de mémoire sont importantes pour un programme.

L'ORIC reconnaît aussi les nombres hexadécimaux. Ce sont des nombres représentant un système de numération différente qui est utilisée fréquemment en programmation, car il permet une représentation commode et concise de nombres binaires. Les nombres binaires sont les suites de Ø et de 1 utilisées par l'ORIC (et tous les autres ordinateurs) pour contenir toutes les formes de mémoire de données.

La représentation binaire est utilisée parce que les contacts élec-

troniques qui sont le fondement même des ordinateurs ne peuvent qu'être actifs ou non-actifs, et chaque "contact" est utilisé pour représenter 1 quand il est actif et Ø quand il est non-actif. Vous en saurez plus sur les systèmes de numérations plus tard (voir les chapitres 5 et 10), mais pour le moment tout ce qu'il faut savoir est que l'ORIC acceptera, et dans un programme et comme donnée, n'importe quoi précédé du dièze #, qui est un nombre héxadécimal autorisé. Essayer d'afficher **#FFFF, #A, #1Ø**, comme exemples, et jetez un coup d'œil plus loin si vous êtes impatient de connaître l'explication. L'ORIC a des programmes internes qui convertissent tous les nombres binaires en messages que nous, humains, pouvons manier plus facilement, tels que des nombres décimaux (et hexadécimaux, pour les fanatiques de l'ordinateur) et vice versa, quand nous entrons des données numériques.

Nous avons introduit des variables chaîne et numérique mais puisque nous venons de dire que toute l'information dans votre ORIC est sous la forme de nombres binaires, comment obtenir des lettres et des symboles sur l'écran ? Et bien, ce sont des nombres (binaires) aussi. Chaque caractère qui apparaît sur l'écran a un code de caractère associé (connu sous le nom de ASCII). La suite complète de tous les codes des caractères est donnée dans l'Annexe 1.

Nous passons maintenant aux fonctions du BASIC.

Le BASIC de l'ORIC contient des fonctions qui réalisent des manipulations variées sur les données qu'il contient. Certaines nous permettent de convertir un type de donnée dans un autre. Nous pouvons trouver le code numérique d'un caractère en utilisant la fonction chaîne **ASC**, l'écriture utilisée avec **ASC** est la suivante : il faut placer le caractère dont nous voulons le code entre parenthèses. Essayez d'entrer :

PRINT ASC ("A")

Vous trouverez que 65 s'affiche, ce qui est le code ASCII pour A. Le programme ci-dessous répète ce processus pour n'importe quel caractère (majuscule ou non) que vous entrez par **A$**, en utilisant l'instruction **GET**. Notez que la chaîne à l'intérieur des parenthèses peut être ou bien une chaîne de caractères, comme ci-dessus ou, comme dans le programme, une variable chaîne. Toutes les fonctions demandent ce qui est connu sous le nom d'*argument* (ce qui est entre parenthèses) sur lequel elles opèrent, et on dit qu'elles renvoient un *résultat* (le code ASCII de l'argument chaîne, dans ce cas)

```
10 PRINT"APPUYEZ SUR UNE TOUCHE"
20 GET A$
30 PRINT"VOUS AVEZ APPUYE SUR "A$
40 PRINT"LE CODE ASCII DE "A$" EST "ASC(A$)
50 GOTO 10
```


William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

Quand vous en aurez assez d'appuyer sur des touches et de voir les codes ASCII qui s'affichent, essayez d'arrêter le programme en utilisant **CTRL-C**. Vous vous apercevrez qu'au lieu d'arrêter le programme, on vous informe que vous avez appuyé sur un blanc et que le code pour un blanc est 3. Le code ASCII pour **CTRL-C** est 3 mais comme c'est un caractère non affichant, vous n'obtenez rien sur l'écran. Vous devrez utiliser le bouton **RESET** sous votre ORIC pour arrêter le programme. (Vous aurez probablement besoin d'un stylo bille ou de quelque chose de similaire pour atteindre le bouton à l'intérieur du trou carré).

Remarquez que puisque ce programme n'utilise pas les ";" comme séparations entre les éléments d'information affichés, la ligne **PRINT**, bien qu'elle fonctionne, n'est pas aussi facile à lire que si des ";" étaient utilisés.

La fonction inverse de **ASC** est **CHR$**. Celle-ci associe à un nombre entre Ø et 255, qui est l'intervalle autorisé de l'argument, une chaîne contenant le caractère dont le code ASCII est ce nombre. Après avoir appuyé sur **RESET**, ajoutez la ligne 45 au programme, selon le listing ci-dessous :

```
10 PRINT"APPUYEZ SUR UNE TOUCHE"
20 GET A$
30 PRINT"VOUS AVEZ APPUYE SUR "A$
40 PRINT"LE CODE ASCII DE "A$" EST "ASC(A$)
50 GOTO 10
```

La ligne 40 affecte le nombre renvoyé par **ASC(A$)** à la variable C. Ceci est ensuite inserré au milieu de l'instruction **PRINT** (qui elle utilise des ";") et pris comme argument pour **CHR$** à la fin de la ligne. Appuyez sur **RUN**. De nouveau vous devrez utilisez le bouton **RESET** pour arrêter le programme. Remarquez que parce que la valeur numérique **C** a été utilisée il y a à la fois un espace devant et derrière le nombre. L'écriture correcte pour **CHR$** est **CHR$(I)**, où I est un nombre de Ø à 255, sans espaces, comme dans la fonction réelle à la fin de la ligne 40.

Les valeurs numériques et chaînes peuvent être traduites en utilisant **VAL** et **STR$**. **VAL** a l'écriture **VAL(a$)**, où **a$** signifie n'importe quelle chaîne littérale ou variable avec des caractères initiaux que l'ORIC peut interpréter comme un nombre. Essayez ce programme pour vous expérimenter.

```
10 PRINT"ENTREZ UN NOMBRE"
20 INPUT NOMBRE$:NOMBRE=VAL(NOMBRE$)
30 PRINT"LA CHAINE ";NOMBRE$;" DEVIENT LE
   NOMBRE ";NOMBRE
40 PRINT "2*";NOMBRE;"=";2*NOMBRE
50 GOTO 10
```

VAL évaluera et renverra le résultat de n'importe quelle chaîne interprétable comme nombre, jusqu'au premier caractère non numérique. Essayez d'entrer **23DF**, **+45**, **#F6** (écriture hexadécimale), **# 2BK** (K n'est pas un caractère héxadécimal autorisé), **1E4** (notation exponentielle), **23,E67**, **−23,6E9**, et **2*3** pour illustrer l'action de **VAL** sur des chaînes différentes parce que nous utilisons **INPUT** dans ce programme, vous pouvez utiliser **CTRL-C** pour arrêter le programme juste après **INPUT**.

La fonction inverse de **VAL** est **STR$**, une fonction chaîne qui renvoie la chaîne contenant le résultat d'évaluation d'une expression numérique. L'expression où le nombre est évalué par les programmes arithmétiques de l'ORIC puis est transformé en chaîne, donc on obtient le même résultat (mais sous forme de chaîne) que si un nombre avait été affiché sur l'écran. Ainsi des expressions complexes (qui peuvent inclure d'autres fonctions) peuvent être transformées dans l'équivalent chaîne de leur résultat. On peut avoir des lignes de cette sorte : **PRINT STR$ (#A4/12. 4E7 + 5*−3 4↑3)**

```
PRINT STR$(#A4/12.4E7+5*-3.4^3)
```

et, pour illustrer l'utilisation d'autres fonctions dans l'argument de **STR$**, essayez :

```
10 LET A$="3.567E7"
20 PRINT STR$(123/12.4*VAL(A$))
```

Il est important de noter à propos des fonctions qui impliquent un calcul arithmétique qu'elles ont la priorité absolue dans l'évaluation d'une expression, au-dessus de l'exponentiation. Le résultat d'application d'une fonction à un argument est évalué d'abord, avant tout autre calcul.

William Saint-Cricq / wsaintcr@wanadoo.fr Page 20 sur 163 15/08/2000

L'autre fonction chaîne pour la conversion numérique est **HEX$**. Celle-ci associe la forme chaîne au nombre hexadécimal équivalent à l'argument. Par exemple **HEX$ (418)** associe **#1A2**. Puisque les nombres hexadécimaux ne peuvent être que des entiers entre **Ø** et 65535, n'importe quel nombre hors de cet intervalle fera apparaître **?ILLEGAL QUANTITY ERROR** (Erreur : quantité interdite), et toute valeur non entière sera automatiquement arrondie.

Une fonction numérique associée à celles dont nous venons de traiter est **INT**. Celle-ci réalise la même opération que sur les variables entières et, comme on vient de le noter, sur les nombres hexadécimaux soit arrondir un nombre à l'entier inférieur le plus proche. On utilise **INT** toutes les fois que l'on veut la partie entière d'un nombre, et l'ORIC fait la même chose automatiquement pour les arguments des fonctions qui demandent une valeur entière. Essayez **PRINT CHR$ (67,9)** par exemple, et **C** apparaîtra sur l'écran l'ORIC arrondit au chiffre inférieur et affecte 67 comme valeur de l'argument, puisque les codes de caractères ne peuvent être que des nombres entiers. Un emploi utile de **INT** est d'arrondir des nombres à un nombre précis de chiffres après la virgule. Ceci est basé sur le fait suivant : tandis que **INT(n)** arrondit au chiffre inférieur, **INT(n + 0,5)** arrondit à l'entier le plus proche. Ainsi **INT(3,7)** donne 3, mais **INT(3,7 + 0,5)** équivaut à **INT(4,2)** c'est-à-dire **4**. Arrondir 4,567 à deux chiffres après la virgule, par exemple, fait arrondir ,Ø67 à ,Ø7. On peut écrire une ligne de programme pour faire cela en multipliant par 1Ø à la puissance du nombre de chiffres après la virgule que l'on veut, en ajoutant Ø,5, en utilisant **INT** sur cette valeur, puis en divisant par la même puissance de 1Ø. On peut produire un programme qui arrondira à un nombre précis de décimales.

```
10 REM *ARRONDIR A N DECIMALES*
20 INPUT"ENTREZ UN NOMBRE";N
30 INPUT"COMBIEN DE DECIMALES";DEC
40 LET ARR=(INT(N*10^DEC+0.5))/10^DEC
50 PRINT ARR
```


Nous avons vu comment on peut convertir des chaînes en nombres et vice versa, mais comment agir sur des chaînes en tant que chaînes ? Et bien, on peut extraire des portions de chaînes en utilisant **LEFT$** (gauche), **RIGHT$** (droite) et **MID$** (milieu) et en ajoutant des chaînes ensemble. L'addition de chaînes (appelée concaténation) utilise le signe + et relie simplement les chaînes ensemble comme suit :

```
10 LET A$="sur ORIC"
20 INPUT"Quel est votre nom";NOM$
30 LET B$="lecon de BASIC"
40 LET ESPACE$=" ":REM espace simple
50 LET A$=B$+ESPACE$+A$
60 LET BNJ$="Salut "+NOM$+",content que vous
   preniez"
70 PRINT BNJ$
80 LET BASIC$="votre"+ESPACE$+A$
90 PRINT BASIC$
```

Pour illustrer le découpage de chaînes voici un exemple de **LEFT$** en action :

```
10 A$="BONNET"
20 B$="JOURNEE"
30 C$="LESSIVEUSE"
40 D$="ISIDORE"
50 L$=LEFT$(A$,3)
60 AN$=LEFT$(B$,4)
70 PRINT L$+AN$
80 R$=LEFT$(C$,3)
90 PRINTR$
100 Y$=LEFT$(D$,2)
110 Z$=LEFT$("AMERIQUE",2)+Y$
120 PRINT Z$
```

L'argument de **LEFT$** est la chaîne d'où les caractères doivent être extraits, et la valeur du nombre exprimé après la virgule indique combien de caractères sont requis à partir du début de la chaîne.

Toutes les fonctions de l'ORIC, avec les autres mots clés du BASIC sont définis et illustrés au chapitre 9 auquel il faut vous rapporter pour toute expression inconnue rencontrée dans ce manuel et pour toute information supplémentaire concernant l'introduction au BASIC. Reportez-vous au chapitre 9 pour **RIGHT$** et **MID$** et la fonction de fin de chaîne, **LEN**. Cette fonction donne la longueur (**LEN**gth) d'une chaîne, c'est-à-dire le nombre de caractères de la

Numérisation http://wsc.n3.net
15/08/2000 Page 21 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

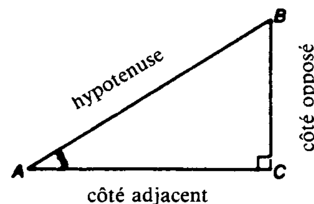
chaîne. Le BASIC de l'ORIC a les fonctions numériques prédéfinies suivantes :

ABS(n)	Valeur absolue de n
ATN(n)	Arc tangente n
COS(n)	Cosinus de n
EXP(n)	Exponentielle de n
INT(n)	Partie entière de n
LN(n)	Logarithme népérien de n (base e)
LOG(n)	Logarithme décimal de n (base 1Ø)
PI	Nombre π
RND(n)	Nombre aléatoire
SGN(n)	Ø si n=Ø, 1 s'il est positif, −1 s'il est négatif
SIN(n)	Sinus de n
SQR(n)	Racine carrée de n
TAN(n)	Tangente de n

Notez que **PI** n'a pas d'argument mais donne seulement la valeur de la constante PI(π). Les fonctions numériques effectuent de difficiles calculs pour lesquels il faudrait autrement écrire des programmes complexes à chaque fois qu'on voudrait les inclure dans des calculs.

Prenons **COS** comme exemple de fonction numérique. Celle-ci calcule le rapport trigonométrique fondamental du cosinus pour l'angle défini par la valeur de l'expression entre parenthèses.

Dans l'écriture **COS(n)** l'argument n peut être un nombre, une variable numérique ou une expression (qui peut inclure d'autres fonctions numériques). L'angle est mesuré en radians et non pas en degrés (360 degrés = 2*PI radians). Pour le triangle rectangle dessiné ci-dessous, le cosinus de l'angle en A sera le rapport du côté adjacent sur l'hypothénuse (AC/AB).



A moins que vous ne soyez expert en trigonométrie, il est peu probable que vous utilisiez beaucoup les fonctions trigonométriques de l'ORIC. Elles sont cependant utiles pour définir des positions sur l'écran en mode de haute résolution graphique (HIRES) dont nous parlerons au chapitre 7.

A côté des fonctions numériques pré-programmées de l'ORIC dont nous venons de voir **COS** et **INT**, il y a une façon de **DEF**inir vos propres FoNctions en utilisant l'énoncé **DEF FN.**

L'instruction **DEF FN**. L'utilisateur (vous donc !) donne une définition de la fonction désirée au début du programme, avec un énoncé sous la forme de : **DEF FN v(z)** = expression numérique, le v est le nom de la fonction (une seule lettre de A à Z) et le z est un nom de variable numérique ordinaire. L'expression qui suit est une expression numérique qui peut comprendre n'importe quelle autre fonction numérique, et utilise la variable z figurant dans l'expression. La fonction définie est alors appelé de la même manière qu'une fonction type, avec le nom (**FN** suivi de la lettre choisie) et une valeur ou paramètre entre parenthèses. Regardons un exemple :

```
5 REM DEFinition de FoNction qui
   convertit des pieds en metres
10 DEF FNM(PDS)=PDS*0.3048
20 INPUT "COMBIEN DE PIEDS";X
30 M=FNM(X)
40 PRINT X;"PIEDS EGAL";M;"METRES"
```

On a défini une fonction appelée M, avec un argument **FEET**, telle que la fonction M multiplie la variable **FEET** par le facteur de conversion 0,3048 pour donner le nombre équivalent en mètres. Quand le nombre de pieds est entré à la ligne **2Ø**, la valeur est affectée à la variable X. A la ligne **3Ø**, on donne à la variable M (ce qui n'est pas la même chose que la fonction M) la valeur **FN M(X)**, ce qui signifie "chercher la fonction M, et calculer la valeur qui en découle, en utilisant la valeur de X pour remplacer le nom de la variable employé dans l'argument et l'expression suivant les signes égal". La variable entre parenthèses est connue sous le nom de "variable muette", car elle ne sert qu'à définir une séquence d'opérations et est remplacée par une variable dont la valeur a été définie dans le programme quand on appelle la fonction en utilisant **FN**. **A** n'importe quel endroit du programme (pourvu que ce soit après l'instruction **DEF FN**) on peut utiliser les fonctions définies, en

William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Numérisation http://wsc.n3.net

choisissant une valeur à la place de la variable muette. Voici un autre exemple

```
10 REM *DEF FoNction de Pouces en cm,
          et aire du disque
20 DEF FNC(P)=P*2.54
30 DEF FNA(R)=PI*R^2
40 INPUT"Rayon du cercle(en Pouces)";RA
Y
50 CMRAY=FNC(RAY)
60 AIRE=FNA(CMRAY)
70 PRINT"Aire du disque:";AIRE;"cm2"
```

La ligne 20 utilise la variable P pour définir une fonction de conversion (FNC) d'inches en centimètres, et la ligne 30 définit FNA afin de donner l'aire d'un cercle (πr^2) en utilisant la variable muette R et la valeur pré-définie PI. La ligne 50 utilise la valeur d'entrée (INPUT) RAD comme paramètre pour FNC (en remplaçant la variable muette I) pour obtenir le rayon en centimètres ; CMRAD qui est à son tour utilisé comme paramètre pour FNA à la ligne 60. Puisque l'on peut utiliser n'importe quelle fonction dans la DEFinition, comme pour PI, les lignes 50 et 60 peuvent se combiner :

```
10 REM *DEF FoNction de Pouces en cm,
          et aire du disque
20 DEF FNC(P)=P*2.54
30 DEF FNA(R)=PI*R^2
40 INPUT"Rayon du cercle(en Pouces)";R
AY
50 AIRE=FNA(FNC(RAY))
60 PRINT"Aire du disque:";AIRE;"cm2"
```

En fait, une seule fonction prendra en charge tout le calcul :

```
10 REM DEF FN aire en cm2 d'un disque
     dont le rayon est en Pouces
20 DEF FNA(RAY)=(RAY*2.54)^2*PI
30 INPUT"Rayon du cercle(en Pouces)";R
AY
40 AIRE=FNA(RAY)
50 PRINT"Aire du disque:";AIRE;"cm2"
```


Remarquez qu'ici nous avons utilisé le même nom pour la variable réelle utilisée dans le calcul que pour la variable muette. On peut utiliser autant de fonctions que l'on veut jusqu'à un maximum autorisé de 26 (lettres de l'alphabet) dans un programme pour éviter des calculs répétitifs.

Avant d'étudier les deux dernières méthodes de stockage de données disponibles dans le BASIC de l'ORIC, il faut vous présenter les boucles. Des opérations répétées sont fréquentes dans les programmes, et plusieurs types de structure de boucle peuvent être utilisés. La boucle simple **GOTO** que nous avons vu présente un sérieux inconvénient, on ne peut l'arrêter qu'en interrompant le programme. L'ORIC a une structure disponible : **FOR...NEXT**, que l'on peut utiliser toutes les fois qu'une suite d'instructions doit être répétée un nombre déterminé de fois. Une boucle **FOR...NEXT** est mise en place en utilisant une instruction de cette forme :

FORv = n1 TO n2 STEP n3

qui définit une variable v (variable numérique ordinaire) qui contrôlera la boucle, et deux expressions numériques qui donnent la valeur initiale (n1) et définissent la valeur finale (n2) de la boucle. L'instruction **STEP** peut être omise si la valeur de **STEP**(n3) est égale à 1. La variable de la boucle prend la valeur initiale, et le programme continue jusqu'à ce qu'il rencontre une instruction **NEXT**v. La valeur de la variable de la boucle est vérifiée, et si elle n'est pas plus grande ou égale à la valeur finale la valeur **STEP** est ajoutée, et le programme retourne à l'instruction suivant, **FOR**... Remarquez que c'est l'instruction, pas la ligne de programme suivant le **FOR**..., afin que l'on puisse avoir une boucle **FOR...NEXT** sur une seule ligne :

```
10 FOR L=1 TO 5:PRINT L*L:NEXT L
```

Si la valeur STEP est négative, la variable de la boucle est vérifiée pour voir si elle est inférieure à la valeur finale :

```
10 FOR L=10 TO 5 STEP -1:PRINT L*L:NEX
T L
```

La possibilité d'utiliser la valeur de la variable qui contrôle la boucle dans des expressions internes au corps de la boucle est très utile :

```
10 CLS
20 FOR BOUCLE=0 TO 25
30 PRINT CHR$(65+BOUCLE);CHR$(97+BOUCL
E);
40 NEXT BOUCLE
```


William Saint-Cricq / wsaintcr@wanadoo.fr Page 23 sur 163 15/08/2000

Les boucles **FOR...NEXT** sont souvent utilisées conjointement avec une autre manière de stocker des données dans un programme en employant **READ** et **DATA**. Une ligne de programme, ou une suite de lignes, commençant par **DATA** est utilisée pour contenir des chaînes ou des nombres, séparés par des virgules. Ceux-ci peuvent aller n'importe où dans le programme, et l'ORIC commence avec le premier élément d'information **DATA** quand il rencontre une instruction **READ**, qui place le nombre ou la chaîne dans la variable suivant **READ** (qui doit être du type convenant à la **DATA**). Un pointeur est ensuite placé en face de l'élément d'information **DATA** suivant, qui sera utilisé quand l'instruction **READ** suivante apparaîtra. **RESTORE** replacera le curseur au point de départ des éléments d'information **DATA**. Ce principe est montré dans le programme ci-dessous, qui utilise une boucle pour **READ** (lire) et **PRINT** (afficher) la chaîne **DATA** (des données), puis **RESTORE** (remet) le curseur de **DATA**, afin que les jours puissent être lus (**READ**) et affichés (**PRINT**ed) à nouveau. Remarquez que la variable boucle n'est pas indiquée après les instructions **NEXT**, ce qui est permis, mais source de confusion en puissance, et que la chaîne d'éléments d'information **DATA** n'a pas besoin de guillemets (bien qu'on puisse et qu'il faille en mettre si les chaînes doivent contenir des espaces au début ou à la fin, des virgules ou deux-points).

```
10 REM ....READ et DATA .........
20 FOR J=0 TO 6
30 READ J$
40 PRINT J$
50 NEXT
60 REM ...encore des lignes .....
70 '
2000 DATA LUNDI,MARDI,MERCREDI,JEUDI
2010 REM ...et des lignes intercalees
3000 DATA VENDREDI,SAMEDI,DIMANCHE
4000 REM ...on peut le refaire !...
4010 RESTORE
4020 FOR K=1 TO 7
4030 READ JOUR$:PRINT JOUR$
4040 NEXT
```

La dernière manière par laquelle l'ORIC peut contenir des chaînes ou des nombres est dans des tableaux. Ceux-ci se construisent normalement en utilisant l'instruction **DIM** et il vous faut consulter le paragraphe sur **DIM** dans le chapitre 9 avant d'aller plus loin.

Bon, vous savez en quoi consiste un tableau. L'intérêt des tableaux réside non seulement dans le fait qu'ils permettent le stockage d'éléments d'information similaires de données sans la définition d'une variable individuelle pour chaque élément séparément, mais aussi l'indexation de leur(s) indice(s). Ceci permet d'établir des correspondances et d'effectuer des opérations sur les éléments des tableaux. Voici un court exemple qui démontre la façon dont les tableaux permettent la programmation simple d'opérations compliquées sans cela :

```
10 INPUT" COMBIEN DE NOMS";N%
20 DIM NOM$(N%):DIM AGE(N%)
30 CLS:SOM=0
40 FOR K=1 TO N%
50 :    PRINT"NOM No"K
60 :    INPUT NOM$(K)
70 :    PRINT "AGE DE "NOM$(K)
80 :    INPUT AGE(K):SOM=SOM+AGE(K)
90 :    CLS
100 NEXT K
110 FOR K=N% TO 1 STEP -1
120 :   PRINT:PRINT NOM$(K)" A ";
130 :   PRINTAGE(K)" ANS"
140 NEXT K
150 PRINT:PRINT"TOTAL DES AGES :"SOM"A
NS"
160 PRINT"AGE MOYEN:"SOM/N%"ANS"
170 END
```

On peut utiliser la variable **INPUT N%** pour déterminer les dimensions **DIM** de deux tableaux de la taille nécessaire et fixer les boucles **FOR...NEXT** pareillement. Notez que les humains trouvent souvent plus facile d'étudier des tableaux ayant des "indices" commençant par 1 comme ici. Ceci laisse simplement les éléments **NAME$(0)** et **AGE(0)** du tableau inutilisés. La valeur de la variable boucle est utilisée pour faire correspondre chaque couple d'éléments du tableau en séquence, une fois pour **INPUT** (l'entrée) et une fois pour **PRINT** (l'affichage). L'âge total peut être aisément calculé en même temps.

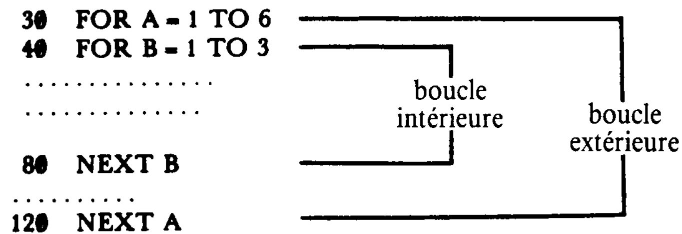
Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Numérisation http://wsc.n3.net

Des boucles à l'intérieur des boucles sont connues sous le nom de boucles *emboîtées*. On peut avoir autant de niveau "d'emboîtement" que l'on veut jusqu'à un maximum de 10. Il est important d'emboîter correctement les boucles :



Des boucles emboîtées peuvent être utilisées pour faire correspondre et débuter des éléments de tableau multidimensionnel :

```
10 DIM N%(10,3)
20 CLS
30 FOR K=1 TO 10
40 :    FOR J=1 TO 3
50 :    :    N%(K,J)=K^J
60 :    :    PRINT N%(K,J)
70 :    NEXT J
80 :    PRINT
90 NEXT K
```

DATA (les données) peuvent aussi se lire (READ) en tableaux :

```
10 DIM M$(12):DIM J%(12)
20 FOR K=1 TO 12
30 :    READ M$(K)
40 :    READ J%(K)
50 NEXT K
60 INPUT"No du mois ";MOIS
70 PRINT M$(MOIS);" A";J%(MOIS);"JOURS
"
100 DATA JANVIER,31,FEVRIER,28,MARS,31
,AVRIL,30,MAI,31,JUIN,30
110 DATA JUILLET,31,AOUT,31,SEPTEMBRE,
30,OCTOBRE,31,NOVEMBRE,30,DECEMBRE,31
```

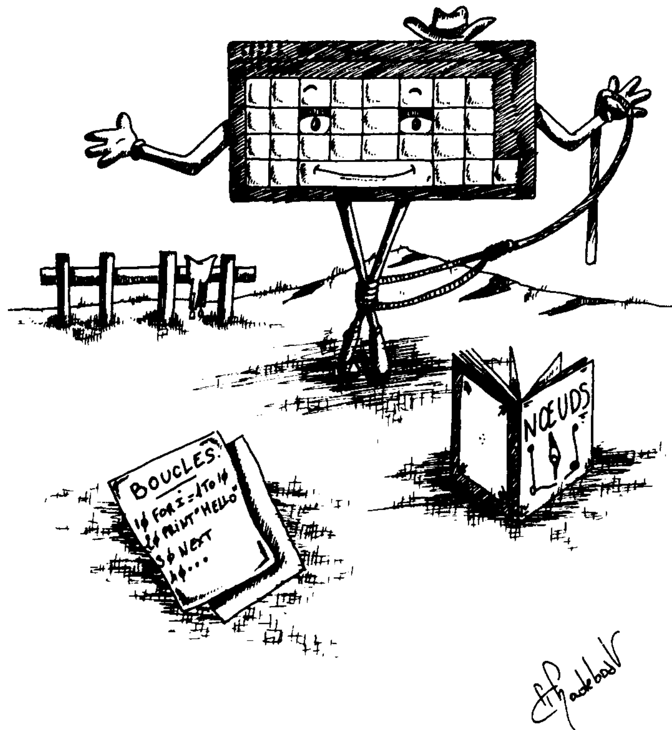

L'ORIC stocke l'adresse de la ligne ou de l'instruction après l'instruction **FOR...TO** (à laquelle le contrôle du programme retourne en rencontrant une instruction **NEXT**) en une pile : dernière entrée, première sortie. C'est la raison pour laquelle le nom de la variable peut être omis (cela ne dérange pas l'ORIC mais la mention de la variable aide l'humain à lire un listing de programme) et c'est aussi pour cela que les boucles doivent être correctement emboîtées, puisque l'ORIC ne regarde que l'adresse sur le dessus de la pile. A la fin d'une boucle, l'adresse disparaît et un **NEXT** accélérera le passage vers la nouvelle adresse en dessus de pile. La pile ne peut contenir que 10 adresses et plus de 10 boucles donneront un **?OUF OF MEMORY ERROR** (erreur dépassement de mémoire).

La même pile est utilisée pour l'autre type de boucle disponible sur l'ORIC, la structure **REPEAT...UNTIL.** Nous traiterons de ceci dans le chapitre suivant.

William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Numérisation http://wsc.n3.net



Chapitre 4
Des boucles sans comparaison

Un lecteur attentif aura pu remarquer que la structure de boucle **FOR...NEXT** implique que l'ORIC est capable de dire si un nombre est plus grand qu'un autre, puisque la variable boucle doit tester la valeur de la fin de la boucle avant que l'ORIC ne décide de reparcourir la boucle ou de continuer avec l'instruction suivante.

Et bien, nous pouvons aussi faire cela en BASIC, et en fait cette capacité est capitale pour les programmes, puisque des décisions peuvent être prises, et la succession d'actions aiguillée vers d'autres séquences d'opérations. L'introduction des boucles **REPEAT** (répétez)... **UNTIL** (jusqu'à) met cette question au premier plan puisque les structures **REPEAT... UNTIL** répètent une suite d'opérations jusqu'à ce qu'une condition soit remplie.

Les conditions sont testées au moyen d'opérations conditionnels :

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

Ceux-ci opèrent quasiment comme pour les valeurs numériques. Il faut se rappeler deux choses cependant. La première est que, pour les nombres négatifs -6 sera inférieur à -5, et ainsi de suite, et la seconde est que l'ORIC, comme n'importe quel ordinateur, n'est pas totalement exact dans ses calculs. Lorsque des opérations numériques complexes ont été effectuées pour produire un nombre dont l'égalité doit être testé avec un autre, il est possible que le chiffre le moins significatif soit erroné. Dans ces cas-là il est préférable de prendre la valeur absolue **(ABS)** (positive) des deux chiffres, puis de tester que la différence est négligeable. Rapportez-vous au paragraphe **ABS** au chapitre 9 et regardez le programme ci-dessous :

```
10 LET NEUF AU CUBE = 9*9*9
20 IF NEUF AU CUBE = 9^3 THEN PRINT "
VRAI" ELSE PRINT" FAUX"
30 IF ABS(NEUF AU CUBE-9^3)<1E-6 THEN
PRINT " VRAI" ELSE PRINT " FAUX"
```


15/08/2000
Page 26 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

En quoi consiste l'instruction **IF...THEN...ELSE** à la ligne 2Ø ? Elle teste si la variable NINECUBE (9^3) est égale à 9↑3, et décide si la condition est vraie ou fausse. Si **(IF)** la condition est vraie alors **(THEN)** elle effectue les opérations indiquées après **THEN**, et passe à la ligne de programme suivante, en ignorant **ELSE** (autrement) et tout ce qui vient après. Si **(IF)** la condition est fausse, **THEN** et la ou les instruction(s) suivante(s) sont ignorées, et la ou les instruction(s) suivant **ELSE** sont effectuées avant de passer à la ligne d'après. Donc on a une structure de programme qui dit : Si **(IF)** la condition est vraie alors **(THEN)** effectuez la vraie tâche autrement **(ELSE)** effectuez la tâche fausse.

A la ligne 2Ø, cependant, la condition sera évaluée comme fausse et **FALSE** (faux) s'affichera, à la ligne 3Ø la condition sera évaluée comme vraie et **TRUE** (vrai) s'affichera.

La partie **ELSE** d'une construction **IF...THEN** est facultative. Quand il n'y aura pas d'instruction **ELSE**, le contrôle passera à la ligne suivante, avec les instructions qui suivent **THEN** exécutées si la condition était vraie et ignorées si la condition était fausse.

```
10 A=2:INPUT"ENTREZ UN NOMBRE ENTRE 0
ET 5";C
20 IF C>A THEN PRINT"C>A"
30 PRINT"CECI S'AFFICHE TOUJOURS"
40 GOTO 10
```

Si nous avions voulu utiliser **ELSE**, nous aurions pu écrire une ligne 2Ø comme ci-dessous :

```
10 A=2:INPUT"ENTREZ UN NOMBRE ENTRE 0
ET 5";C
20 IF C>A THEN PRINT"C>A" ELSE PRINT "
C<A"
30 PRINT"CECI S'AFFICHE TOUJOURS"
40 GOTO 10
```

Cependant si vous entrez 2 comme valeur de C, vous obtiendrez "C<A". Est-ce le résultat que nous voulons, demanderez-vous ? Et bien nous sommes maintenant dans le royaume de la logique, et nous devons surveiller nos compléments. L'une ou l'autre des instructions THEN et ELSE doit être exécutée. Les ordinateurs sont strictement incapables d'une indécision, donc c'est ou bien une chose ou bien une autre. Ce qu'il faut faire est de reconnaître les vrais compléments, puisque l'erreur vient de nous, pas de l'ORIC.

Le contraire, ou condition complémentaire de > est < =, non pas <. Les relations sont :

< > complète =
< complète > =
> complète < =

Donc notre sortie logiquement correcte sera produite par :

```
10 REM Compteur
20 C=1
30 PRINT "VALEUR DU COMPTEUR"C
40 C=C+1
50 IF C<4 GOTO 30
60 END
```

Avec l'utilisation d'un test conditionnel **IF...THEN**, on peut produire l'équivalent d'une boucle **FOR...NEXT**, appelé une boucle-compteur. Le programme ci-dessous fixe un compteur C, l'incrémente de 1 en ligne **4Ø** et teste la valeur à la ligne **5Ø**, après le corps de la boucle. Une variante de l'écriture **IF...THEN** est employée. Si l'instruction après **THEN** est **GOTO** (numéro de ligne), elle peut être remplacée par **IF...GOTO**, en omettant le **THEN**, ou par **IF...THEN** (numéro de ligne) en omettant le **GOTO**. L'ORIC interprète toutes ces écritures de la même façon, **GOTO** peut de la même façon être omis après **ELSE** si l'on veut.

```
10 REM Compteur
20 C=1
30 PRINT "VALEUR DU COMPTEUR"C
40 C=C+1
50 IF C<=4 GOTO 30 ELSE END
```

Si la condition **C<4** est vraie, le programme retournera en ligne 3Ø. Remarquez qu'avec cette condition la boucle sera exécutée trois fois. Si l'on remplace la condition de la ligne 5Ø par **C< =4** elle sera exécutée 4 fois et sera un équivalent exact d'une boucle **FOR C=1 TO 4**. L'utilisation de **ELSE** donnerait :

```
10 REM Compteur
20 C=1
30 PRINT "VALEUR DU COMPTEUR"C
40 C=C+1
50 IF C<=4 GOTO 30 ELSE END
```


William Saint-Cricq / wsaintcr@wanadoo.fr 15/08/2000

La construction **REPEAT... UNTIL** commence avec une instruction **REPEAT** et le programme exécute les instructions dans le corps de la boucle jusqu'à ce qu'(**UNTIL**) une condition soit remplie. Si la condition est vraie, la boucle s'arrête, et le programme continue avec l'instruction après **UNTIL** (condition). Si la condition est fausse le programme retourne à l'instruction après **REPEAT**. Celle-ci peut être utilisée pour la même sorte de boucle-compteur :

```
10 REM Boucle REPEAT....UNTIL
20 C=1
30 REPEAT
40 PRINT "REPEAT :VALEUR"C
50 C=C+1
60 UNTIL C=4
```

Cependant, la condition **UNTIL** peut se rapporter à n'importe quelle condition, c'est une structure extrêmement souple. Le programme suivant utilise une boucle **REPEAT...UNTIL** pour calculer la factorielle d'un nombre :

```
10 REM .....FACTORIELLE........
20 INPUT"DONNEZ UN ENTIER";J%
30 N%=J%-1:FACT=J%
40 REPEAT
50 :    FACT=FACT*N%
60 :    N%=N%-1
70 UNTIL N%=0
80 PRINT"FACTORIELLE";J%;"=";FACT
```

Elles peuvent être utiles dans les programmes d'entrée de données :

```
10 REPEAT:UNTIL KEY$="G"
20 PRINT"END"

10 REPEAT
20 INPUT"DONNEZ UN ENTIER PLUS GRAND Q
UE 9";N%
30 UNTIL N%>9
40 REM...RESTE DU PROGRAMME.......
```


L'ORIC établit la vérité ou la fausseté d'une condition par comparaison entre des nombres et, puisqu'il ne connaît rien d'autre, affecte aussi une valeur numérique (– 1) pour représenter le vrai et considère Ø comme faux. Il y a des mots clés **TRUE** et **FALSE** dans le BASIC de l'ORIC pour représenter ces valeurs. Alors qu'elles ne font que se substituer aux valeurs numériques, leur emploi permet de rendre un programme plus accessible au cerveau humain. Voici une boucle sans fin :

```
10 REPEAT
20 REM ...AFFICHAGE PERPETUEL
30 PRINT"ON CONTINUE"
40 UNTIL FALSE
```

où nous pourrions remplacer la condition **N% = Ø** par **N% = FALSE** dans le programme de factorielles :

```
10 REM  * FACTORIELLES *
20 INPUT "DONNEZ UN ENTIER";J%
30 N%=J%-1:FACT=J%
40 REPEAT
50 :     FACT=FACT*N%
60 :     N%=N%-1
70 UNTIL N%=FALSE
80 PRINT"FACTORIELLE"J%"="FACT
```

Il est important de noter qu'alors que l'ORIC considère toujours **Ø** comme **FALSE** (faux), et en évaluant une instruction conditionnelle, considère – 1 comme **TRUE** (vrai), il acceptera n'importe quel nombre non nul comme **TRUE** (vrai) en testant des variables numériques.

```
5 REPEAT
10 INPUT A
20 IF A THEN PRINT"A N'EST PAS NUL" EL
SE PRINT" A=0,VALEUR POUR 'FALSE'"
30 UNTIL A=FALSE
```

Les caractères et les chaînes peuvent être comparés dans des tests conditionnels, et il faut souvent comparer des chaînes pour découvrir si elles sont égales. La simple comparaison de chaînes pour

Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr Page 28 sur 163 15/08/2000

Numérisation http://wsc.n3.net

rechercher des listes ou tester des entrées pose peu de problèmes puisque l'égalité est la seule chose à être testée :

```
 10 PRINT"VOUS APPELEZ-VOUS FRED "
 20 INPUT"REPONDEZ OUI OU NON";A$
 30 IF A$="OUI" THEN PRINT"SALUT,FRED !
"
 40 IF A$="NON" THEN PRINT"VOUS NE POUR
RIEZ PAS MENTIR UN PEU ?"
 50 PRINT"BIEN,QUI QUE VOUS SOYEZ,DONNE
Z LE MOT DE PASSE"
 60 INPUT PASS$
 70 IF PASS$<>"FX123"THEN PRINT"VOUS ET
ES UN IMPOSTEUR":END
 80 PRINT "AVE FRATER"
```

Cependant, on doit faire attention en comparant des chaînes entre elles dans le but de mettre en ordre. La comparaison entre les chaînes est effectuée en regardant chaque code de caractère ASCII tour à tour, et tandis qu'en comparant, une différence d'un seul caractère nous indique ce qu'il faut connaître, en triant alphabétiquement nous avons le problème des majuscules et des minuscules et peut-être aussi des nombres. D'abord étudions un simple programme de tri pour mettre des nombres en ordre croissant. Chaque élément d'une liste (stocké dans le tableau NUM) est comparé avec chaque autre élément de la liste, et permuté avec n'importe quel nombre plus grand qu'il rencontre. La répétition de ce processus pour chaque nombre de liste donne l'ordre désiré. Notez l'utilisation de la variable **TEMP** pour transférer des éléments du tableau :

```
5 REM *  CLASSEMENT DE NOMBRES  *
10 CLS:LETJ=15 ' nombre d'elements
20 DIM NUM(J)  ' tableau de nombres
30 ' nombres tires au hasard
40 FOR K=0 TO J
50 :    LET NUM(K)=RND(1)*1000
60 :    PRINT NUM(K)
70 NEXT K
99 '
100 REM * PROGRAMME DE TRI  *
101 '
110 FOR M=1 TO (J-1)
120 :  FOR N=M TO J
125 :  : REM Si l'ordre est correct,
    :        sauter cette ligne
130 :  : IF NUM(M)<NUM(N) THEN 170
```



```
135 :  : REM Si l'ordre n'est pas
    : correct,echanger
140 :  : TEMP=NUM(M)
150 :  : NUM(M)=NUM(N)
160 :  : NUM(N)=TEMP
170 :  NEXT N
180 NEXT M
189 '
190 REM  *  FIN DU CLASSEMENT  *
191 '
200 REM Affichage de la liste finale
210 CLS:PRINT"LISTE FINALE:"
220 FOR K=1 TO J
230 :   PRINT NUM(K)
240 NEXT K
```

Le programme suivant est une version chaîne du tri de bulle donné ci-dessus. La seule différence est l'utilisation d'un drapeau (**FLAG**) pour empêcher que des comparaisons inutiles se fassent quand un passage n'a produit aucune permutation (éléments dans le bon ordre). Tapez-le selon la liste pour mettre en évidence la suite ordonnée que des chaînes contenant des types de caractère différents produisent, puis utilisez le programme interne de tri dans un programme personnel (où les données sont entrées en majuscules seulement) pour trier n'importe quelles listes alphabétiques :

```
5 REM ** CLASSEMENT ALPHABETIQUE **
10 CLS
15 'Initialise le tableau de chaines
 on peut le remplacer par des INPUT
20 A$(0)="1"
30 A$(1)="B"
40 A$(2)="ab"
50 A$(3)="Ab"
60 A$(4)="zz"
70 A$(5)="aZ"
80 A$(6)="D"
90 A$(7)="d"
100 A$(8)="X"
110 A$(9)="A2"
120 A$(10)="bwert"
125 '
130 REM *** AFFICHER LA LISTE ***
140 FOR K=0 TO 10
150 :   PRINT A$(K);" ";
```


William Saint-Cricq / wsaintcr@wanadoo.fr Page 29 sur 163 15/08/2000

```
160 NEXT
165 '
170 REM *  CLASSEMENT  *
180 FOR K=0 TO 9 ' nb de donnees -1
190 :     DRAPEAU=FALSE
200 :     FOR M=K TO 10
210 :     :     IF A$(K)<A$(M) THEN 260
220 :     :     TEMP$=A$(M)
230 :     :     A$(M)=A$(K)
240 :     :     A$(K)=TEMP$
250 :     :     LET DRAPEAU=TRUE
260 :     NEXT M
270 IF NOT DRAPEAU THEN K=99
280 NEXTK
285 '
290 REM * AFFICHAGE LISTE CLASSEE  *
300 PRINT:PRINT
310 FOR K=0 TO 10
320 :     PRINT A$(K)
330 NEXT
340 END
```

Des boucles **REPEAT...UNTIL** et **FOR...NEXT** peuvent aussi être emboîtées. Notez l'utilisation de blancs (qui nécessitent des deux-points au début de la ligne pour empêcher l'ORIC de les enlever) pour rendre plus claire la structure de boucle du programme.

```
10 REM * BOUCLES EMBOITEES *
20 PRINT"BOUCLES AVEC L'ORIC"
30 FOR F=1 TO 3
40 :    PRINT TAB(15+F);"FOR...NEXT"F
50 :    PRINT TAB(15+F);"MAINTENANT,REP
EAT"
60 :    C=1     'compteur
70 :    REPEAT
80 :    :  PRINT TAB(15+F+C);"REPEAT"C
90 :    :  C=C+1
100 :   UNTIL C>2
110 :   PRINT TAB(15+F);"HORS DE REPEAT
"
120 :   PRINT TAB(15+F);"MAINTENANT, F
SUIVANT"
230 NEXT
```


Il y a un point de programmation important à noter concernant les boucles **REPEAT... UNTIL** et **FOR...NEXT**. On doit en sortir correctement, en satisfaisant à la condition de sortie. Sauter hors d'une boucle avec un **GOTO** laissera l'adresse de la boucle sur la pile, et si vous le faites plusieurs fois, la pile se remplira et vous obtiendrez une **?OUT OF MEMORY ERROR** (erreur : sortie de mémoire). On peut sortir des boucles **FOR... NEXT** en donnant à la variable de la boucle une valeur qui satisfera le test de sortie :

```
10 PRINT"ENTREZ DES NOMBRES,OU -999 PO
UR FINIR
15 DIM A(20),D(20)
20 FOR F=1 TO 20
30 INPUT A(F)
40 IF A(F)=-999 THEN A(F)=0:F=21:GOTO
60
50 LET D(F)=A(F)*31
60 NEXT F
70 PRINT"ENTREE DES DONNEES TERMINEE"
```

L'entrée de la valeur muette ou sentinelle qui termine l'entrée des données satisfait la condition à la ligne **40**, qui remet l'élément du tableau à **0**, fixe F à 21 et passe le contrôle à la ligne **60**. Puisque F est plus grand que **20**, la boucle est terminée. Une telle utilisation de l'instruction **GOTO**, dépassant un contrôle dans un programme pour éviter des blocs de lignes de programme est considérée par les gourous puristes de la programmation (!) comme la seule fois où l'utilisation de **GOTO** est pardonnable. Un juste milieu est conseillé à nos lecteurs.

Les boucles **REPEAT** peuvent être arrêtées de la même façon par l'utilisation de drapeaux, originairement fixés à **TRUE** ou **FALSE** avant la boucle, et inversés quand une condition de sortie est remplie. Un **GOTO** peut aussi être nécessaire si le codage doit être évité. Cependant, il y a une autre possibilité offerte au programmeur, — bien que considérée comme douteuse par certains fanatiques de la programmation structurée — celle de l'utilisation de **PULL** (tirer), qui enlève l'adresse du dessus de la pile, et permet à un **GOTO** de sortie de boucle d'être codé sans problème même si les plus libéraux

Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr Page 30 sur 163 15/08/2000

Numérisation http://wsc.n3.net

d'entre nous admettent que son utilisation habituelle n'est pas bonne.

```
10 REM......PULL.................
20 A=9
30 REPEAT
40 B=A
50 REPEAT
60 PRINT B;
70 B=B-1
80 IF B<0 THEN PULL:GOTO 100
90 UNTIL B=0
100 A=A-1
110 PRINT
120 UNTIL A=-5
```

Les tests conditionnels peuvent impliquer plus d'une condition, en association (en utilisant les opérateurs logiques **AND, OR,** et **NOT** et, où et pas). On peut, par exemple, avoir une ligne ainsi composée **IF N<10 A=B THEN...** L'ORIC testera les deux conditions séparément et puis testera si l'association des deux est vraie, selon les tables de vérité pour **AND, OR** et **NOT. AND** fonctionne comme "et" en français courant, c'est à dire que si la condition 1 est vraie et si la condition 2 est vraie, alors l'association des conditions est aussi vraie. Si l'une des conditions est fausse alors l'association est fausse. Le programme suivant produit la table de vérité pour **AND** (et).

```
10 LET L(1)=TRUE
20 LET L(2)=FALSE
30 LET L$(1)="TRUE"
40 LET L$(2)="FALSE"
50 FOR K=1 TO 2
60 :   FOR H=1 TO 2
70 :   :   PRINTL$(K)" AND "L$(H)"=";
80 :   :   IF L(K) AND L(H) THEN PRINT
          L$(1) ELSE PRINT L$(2)
90 :   :   PRINT
100 :  NEXT H
110 NEXT K
```


De nouveau le programme a été disposé afin que la structure soit claire. Puisque l'ORIC utilise des valeurs numériques pour **TRUE** et **FALSE**, on peut les utiliser dans des expressions numériques. La ligne 80 modifiée dans le programme ci-dessus emploie la valeur (−1 ou 0) que l'ORIC dérive de l'expression **AND**, plus 2, pour donner l'écriture de tableau approprié (1 ou 2) pour **L$**.

```
10 LET L(1)=TRUE
20 LET L(2)=FALSE
30 LET L$(1)="TRUE"
40 LET L$(2)="FALSE"
50 FOR K=1 TO 2
60 :   FOR H=1 TO 2
70 :   :   PRINTL$(K)" AND "L$(H)"=";
80 :   :   PRINT L$(2+(L(K) AND L(H)))
90 :   :   PRINT
100 :  NEXT H
110 NEXT K
```

Faites attention à la complexité des parenthèses. La meilleure façon de vous assurer que vous n'en avez pas oublié une est de vérifier qu'il y en a bien le même nombre à gauche et à droite de l'expression. Une expression utilisant **OR** (ou) donne une valeur **TRUE** (vraie) si l'une des conditions qu'elle relie est **TRUE** (vraie) et il en est de même si les deux sont vraies. Le résultat **FALSE** (faux) n'apparaît que si les deux sont fausses. NOT (pas) placé devant une condition inverse les valeurs **TRUE** et **FALSE**.

Des conditions multiples peuvent être associées, en utilisant **OR** et **AND**. Voici un exemple :

```
10 LET L(1)=TRUE
20 LET L(2)=FALSE
30 LET L$(1)="TRUE"
40 LET L$(2)="FALSE"
50 FOR K=1 TO 2
60 FOR H=1 TO 2
70 FOR J=1 TO 2
80 PRINT L$(K)" AND "L$(H)" OR "L$(J)"
DONNE ";
90 PRINT L$(2+(L(K) AND L(H) OR L(J)))
100 PRINT
110 NEXT:NEXT:NEXT
```


William Saint-Cricq / wsaintcr@wanadoo.fr Page 31 sur 163 15/08/2000

Numérisation http://wsc.n3.net

En constraste avec les programmes précédents, remarquez la difficulté accrue de lecture de ce programme, en l'absence d'instructions **LET**, de rentrées, et de noms de variables de boucles. Même ceci est mieux que la sorte de codage souvent rencontrée, qui est écrite de façon aussi dense que possible. Voici le même programme en forme condensée :

```
10 L(1)=TRUE:L(2)=FALSE:L$(1)="TRUE":L
$(2)="FALSE":FOR K=1 TO 2
20 FOR H=1 TO 2:FOR J=1 TO 2:PRINT L$(
K)" AND "L$(H)" OR "L$(J)" DONNE ";
30 PRINT L$(2+(L(K) AND L(H) OR L(J)))
:PRINT:NEXT:NEXT:NEXT
```

Des parenthèses peuvent être employées avec de multiples expressions conditionnelles pour s'assurer que la signification que vous voulez transmettre est comprise par l'ORIC. Essayez de mettre entre parenthèses les deux premières expressions, puis la deuxième avec la troisième, en lançant (**RUN**) le programme à chaque fois pour voir l'interprétation différente donnée ainsi.

L'utilisation d'instructions **GOTO** conditionnelles peut être renforcée en employant une instruction **ON...GOTO**. **ON** est suivie d'une variable ou d'une expression, qui donne une valeur entière, définissant ainsi quels numéros de ligne suivant l'instruction **GOTO** sont rendus actifs :

```
10 INPUT"ENTREZ 1,2 OU 3,SVP";N
20 ON N GOTO 150,200,300
140 REM
150 PRINT"LIGNE 150 POUR N=1
190 REM
200 PRINT"LIGNE 200 POUR N=2"
290 REM
300 PRINT"LIGNE 300 POUR N=3"
```

ON peut aussi être utilisé avec la dernière structure de programme que nous allons présenter, l'écriture **GOSUB... RETURN**. L'instruction **GOSUB** (numéro de ligne) agit comme un **GOTO** en transférant le contrôle du programme au numéro de ligne désigné. Cependant, à la différence de **GOTO**, l'instruction **GOSUB** stocke l'adresse actuelle avant de se déplacer. L'adresse est placée sur la pile et, en rencontrant une instruction **RETURN**, le contrôle du programme est renvoyé (**RETURN**ed) à l'instruction suivant l'appel **GOSUB**.

Le programme ci-dessous illustre les principes :

```
10 REM *     GOSUB      *
20 CLS:REM ...LIMITES
30 SUP=9:INF=1:GOSUB 1000
40 REM...APPEL DU PROGRAMME 'MENU'
50 X=A:GOSUB 2000
60 REM...ETES-VOUS FATIGUE
70 GOSUB 3000
80 IF FA THEN STOP ELSE GOTO 20
90 END
1000 REM...ENTREE DES NOMBRES
1010 PRINT"TAPEZ UN ENTIER ENTRE"INF"E
T"SUP:PRINT
1020 REPEAT
1030 PRINT CHR$(11);:INPUT A$
1040 A=VAL(A$)
1050 UNTIL A>=INF AND A<=SUP AND A=INT
(A)
1060 RETURN
2000 REM..PROGRAMME MENU ET OPTIONS
2010 PRINT:PRINT SPC(7);CHR$(29)"***
   MENU   ***"
2020 PRINT:PRINTCHR$(130);"TAPEZ 1 POU
R FACTORIELLE"X
2030 PRINT:PRINTCHR$(130);"TAPEZ 2 POU
R"X"AU CARRE"
2035 PRINTCHR$(29)
2040 REM...CHOIX A L'AIDE D'UN NOMBRE
2050 SUP=2:INF=1:GOSUB 1000
2060 ON A GOSUB 2100,2200
2070 RETURN
2100 REM...CALCUL DE FACTORIELLE
         PAR ADDITIONS REPETEES
2110 Y=X:S=1:GOSUB 2500
2120 PRINT"FACTORIELLE"X"VAUT"S
2130 RETURN
2200 REM...CALCUL DE CARRE
2210 PRINT"LE CARRE DE"X"EST"X*X
2220 RETURN
2500 REM...SOUS-PROGRAMME RECURRENT
2510 IF Y=0 THEN RETURN
2520 S=S*Y:Y=Y-1
2530 GOSUB 2500
2540 RETURN
```


15/08/2000 Page 32 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

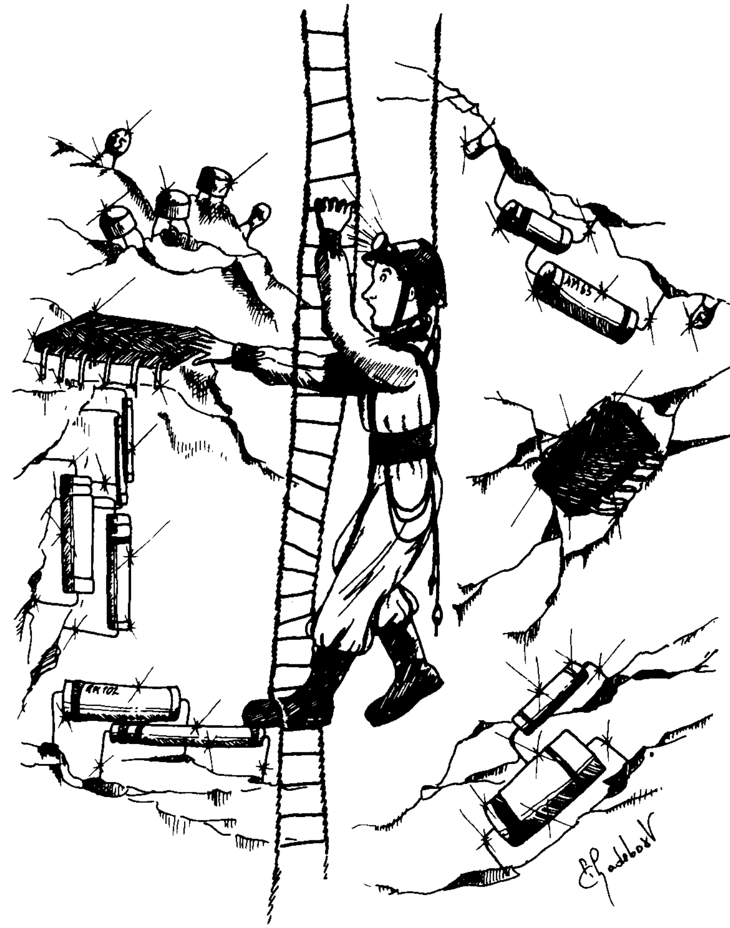
Numérisation http://wsc.n3.net

```
 3000 REM...FATIGUE ?
 3010 PRINT:PRINT"EN AVEZ-VOUS ASSEZ ?
(O/N)"
 3020 IF KEY$="O" THEN FA=TRUE:GOTO 305
0
 3030 IF KEY$="N" THEN FA=FALSE:GOTO 30
50
 3040 GOTO 3020
 3050 RETURN
```

Pendant que nous traitons des sous-programmes, peut-être devrions-nous voir pourquoi les bons programmeurs aiment tant les utiliser. Et bien, d'abord il y a les avantages évidents de gain de place en gardant des parties de programmes à un même endroit au lieu de les répéter à chaque fois. Ceux-ci peuvent aussi faire gagner du temps dans la programmation puisque vous n'avez pas à retaper sans arrêt les mêmes lignes à chaque nouvelle occurrence.

Beaucoup de gens, en fait, sont si passionnés pour ces méthodes qu'ils insistent toujours sur les aspects les plus intellectuels de la programmation structurée, nuisant en cela à la clarté de leurs arguments. Pragmatiquement cependant, cela vaut la peine d'essayer de "structurer" vos pensées, et de considérer la tâche à effectuer sous la forme de sous-tâches plus petites et plus simples. Cette manière de procéder vous permet d'écrire des parties courtes de code, libres de toute confusion et qui se suffisent à elles-mêmes bien "qu'aucun programme ne soit une île", bien sûr.

Quand vous revenez à un programme quelque temps après l'avoir commencé, il peut être très difficile de retrouver la ligne exacte où quelque chose se passe si le travail de programmation a été originairement conçu comme une masse sans forme, interminable et imbriquée. Revenons aux choses pratiques, il n'est pas vraiment important de transformer toutes ces sous-parties de codage en sous-programmes constitués. En fait il est bien préférable de ne pas le faire si elles ne sont utilisées qu'une fois dans tout le programme. En vérité tout ce que l'on peut dire à propos des mérites et autres de la programmation structurée est qu'il faut avoir une idée claire de ce que l'on est en train de faire à chaque stade du programme et garder des tâches non liées sur des lignes différentes afin de pouvoir les résoudre plus tard. Si vous gardez vos parties de programme ou modules bien séparées et clairement signalées par des instructions **REM**, vous deviendrez un programmeur moins frustré.



William Saint-Cricq / wsaintcr@wanadoo.fr Page 33 sur 163 15/08/2000

Numérisation http://wsc.n3.net

Chapitre 5
Suivons le parcours des mémoires

Une des grandes vertus du BASIC est de nous permettre d'énoncer ce que nous désirons que l'ordinateur fasse, et pourvu que l'on mette les bonnes instructions dans un programme, on ne s'inquièter pas de la manière dont l'ordinateur procède. Cependant il y a des instructions en BASIC qui nous permettent d'intervenir directement dans la mémoire de l'ORIC. Il nous faut d'abord savoir ce qu'il y a dans la mémoire et comment elle est organisée avant de pouvoir nous en servir.

Il faut d'abord avoir quelques notions du système de numération binaire. Celui-ci utilise des suites de **Ø** et de **1** pour représenter des nombres dans la mémoire de l'ordinateur. Un emplacement mémoire peut être considéré comme un ensemble ordonné de huit cases, chaque case pouvant être soit libre représentée par un **Ø** (état non actif) soit occupée, représentée par **1** (état actif). Chaque case contient ainsi une suite de 8 chiffres binaires ou bits, et les huit ensemble forment un octet. A l'intérieur d'un octet les bits sont numérotés de Ø à 7 de droite à gauche, et chaque bit, quand il est actif, représente une valeur de 2 fois celle du bit à sa droite. Le bit Ø peut être ou Ø ou 1, et représente ces valeurs, tandis que le bit 1 représente 2 s'il est actif et Ø s'il est non actif. La valeur que prend la suite de 8 bits est la somme des nombres représentés par les bits actifs. La plus grande valeur qui peut être contenue dans une suite de huit bits est 255 et la plus petite est bien sûr Ø :

Bit	7	6	5	4	3	2	1	Ø	
Binaire	1	1	1	1	1	1	1	1	
Valeur	128	64	32	16	8	4	2	1	= 255 (décimal)

Le nombre binaire Ø11Ø11Ø1, par exemple, serait :

Bit	7	6	5	4	3	2	1	Ø	
Binaire	Ø	1	1	Ø	1	1	Ø	1	
Valeur	Ø	+64	+32	+Ø	+8	+4	+Ø	+1	= 109 (décimal)

15/08/2000 Page 34 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

Le système binaire procède par puissance de deux (non pas par puissances de 1Ø comme notre système décimal habituel). Ceci peut être illustré par le programme suivant :

```
10 REM ....PUISSANCES DE 2.....
20 FOR PUIS=0 TO 7
30 PRINT"2 A LA PUISSANCE"PUIS"="2^PUI
S
40 NEXT PUIS
```

Voici un programme qui convertit le système binaire en décimal et inversement :

```
10 REM...CONVERSIONS BINAIRE/DECIMAL
15 '      ........................
16 '      ...PROGRAMME PRINCIPAL.....
17 '      ........................
18 CLS
20 PRINT"APPUYEZ SUR B POUR BINAIRE->D
ECIMAL"
30 PRINT"APPUYEZ SUR D POUR DECIMAL->B
INAIRE"
40 GET M$:IF M$<>"D" AND M$<>"B" THEN
40
50 IF M$="D" THEN GOSUB 500 ELSE GOSUB
1000
60 PRINT"UN AUTRE NOMBRE ? (O/N)"
70 GET M$:IF M$="O" THEN CLS:GOTO 20 E
LSE END
80 '..............................
85 '......FIN DU PG PRINCIPAL.......
90 '..............................
490 '
500 '..............................
501 '.....SOUS PROGRAMME CONVERSION .
502 '.....DECIMAL EN BINAIRE.........
503 '..............................
504 '
510 CLS:INPUT"DONNEZ UN ENTIER EN DECI
MAL";N%
520 PRINT:PRINT N%;:B$=""
530 REPEAT
540 :   I=INT(N%/2)
550 :   BIT=N%-2*I
560 :   IF BIT=0 THEN B$="0"+B$ ELSE
      B$="1"+B$
```



```
570 :   N%=I
580 UNTIL N%=0
590 PRINT"S'ECRIT "B$" EN BINAIRE
595 PRINT
600 RETURN
610 '
620 '###  FIN DU SOUS PROGRAMME ###
630 '
1000 '..............................
1001 'SOUS-PROGRAMME DE CONVERSION
1002 'DE BINAIRE EN DECIMAL
1003 '..............................
1004 '
1010 CLS:INPUT"DONNEZ UNE SEQUENCE BIN
AIRE ";B$
1020 N=0:P=0
1030 FOR J=LEN(B$) TO 1 STEP -1
1040 :    BIT$=MID$(B$,LEN(B$)-P,1)
1050 :    N=N+VAL(BIT$)*2^P
1060 :    P=P+1
1070 NEXT J
1080 PRINT B$" EST LE DECIMAL"N
1085 PRINT
1090 RETURN
1100 '
1110 '#### FIN DU SOUS PROGRAMME ###
```

Remarquez que le programme est structuré avec un module de programme principal, d'où le sous-programme approprié est appelé selon l'entrée de l'utilisateur une fois le menu présenté. Le programme montre aussi comment un listing peut être rendu plus lisible en incluant des instructions **REM** pour séparer le programme en modules et en ménageant des rentrées à l'intérieur des boucles.

Le programme interne décimal/binaire utilise une boucle **REPEAT...UNTIL** pour réduire le nombre d'une puissance de deux à chaque passage dans la boucle, en affectant un **Ø** ou un **1** à la chaîne binaire **B$** selon qu'il existe un reste ou non après la division par 2. La boucle se termine quand **N% = Ø** et il n'y a plus de nombre à manipuler. Le programme interne binaire/décimal utilise une boucle **FOR...NEXT** pour tester chaque bit de la chaîne binaire en partant de la droite, en augmentant de 2 la puissance (P) par laquelle la valeur du bit est multipliée à chaque passage par la boucle.

L'organisation de la mémoire de l'ORIC est montrée dans un diagramme à l'Annexe 5. Les versions 48K et 16K ont la même organi-

15/08/2000 Page 35 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

sation de mémoire, sauf un tronçon manquant au milieu de la mémoire du 16K entre #4000 et #C000 hex (16384 et 49152 en base 10). En vous rapportant à ce diagramme, et en utilisant votre ORIC pour convertir les héxadécimaux (**PRINT** #4000, etc.) vous verrez que les premières cases 16384, au-dessus #C000, sont des Read Only Memory (Mémoire que l'on ne peut lire : mémoire morte) (ROM) sur les machines 16K et 48K. C'est là que les programmes internes BASIC d'interprétation et d'arithmétique sont contenus en mémoire permanente fixe stockés dans les puces (chips) ROM de l'ORIC.

La mémoire ROM ne peut pas être transformée (bien que, comme son nom l'indique, elle puisse être lue afin que nous puissions découvrir ce qu'elle contient). Cette zone de mémoire contient des instructions et données en langage machine, et le processus de compréhension des contenus de la mémoire en lecture (appelé désassemblage, par opposition à l'assemblage que vous pourrez décrouvrir au chapitre 10) est complexe.

Le reste de la mémoire de l'ORIC consiste en Random Access Memory (mémoire vive) (RAM) qui permet à la fois la lecture et l'introduction de données, afin que l'on puisse insérer dans n'importe quelle case un nombre entre 0 et 255 et aussi découvrir la valeur que prend n'importe quelle case. Le nombre stocké dans une case peut représenter une partie d'un nombre, un mot clé du BASIC, un code de caractère ou une partie d'une adresse. Si vous programmez en langage machine, cela peut être aussi une instruction en langage machine. Le traducteur stocké en ROM (mémoire morte) décide ce qu'un nombre précis représente selon le contexte et la place en mémoire.

En commençant par le bas de la RAM, nous avons cinq pages de mémoire, chacune des 256 cases étant réservée à un but spécifique (précis). La première page (page zéro) contient des renseignements sur les affaires courantes à l'intérieur de l'ORIC qui sont nécessaires à la puce 6502 (CPU : Central Processing Unit = Unité Centrale de Traitement), tels que les adresses de début et de fin de programme BASIC, des pointeurs de stockage de variable chaîne etc.

La page 1 est une pile pour l'utilisation des programmes internes d'arithmétique, le stockage des nombres et des valeurs intermédiaires impliqués dans les calculs courants.

La page 2 stocke le temps de passage (run-time) ou des variables de système qui contiennent des valeurs nécessaires pour suivre l'action du BASIC telles que les positions de curseur, **CAPS** (majuscules/minuscules), l'enfoncement ou non des touches.

La page 3 est traitée au chapitre 11 car elle contient l'adresse pour Input (entrée)/Output (sortie) entre l'ORIC et le monde extérieur et les adresses pour le transfert de données entre les différentes puces spécialisées de l'ORIC (voir le diagramme à l'Annexe 11).

Les adresses de la page 4 entre #0400 et #0420 sont disponibles pour les programmes de l'utilisateur en langage machine, et le reste de la page est réservé à l'utilisation de systèmes.

Les adresses de #0500 à plus sont les cases mémoire pour le stockage de vos programmes en BASIC et des valeurs variables. Nous verrons comment cette partie de la mémoire est organisée ci-dessous. Le programme en BASIC grandit en hauteur en s'allongeant, en prenant avec lui la zone des variables qui est au-dessus.

Les cases au-dessus #9800 (#1800 pour l'ORIC 16K) stockent les ensembles de caractères et l'affichage écran. Le mode **TEXT** (texte) (qui est celui qui apparaît quand on met en marche et c'est celui que nous avons utilisé) prend beaucoup moins de place que le **HIRES** (haute résolution). Les différerents modes d'affichage seront traités au chapitre 7, où nous abordons toute la question du mode graphique et des affichages écran. Cependant, il vaut mieux dire ici que, pour permettre de passer d'un mode à l'autre, toute la zone au-dessus de #9800 (#1800 pour le 16K) est réservée. Si l'on n'a besoin que d'une sortie en mode **TEXT** sur l'écran au cours d'un programme, la commande **GRAB** peut être utilisée pour libérer la zone de #9800 à #B400 (#1800 à #3400 sur le 16K) pour l'emploi du BASIC. Ceci est inversé par la commande **RELEASE** quand le mode **HIRES** (haute résolution) est à nouveau nécessaire. Les ensembles de caractères standards et alternés sont stockés dans deux suites de cases mémoire, une de 1024 octets, stockant les 128 caractères de l'ensemble des caractères standards, et une de 896 octets, contenant les 112 caractères de l'ensemble alterné (zone utilisé en bascule). Chaque caractère est stocké en une suite de 8 octets, qui donne la combinaison des bits définissant la disposition des points affichés sur l'écran. Le chapitre 7 explique comment cela fonctionne. Tous les caractères sont stockés, bien que les caractères non affichants aient un nombre nul stocké dans leurs octets de définition de caractère, simplement pour s'assurer qu'ils ne s'affichent pas par **PRINT**.

Les caractères stockés dans cette zone peuvent être redéfinis par l'utilisateur pour former n'importe quels caractères spéciaux nécessaires à un programme. Ceci est accompli en transmettant des valeurs différentes dans les adresses en mémoire, ce qui forme un caractère spécifique. La mémoire d'écran est aussi traitée au chapitre 7. Au-dessus du stockage de la mémoire d'écran existe une zone de mémoire libre disponible pour la langage machine ou l'utilisation Input/Output.

L'instruction en BASIC pour lire le contenu d'un octet est **PEEK**, l'écriture est **PEEK** (adr) où adr est un nombre héxadécimal ou décimal indiquant l'adresse. **PEEK** donne la valeur stockée dans l'octet en nombre décimal. **PRINT PEEK** (1280) par exemple, affi-

Numérisation http://wsc.n3.net
William Saint-Cricq / wsaintcr@wanadoo.fr Page 36 sur 163 15/08/2000

chera la valeur contenue dans l'octet à la case mémoire 128Ø. Le même résultat est obtenu en tapant **PRINT PEEK (#5ØØ)**.

POKE transmet des données à un octet de la mémoire. **POKE** adr,i met la valeur de i (de Ø à 255) dans l'octet à la case mémoire désignée par adr. **POKE 16ØØ, 134** met la valeur 134 à la case 16ØØ. Les valeurs peuvent toutes deux être désignées en notation hexadécimal : **POKE #64Ø, #56**.

Les valeurs entières jusqu'à 65535 peuvent être stockées dans deux octets, et l'ORIC utilise ce système pour des cases adresses qu'il faut stocker en mémoire. Les deux octets contiennent la valeur sous la forme (valeur du premier octet) plus (256* valeur du second octet). Il y a des instructions en BASIC pour lire et écrire de telles valeurs directement, sans calcul. Ce sont **DEEK** (adr) qui donne la valeur stockée en adr et adr + 1, et **DOKE** adr,i qui place la valeur entière i (de Ø à 65535) dans l'octet désigné par adr. et l'octet suivant.

En utilisant ces instructions, nous pouvons observer comment un programme en BASIC est stocké dans la mémoire de l'ORIC. Le programme ci-dessous affichera l'adresse mémoire, la valeur **PEEKed** (regardée) contenue dans l'octet désigné par cette adresse, et le caractère correspondant à cette valeur si le caractère est affichable. La ligne **4Ø** découpe simplement les données en pages qui tiendront sur l'écran. La ligne **6Ø** ajuste à droite le nombre contenu dans l'octet en utilisant **SPC**.

```
 10 REM... STOCKAGE DU PROGRAMME
 20 LET MEM=#500
 30 CLS:FOR K=1 TO 200
 40 IF INT(K/25)=K/25 THEN PRINT"APPUYE
Z SUR UNE TOUCHE POUR LA SUITE":GETA$
 50 N=K+MEM:M=PEEK(N):L=LEN(STR$(M))
 60 PRINTN;SPC(10-L);M;
 70 IF M<129 AND M>32 THEN PRINT SPC(4)
;CHR$(M) ELSE PRINT" "
 80 NEXT K
```

L'affichage pour les trois premiers écrans produits se fait comme suit :

Adresse	Valeur	Car.	Adresse	Valeur	Car.
1281	32		1288	46	.
1282	5		1289	32	
1283	10		1290	83	S
1284	0		1291	84	T
1285	157		1292	79	O
1286	46	.	1293	67	C
1287	46	.	1294	75	K

Adresse	Valeur	Car.	Adresse	Valeur	Car.
1295	65	A	1332	58	:
1296	71	G	1333	141	
1297	69	E	1334	32	
1298	32		1335	75	K
1299	68	D	1336	212	
1300	85	U	1337	49	1
1301	32		1338	32	
1302	80	P	1339	195	
1303	82	R	1340	32	
1304	79	O	1341	49	1
1305	71	G	1342	48	0
1306	82	R	1343	48	0
1307	65	A	1344	0	
1308	77	M	1345	130	
1309	77	M	1346	5	
1310	69	E	1347	40	(
1311	0		1348	0	
1312	47	/	1349	153	
1313	5		1350	32	
1314	20		1351	215	
1315	0		1352	40	(
1316	150		1353	75	K
1317	32		1354	207	
1318	77	M	1355	50	2
1319	69	E	1356	53	5
1320	77	M	1357	41	)
1321	212		1358	212	
1322	35	#	1359	75	K
1323	53	5	1360	207	
1324	48	0	1361	50	2
1325	48	0	1362	53	5
1326	0		1363	32	
1327	65	A	1364	201	
1328	5		1365	32	
1329	30		1366	186	
1330	0		1367	34	"
1331	148		1368	65	A

Il se peut que vous vous y perdiez un peu, mais tout sera clarifié ! La première adresse **PEEKed** (regardée) est **#5Ø1** (1281 décimal), suivant le zéro stocké à l'adresse 128Ø, marquant le commencement d'une zone de programme en BASIC. La valeur stockée dans les cases 1281 et 1282 est un indicateur pour l'adresse mémoire contenant le début de la ligne de programme suivante. Si vous entrez la commande **PRINT DEEK (1281)** en commande directe avec ce programme en mémoire, vous obtiendrez 1312 sur l'écran. En regar-

Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr Page 37 sur 163 15/08/2000

dant cette case, vous verrez qu'elle suit un zéro à l'adresse 1311 qui est utilisé par l'ORIC pour séparer des lignes de programme. Les octets 1312 et 1313 sont les deux premiers octets de la ligne 2Ø, formant l'indicateur pour la ligne 3Ø. Toutes les lignes de programme sont reliées par ces indicateurs, et l'ORIC peut avancer "en gambadant" dans les lignes de programme pour trouver la ligne désignée par une instruction **GOTO** ou **GOSUB** avec le maximum d'efficacité.

Les adresses 1283 et 1284 contiennent une valeur de deux octets dans la même écriture (octet 1 = 256* octet 2) ce qui donne le numéro de ligne, dans ce cas n° 1Ø. La case 1285 contient la valeur 157, qui est la forme codée de **REM**. L'ORIC stocke chaque mot clé du BASIC qu'il reconnaît quand la ligne de programme est placée en mémoire sous une forme qui occupe seulement un seul octet de mémoire. L'annexe 12 donne une liste des mots clés du BASIC et de leurs codes.

Après le code correspondant à l'instruction **REM**, on trouve trois codes 46 correspondant aux trois "." suivi du code 32 qui correspond à un espace puis les codes ASCII du titre : STOCKAGE DU PROGRAMME.

Le zéro termine la ligne. Les adresses 1312 et 1313 stockent l'indicateur à la ligne suivante et 1314/5 le numéro de ligne. Bien que le caractère soit/affiché, la valeur stockée dans l'octet est une valeur numérique, pas un code de caractère, ce qui est clair (pour l'ORIC) d'après le contexte. Un coup d'œil à l'Annexe 12 vous dira que la valeur de 15Ø stockée à la case 1316 est le symbole pour **LET**. Celui-ci est suivi par la variable **MEM** dans les trois cases suivantes. L'adresse 1321 contient 212, le symbole pour l'opérateur d'égalité, c'est-à-dire le signe "=" quand il est stocké comme une fonction arithmétique. L'ORIC transforme le caractère en opérateur pour l'insertion en mémoire, puis le convertit à nouveau en caractère quand un programme est listé, comme il le fait avec les mots clés du BASIC. Les symboles pour les opérateurs arithmétiques sont :

=	212	Égalité
+	2Ø4	Addition
−	2Ø5	Soustraction
*	2Ø6	Multiplication
/	2Ø7	Division
↑	2Ø8	Exponentiation

Les adresses 1313 à 1316 stockent **#5ØØ**, suivies par le zéro signifiant la fin de ligne.

Nous vous laisserons poursuivre seul le reste du listing. Si vous voulez découvrir le mot clé correspondant à une valeur particulière, vous pouvez utiliser **POKE** (piquer) :

```
10 REM
20 INPUT"DONNEZ LA VALEUR DU SYMBOLE";
N%
30 POKE 1285,N%
40 LIST
```

L'adresse 1285, qui est occupée par le symbole pour **REM** est introduite (**POKEd**) avec la valeur d'entrée (**INPUT**). Quand la ligne **4Ø** listera le programme, l'ORIC convertira la valeur introduite (**POKEd**) en forme affichée (**PRINTed**). Une illustration supplémentaire de la valeur de **POKE** est la simplicité avec laquelle on peut changer les instructions **PRINT** des lignes **4Ø**, **6Ø** et **7Ø** du programme d'affichage mémoire en instructions **LPRINT**. C'est ainsi que ci-dessus le listing du contenu de la mémoire a été produit. Les lignes supplémentaires **9Ø-14Ø** ajoutées au programme, comme vous le voyez ci-dessous, testeront les cases mémoires en train de stocker le programme, remplaçant chaque occurrence d'un symbole **PRINT** (valeur 186) par un symbole **LPRINT** (143). La boucle **REPEAT... UNTIL** se termine quand les cases qui suivent sont toutes les deux **Ø**, ce qui est le marqueur de la fin du programme en BASIC, en utilisant **DEEK**

```
90 REM... change PRINT en LPRINT
100 M=#500:C=0
110 REPEAT
120 M=M+1
130 IF PEEK(M)=186 THEN POKE M,143
140 UNTIL DEEK(M+1)=0
```

Comme autre exemple de **DEEK** et **DOKE**, voici un simple programme de rénumérotation, qui renumérotera des lignes à des pas donnés, commençant à une ligne désignée. Il ne renumérote pas les lignes de destination de **GOTO** et **GOSUB**. Ceci n'est pas difficile à faire mais demande plus d'un passage par la zone mémoire.

```
10 REM...... RENUMEROTATION.......
11 'debut du pg ligne 60000
12 'ne modifie pas les adresses de
13 'destination de GOTO ou GOSUB
14 'Notez-les avant renumerotation
60000 MEM=#501  'Pointeur en l.1
60010 INPUT"1ere ligne a renumeroter";
DEBUT
```


Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr Page 38 sur 163 15/08/2000

```
60020 INPUT"Quelle ligne devient-elle
";NLIN
60030 INPUT"Avec quel pas ";INC
60040 REPEAT
60050 LIGNE=DEEK(MEM+2)'l. actuelle
60060 IF LIGNE<DEBUT THEN 60080
60070 DOKE (MEM+2),NLIN:NLIN=NLIN+INC
      introduit nouveau no et pas
60080 MEM=DEEK(MEM)'fixe nlle ligne
60090 UNTIL DEEK(MEM+2)=60000 'ne se
     renumerote pas lui-meme
```

Au-dessus du programme en BASIC sont stockées les variables utilisées à l'intérieur du programme et leurs valeurs. C'est la zone qui est effacée quand **CLEAR** est utilisé. Les noms de variables sont stockés dans deux octets (les deux caractères d'un nom de variable que l'ORIC reconnaît) avec le type de variable identifiée par la modification du code ASCII des caractères du nom. Les variables numériques réelles sont stockées sans changement de code, les variables entières ont 128 ajouté à chaque code de caractère, et les variables chaîne ont 128 ajouté au deuxième caractère seulement. Les noms d'une seule lettre ont Ø (+ 128 si chaîne ou entiers) stocké dans le deuxième octet. L'ordre dans lequel les variables simples variables sont utilisées dans un programme est déterminé par la suite dans laquelle elles sont affectées dans le programme, et les tableaux sont stockés après les simples variables dans l'ordre dans lequel ils sont **DIM**ensionés. Après le nom des variables numériques il y a cinq octets qui, dans le cas de variables numériques réelles, contiennent le nombre dans une écriture où le premier octet contient un exposant et les quatre restant la mantisse. Cette écriture est connue sous le nom d'écriture à virgule flottante à cinq octets, qui fonctionne en binaire comme la notation exponentielle en décimal (base 1Ø). Les variables entières sont stockées dans les premiers octets des cinq, en forme standard de deux octets. Les autres octets contiennent Ø. Les variables chaînes sont stockées avec cinq octets suivant le nom de la variable. Le premier octet contient la longueur de la chaîne, et le second et le troisième sont un indicateur pour l'adresse en mémoire où la chaîne de caractères est contenue. Ceci sera une adresse à l'intérieur du listing du programme en BASIC si la variable chaîne contient une chaîne littérale définie dans le programme. Si c'est une chaîne de calculs, ou si elle est redéfinie au cours du programme, la chaîne qui en résulte est stockée sur le dessus de la zone des variables immédiatement en dessous des zones de caractères fixes, et l'indicateur désignera cette case. La valeur de longueur stockée dans le premier octet après le nom de la chaîne

indique combien d'octets doivent être lus, en commençant à l'adresse donnée par les octets indicateurs. Deux octets inutilisés suivent l'indicateur.

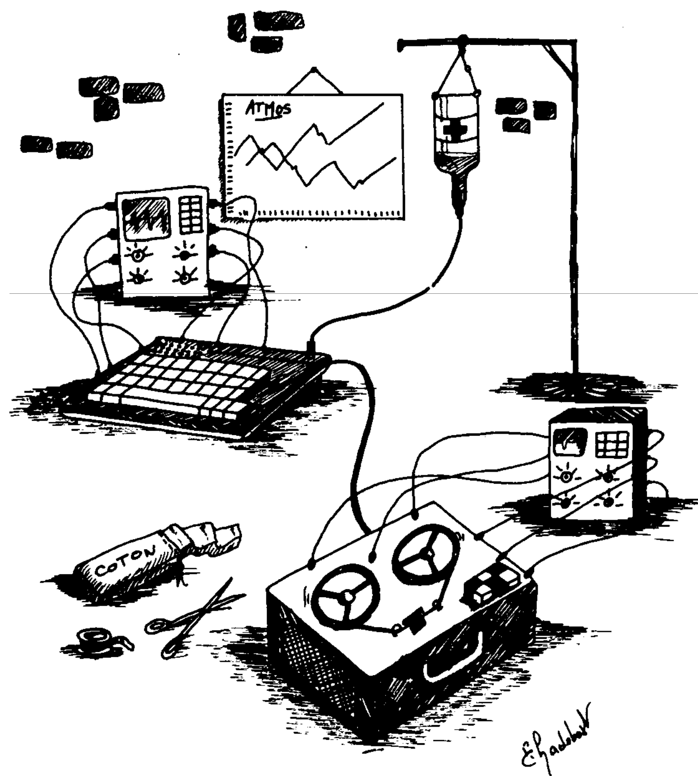
Les tableaux ont le même type de noms que les variables simples, mais ceux-ci sont suivis d'un octet indiquant le nombre total d'octets nécessaires pour stocker les données du tableau (ainsi que le nom du tableau). Deux octets contiennent ensuite le nombre de dimensions dans le tableau, suivis de deux octets pour chacune des dimensions, désignant le nombre d'éléments de chaque dimension. Les tableaux numériques ont cinq octets pour chaque élément, stockés à la suite, alors que les tableaux d'entiers n'ont que deux octets dans chaque élément. Chaque élément chaîne dans un tableau a un octet de longueur, et un indicateur de deux octets. Vous pouvez accéder au début de la zone des variables et afficher le contenu pour voir comment les données sont stockées avec un programme semblable à celui que nous avons utilisé pour explorer le stockage du programme en BASIC. Fixez quelques variables, trouvez le début de la zone des variables avec **DEEK (#9C)** et vous pouvez, commencer à explorer **(PEEK)**. Les deux octets aux cases #9C et #9D sont un système de variable contenant l'adresse du début du stockage des variables. De la même manière, **DEEK (#9E)** fournira l'adresse de fin de zone des variables et **DEEK (#A2)** vous donnera l'adresse du bout de la zone de variables chaîne.

Ceci conclut notre voyage le long des chemins de traverse de la mémoire de l'ORIC. Si vous avez apprécié l'excursion, vous pouvez utiliser POKE bien au delà avec l'aide de l'annexe 9. Notre prochain chapitre concerne l'enregistrement du contenu des mémoires sur bande pour un stockage sûr, vous permettant ainsi de ne pas toujours laisser votre ORIC branché.

Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr Page 39 sur 163 15/08/2000

Numérisation http://wsc.n3.net



Chapitre 6
Enregistrements et Données

Aussitôt que vous débranchez votre ORIC, tout programme mis en mémoire est perdu. C'est parce que la mémoire vive (RAM) qui stocke le programme en cours et les variables est "volatile", c'est à dire que quand le courant est coupé, la mémoire vive (RAM) et les registres de l'unité centrale sont effacés, prêts pour un nouveau départ lorsque vous rebranchez votre ORIC.

Il n'est évidemment pas pratique de taper un programme à chaque fois que vous voulez l'utiliser (un programme de 5Ø lignes vous prendra presque une heure) et donc quelques méthodes de stockage "hors lignes" sont clairement nécessaires. Le moyen le moins cher et le plus utilisé de stockage de programme est la cassette. De nos jours, bien que le stockage sur cassette n'est, ni le plus rapide, ni le plus fiable moyen pour préserver vos programmes, il est considérablement moins cher que les systèmes basés sur des disquettes, et votre ORIC ATMOS a de nombreuses caractéristiques qui rendent la manipulation de cassettes plus fiable et plus simple que d'ordinaire.

Pour sauver des programmes de l'ORIC sur cassette, il vous faudra à la fois un magnétophone à cassettes et un cordon approprié pour le connecter à votre ordinateur. Le choix de magnétophones est large, mais ceci n'est pas pour vous suggérer de prendre le plus cher. En fait, il est préférable d'utiliser un magnétophone bon marché en monophonie (si vous souhaitez utiliser votre appareil stéréo, assurez-vous que vous n'utilisez qu'un seul canal) et les enregistreurs de données spécialisés disponibles dans le commerce au prix de 45Ø F environ sont idéaux pour cette tâche. Vous vous rendrez la vie plus facile si vous choisissez un appareil avec un compteur, parce-que vous pouvez perdre beaucoup de temps à rechercher des programmes si vous n'avez aucun moyen d'établir exactement où ils sont sur une bande.

Essayez de toujours garder le même magnétophone si possible car vous pouvez rencontrer des problèmes quand vous essaierez de charger des programmes sauvés avec un autre appareil (de légères différences dans l'alignement des têtes de lecture peuvent rendre inutilisable une cassette bien enregistrée quand elle est passée sur un autre appareil).

Le type de cordon qu'il vous faut dépend de la sorte de magnétophone que vous déciderez d'utiliser. L'ORIC lui-même demande une prise mâle DIN 3 ou 7 fiches qui entrera dans la prise magnétophone à l'arrière de l'appareil. Un cordon DIN de 3 à 3 fiches est

William Saint-Cricq / wsaintcr@wanadoo.fr Page 40 sur 163 15/08/2000

fourni. La majorité des enregistreurs de données utilise une simple prise femelle DIN qui sert à la fois à l'enregistrement et à la lecture, et la prise dont vous aurez besoin dans ce cas est exactement la même que celle de l'ORIC à l'autre bout du branchement. Cependant certains magnétophones à cassettes mono n'ont que des prises EAR et MIC (écoute et micro) ce qui complique légèrement les choses. Dans ce cas vous aurez besoin de prises jack séparées pour l'entrée et la sortie (c'est-à-dire pour la lecture et l'enregistrement) et il est sage de faire une marque bien visible sur chaque prise afin de ne pas les confondre. Les divers types de cordon sont en général disponibles dans le commerce mais si vous rencontrez des difficultés votre détaillant en matériel hi-fi acceptera d'en faire un pour vous.

Il est bien préférable d'utiliser des cassettes pour ordinateur de courte durée. (C10 ou C15) provenant d'un bon fabricant que des cassettes C90 ou C60. En dehors du fait qu'il faut du temps pour repérer un programme sur une cassette C90 (avec ou sans compteur), les bandes plus longues (et plus minces) tendent à s'étirer beâucoup plus facilement que les cassettes de courte durée, et votre enregistrement peut plus facilement être endommagé.

L'ORIC a deux commandes principales liées à la manipulation de la cassette ; **CSAVE** et **CLOAD** (sauver et charger). Regardons d'abord les écritures et les facilités offertes par l'utilisation de ces commandes, avant de parler des autres possibilités de fichier sur bande de votre ATMOS.

Une fois que vous avez chargé votre cassette neuve dans votre magnétophone, il vaut mieux la faire défiler jusqu'au bout puis la rembobiner pour être sûr que la cassette est enroulée bien serrée et bien également. La plupart des cassettes débutent par un court morceau de plastique sur lequel vous ne pouvez pas enregistrer. Avant de commencer à enregistrer votre programme assurez-vous que vous avez avancé la bande au-delà de ce morceau.

En fait, cela vaut la peine de laisser défiler la bande environ 15 secondes, car la majorité des enregistrements abîmés résultent de dégâts causés au début de la cassette ou d'un étirement dans les premiers centimètres de bande. Mettez le compteur de votre magnétophone à zéro et vous êtes prêts à y aller.

Pour sauver un programme sur cassette, suivez l'ordre ci-dessous :

1. Vérifiez que le magnétophone à cassettes est branché

2. Assurez-vous qu'il est bien relié à votre ORIC. Le bout de la liaison côté ORIC doit être branché dans la prise à orifice DIN, et l'autre bout doit être branché à la prise entrée sortie du magnétophone si vous utilisez un magnétophone avec une prise DIN ou la prise mâle jack MIC doit être dans la prise femelle MIC si vous utilisez un cordon à prises jacks (dans ce cas ci, la prise EAR (écoute) ne doit pas être connectée pour éviter de possibles boucles de retour de signal).

3. Vérifiez que vous avez une cassette vierge dans le magnétophone et que vous avez avancé la bande au-delà du morceau de plastique (bande amorce).

4. Listez **(LIST)** le programme que vous voulez sauver **(CSAVE)** sur l'écran.

5. Tapez en commande directe :

CSAVE "NOM DE FICHIER", S

où **"NOM DE FICHIER"** représente le nom du programme que vous souhaitez sauver. Le nom peut comporter jusqu'à 16 caractères mais il est préférable de donner un nom aussi court, aussi pertinent et aussi facile à se rappeler que possible. N'importe quels caractères peuvent être employés. Vous remarquerez que dans l'écriture ci-dessus le "nom du fichier" entre guillemets est suivi d'une virgule et d'un S. Le S signifie Slow (lent) et c'est une partie facultative de l'écriture. L'ORIC sauve (ou sauvegarde) **(CSAVE)** et charge **(CLOAD)** à deux modes de transmission des données. Sans le ,S l'ORIC applique automatiquement le mode Fast (rapide) à 2400 baud et si vous ajoutez ,S le mode Slow (lent) est mis en place (300 baud). Un baud est une mesure de transmission des bits de données par seconde. Bien que les deux écritures soient fiables, vous devez toujours prendre la précaution de faire au moins une copie de tout programme important en mode lent (Slow) pour une sécurité parfaite. Continuons.

6. Appuyez les touches **RECORD** et **PLAY** (enregistrement et lecture) sur votre magnétophone.

7. Appuyez sur la touche **RETURN** de l'ORIC, le message **SAVING NOM DE FICHIER** apparaîtra en haut de l'écran (sauvegardant nom du programme), suivi par l'indicateur du type de fichier (B signifiant programmé en BASIC). Quand le programme aura été sauvé, le message **READY** habituel apparaîtra à la position courante du curseur. Arrêtez le magnétophone.

Si vous avez respecté scrupuleusement les étapes ci-dessus, vous devez maintenant avoir une copie de votre programme sur cassette. Afin d'être certain que le programme a été sauvé correctement sans avoir à le recharger dans l'ORIC, vous avez la possibilité de vérifier **(VERIFY)** l'enregistrement pour vous assurer que le programme a été correctement sauvé, mais nous y reviendrons. Pour l'instant, parlons du chargement.

Quand vous voulez charger un programme à partir de la cassette, les étapes suivantes doivent être respectées :

1. Vérifiez que le magnétophone est branché, que le volume est réglé en position moyenne, et que les aigus sont en position maximum.

Numérisation http://wsc.n3.net
15/08/2000 Page 41 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

2. Assurez-vous que le magnétophone est correctement relié à l'ORIC. Le branchement de la prise DIN de l'ordinateur sera la même que lorsque vous avez sauvé le programme. Si vous utilisez le cordon fourni ou un cordon similaire DIN/DIN, il sera branché dans la prise d'entrée/sortie du magnétophone. Si vous utilisez un cordon prise DIN/prise jack, la fiche jack de EAR doit être branchée à la prise EAR et la fiche jack MIC déconnectée.

3. En utilisant votre compteur, repérez le début du programme que vous souhaitez charger.

4. Tapez :

CLOAD "NOM DE FICHIER", S

Le nom du programme doit être correctement libellé et doit comporter les mêmes espaces que ceux ménagés au moment de la sauvegarde **(CSAVE)** ou l'ORIC ne reconnaîtra pas le programme comme étant celui que vous essayez de charger. Si vous avez oublié ou n'êtes pas sûr du nom précis, vous pouvez utiliser l'écriture :

CLOAD "", S

et l'ORIC chargera le premier programme qu'il trouve sur la cassette. Une fois de plus le ,S a été inclus dans l'écriture de l'instruction mais ne doit être utilisé que si le programme a été sauvé en mode Slow (lent).

5. Appuyez sur la touche **RETURN** de votre ORIC.

6. Appuyez sur le bouton **PLAY** de votre magnétophone. L'ORIC laissera défiler la cassette jusqu'à ce qu'il trouve le programme "nom de fichier" et pendant ce temps le message **SEARCHING**... (en recherche) figure en haut de l'écran. Quand le programme aura été trouvé le message se transformera en **LOADING NOM DE FICHIER** (suivi par l'indicateur de type de fichier B) jusqu'à ce que le processus de chargement soit fini, et le message **READY** apparaîtra. S'il y a un défaut sur la cassette vous obtiendrez un message **ERRORS FOUND** sur l'écran à la fin du chargement. Si cela arrive, ne désespérez pas tout de suite. Cela peut être parce que le niveau du volume de votre magnétophone est réglé trop haut ou trop bas, ou que la tonalité n'est pas au bon niveau. Avant d'abandonner, passez quelque temps à faire de petits réglages sur votre magnétophone. Si vous obtenez le chargement du programme à la suite de ces réglages, il sera sage de faire une autre copie du programme tout de suite, par mesure de sécurité.

Vous connaissez maintenant tout ce qu'il vous faut sur l'utilisation directe des commandes **CSAVE** et **CLOAD** sur l'ORIC. Regardons maintenant les autres possibilités d'utilisation de la cassette. Si vous voulez qu'un programme tourne automatiquement dès qu'il

est chargé, vous devez ajouter le paramètre suivant à votre commande **CSAVE** :

CSAVE "NOM DE FICHIER", AUTO ,S

Encore une fois, le **,S** ne doit être inclus que si vous souhaitez sauvegarder le programme en mode lent (Slow). Sauvé selon l'écriture ci-dessus, le programme tournera aussitôt que le processus de chargement sera terminé. Les programmes sauvés en **AUTO** ne nécessitent pas de changement dans l'écriture de l'instruction **CLOAD**. Il est non seulement possible de stocker des programmes sur cassette mais aussi des blocs de mémoire. Ceci est particulièrement utile si vous voulez sauver ce qui figure sur l'écran. Cependant, si vous utilisez cette méthode pour sauver l'affichage écran, vous devez vous assurer que l'ORIC est dans le mode qui convient pour l'affichage en question. Pour stocker n'importe quel bloc de mémoire il est nécessaire de connaître l'Adresse où commence le bloc et où il se termine (End). Avec ces renseignements disponibles, les blocs mémoires peuvent être sauvegardés avec l'écriture suivante :

CSAVE "MEMBLOC", A # 4ØØ, E # 42Ø ,S

Ceci sauvegarderait (en mode lent) le contenu de RAM (mémoire vive) des cases # 4ØØ à # 42Ø. Vous pouvez employer cette possibilité pour sauvegarder **(CSAVE)** et charger **(CLOAD)** une zone de mémoire en stockant un ensemble de caractères différent ou un programme interne en langage machine. Parce que le reste de la mémoire de l'ORIC ne serait pas touchée par ce procédé, n'importe quel programme en mémoire en BASIC serait protégé des programmes internes ajoutés. Les zones de mémoire utilisables pour la sauvegarde des écrans sont les suivantes :
pour le mode Haute Résolution **(HIRES)**

CSAVE "nom de fichier" ,A 4Ø96Ø ,E 48ØØØ (ORIC 48K)
CSAVE "nom de fichier" ,A 8192 ,E 15232 (ORIC 16K)

pour les modes TEXTE et Basse Résolution **(LORES)**

CSAVE "nom de fichier" ,A 48ØØØ ,E 49119 (ORIC 48K)
CSAVE "nom de fichier" ,A 15232 ,E 16351 (ORIC 16K)

L'Adresse de début et de fin (End) accepteront des valeurs décimales ou hexadécimales, et encore une fois ,S peut être ajouté pour une sauvegarde en mode lent (Slow). Les blocs de mémoire sont automatiquement rechargés dans leurs emplacements mémoires d'origine donc seule l'instruction **CLOAD "NOM DE FICHIER"** est nécessaire. L'indicateur de type de fichier est **C** (pour Code) en sauvegarde ou en chargement des blocs mémoires.

Il y a deux autres possibilités d'utilisation de la cassette. La plus importante de ces commandes supplémentaires est la possibilité de

William Saint-Cricq / wsaintcr@wanadoo.fr Page 42 sur 163 15/08/2000

vérifier (**VERIFY**). Quand vous avez sauvegardé un programme e
souhaitez voir si le processus s'est bien déroulé, les vérifications sui
vantes peuvent être faites :

1. Rembobinez la cassette jusqu'au début du programme qu
vous venez de sauvegarder.

2. Vérifiez que le volume de votre magnétophone est réglé a
niveau correct pour le chargement et que la tonalité est au maxi
mum des aigus.

3. Assurez-vous que vos branchements de cordon sont corrects
L'extémité du côté de l'ORIC doit rester dans la même prise, l'autr
bout doit être branché à la prise entrée/sortie du magnétophone s
c'est une prise DIN ou, si vous utilisez des prises jack, la fiche jac
EAR doit être branché dans la prise EAR (la fiche jack MI
débranchée).

4. Tapez l'instruction suivante en commande directe :

CLOAD "NOM DE FICHIER", V ,S

(encore une fois le ,S ne doit être utilisé que si le programme a ét
sauvegardé en mode lent), le nom du programme peut être omis s
vous êtes certain de la place du programme, l'ordinateur commen
cera à chercher (**SEARCHING...**) à la manière habituelle, et quan
il repérera le programme il écrira **VERIFYING NOM DE FICHIE**
(plus l'identification de type de fichier). Si l'enregistrement s'éta
bien déroulé, le message Ø Verify errors detected apparaîtra à l
place habituelle du curseur. Si l'enregistrement s'est mal pass
(autre nombre que Ø pour les erreurs), vous devez retourner a
opérations décrites à l'étape 6 de la procédure de chargeme
(**CLOAD**) donnée ci-dessus, si un deuxième essai ne marche tou
jours pas.

L'autre commande supplémentaire de chargement disponible s
l'ORIC est la possibilité de liaison (**J**oindre). Celle-ci vous perm
d'enchaîner un second programme à la fin d'un programme déj
tapé et chargé dans votre ORIC. L'écriture pour cette opération e
la suivante :

CLOAD "NOM DE FICHIER", J ,S

Tout ce que cette particularité fait en vérité est d'empêch
l'ORIC de vider la mémoire comme il fait normalement quand u
écriture **CLOAD** est utilisée et d'insérer les nouvelles lignes de pr
gramme séquentiellement en mémoire. Si vous voulez enchaîner u
second programme de cette façon, vous devez vous assurer que *to*
les numéros de lignes du second programme sont plus grands que l
plus grand numéro de ligne du premier. S'il n'en est pas ainsi le pr
duit final ne tournera pas, car les numéros de ligne du second pr
gramme ne "s'entremêlent" pas avec ceux du premier, et l
deuxième programme est simplement inséré en mémoire au dess

du programme existant. Les programmes enchaînés ne doivent pas dépasser la capacité de mémoire disponible.

Il y a une autre possibilité d'enregistrement de fichier. **STORE** et **RECALL** permettent de sauvegarder des tableaux et de les relire ensuite dans un programme, fournissant ainsi un moyen de transférer des données d'un programme à un autre. A l'intérieur d'un programme, le contenu d'un tableau peut être défini, puis placé sur fichier cassette. N'importe quel type de tableau peut être stocké. L'écriture pour le stockage (**STORE**) est :

STORE V, "NOM DE FICHIER", S

où le V est le nom du tableau à sauvegarder (A$ par exemple, pour un tableau A$(3,4), G pour un tableau G(3Ø) etc.).

La procédure d'utilisation de l'instruction est la même que pour **CSAVE** et le mode rapide (24ØØ bauds) sera utilisé par défaut si le ,S est omis. Le message SAVING NOM DE FICHIER apparaît sur la ligne habituelle, suivi d'une lettre indiquant le type de tableau ; R pour nombre réel à virgule flottante, I pour un tableau d'entiers (Integer) et S pour les tableaux de chaînes (String).

RECALL rappellera le contenu d'un tableau préalablement sauvegardé sur cassette en utilisant **STORE** et la procédure est la même qu'en utilisant **CLOAD**. Le tableau pour stocker le tableau rappelé (RECALLed) doit avoir été dimensionné avant d'utiliser **RECALL**, ou une erreur **OUF OF DATA** se produit. La taille du tableau doit être la même (ou plus grande) que le tableau d'origine et du même type (entiers, chaînes ou réels). **RECALL** peut être utilisé à l'intérieur d'un programme, ou en commande directe mais dans ce dernier cas aucune instruction qui efface les variables stockées ne peut être utilisée (**CLEAR, RUN, etc.**) ou les valeurs stockées du tableau seront perdues.

L'écriture pour **RECALL** est :

RECALL V, "NOM DE FICHIER", S

où le V est le nom du tableau dans lequel les données rappelées (**RECALL**ed) doivent être placées. Il n'est pas utile que ce soit le nom attribué au tableau d'origine qui a été stocké mais il doit être du même type. Essayez les programmes exemples ci-dessous pour illustrer cela. Notez que bien que (dans l'exemple **RECALL**) le tableau B ait été dimensionné B(2Ø), la même taille que le tableau A(2Ø) utilisé pour le stockage, il pourrait être dimensionné plus grand. Essayez de changer la ligne 1Ø du programme pour lire **DIM B(25)** et rappelez (**RECALL**) le tableau nouveau. A partir de la cassette, la vitesse de transmission de données Slow (lente) (comme dans l'écriture ci-dessus en utilisant ,S) ou Fast (rapide) (en omettant ,S) peut être indiquée. La même vitesse doit aussi avoir été utilisée au moment de l'instruction **STORE**.

Numérisation http://wsc.n3.net
William Saint-Cricq / wsaintcr@wanadoo.fr Page 43 sur 163 15/08/2000

```
10 DIMA(20)
20 FOR K=0 TO 20
30 LET A(K)=2^K
40 NEXT K
50 PRINT"ENFONCEZ UNE TOUCHE QUAND VOU
S ETES   PRET A STOCKER"
60 GET A$
70 STORE A,"FICHTAB"
80 PRINT"TABLEAU STOCKE"
90 END
```

```
10 DIM B(20)
20 PRINT"METTEZ LE MAGNETOPHONE EN LEC
TURE"
30 PRINT"ENFONCEZ UNE TOUCHE"
40 GET A$
50 RECALL B,"FICHTAB"
60 CLS
70 FOR L=0 TO 20
80 PRINT B(L)
90 NEXT L
100 END
```

Entretien des cassettes

Si vous avez pris la peine de composer un programme et de le sauvegarder sur cassette, cela vaut la peine de faire en sorte que votre copie de programme dure aussi longtemps que possible. Il faut souligner que si une cassette n'est pas utilisée pendant un mois, il se peut que l'enregistrement soit altéré quand vous viendrez à vous en servir. Cependant, il y a des mesures à prendre pour minimiser ce risque :

— N'enregistrez pas sur les 10-15 premières secondes d'une cassette. La plupart des problèmes d'étirement et de détérioation du support magnétique arrivent sur cette section de bande.

Quand une cassette n'est pas utilisée assurez-vous toujours qu'elle soit replacée dans son étui.

Ne laissez jamais une cassette sur un poste TV ou sur n'importe quel autre appareil électrique. Les champs électromagnétiques générés par de tels équipements peuvent altérer les signaux stockés sur la cassette.

Ne touchez jamais la surface de la bande, et mettez un point d'honneur à rembobiner la cassette après usage afin que seul le morceau de plastique (la bande amorce) soit laissé à découvert.

Les nouvelles cassettes devraient toujours être filées et rembòbinées avant de tenter tout enregistrement. Ceci vous assurera d'une tension bien égale.

Nettoyez les têtes de lecture et d'enregistrement de votre magnétophone régulièrement. Ceci devrait être fait toutes les deux ou trois heures d'utilisation. Achetez une cassette démagnétisante et démagnétisez les têtes environ toutes les douze heures d'utilisation.

Une fois que vous avez enregistré un programme, assurez-vous que son nom et sa position sur la cassette sont bien inscrits sur la cassette et sur son étui. Assurez-vous aussi que vous avez noté en quel mode la cassette a été enregistré (c'est-à-dire lent (Slow) ou rapide (Fast)) et si le programme est encore en cours de composition, notez bien quelle version du programme a été enregistrée.

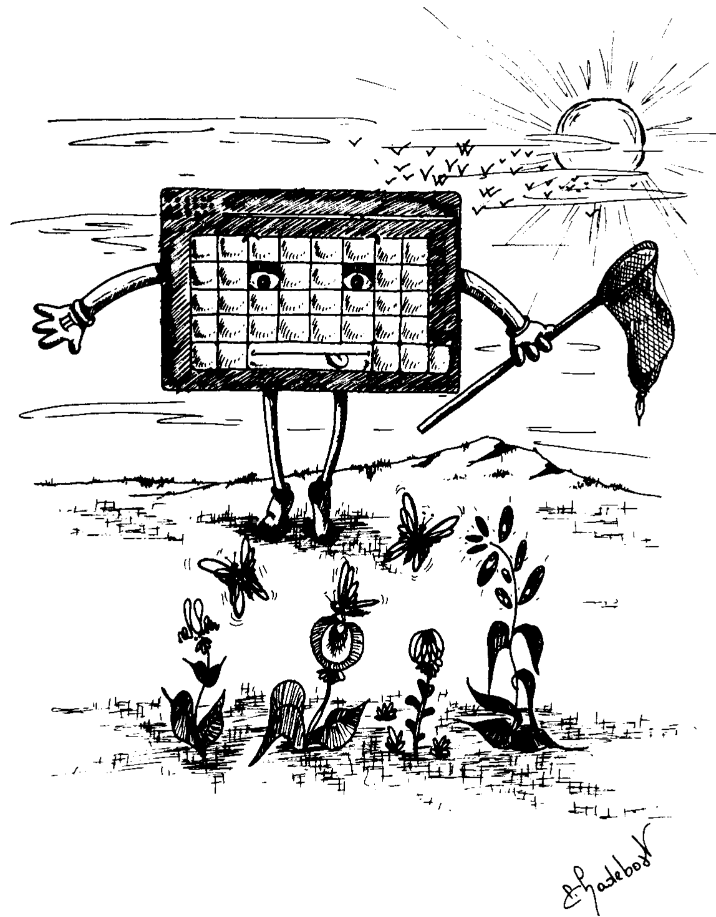
Pour la version finale d'un programme, les taquets de sécurité sur le dessus de la cassette doivent être enlevés pour empêcher tout effacement accidentel.

Faites des copies doubles des programmes importants. Rembobinez les cassettes de temps à autre, même si vous ne vous en servez pas pour charger des programmes. Ceci empêche une magnétisation par contact permanent d'une spire sur l'autre.

N.B. Certains magnétophones à cassette peuvent provoquer de faux messages d'erreur (**ERRORS FOUND**) après le chargement. Une variation du signal dans le cordon du magnétophone due par exemple à la recherche de niveaux automatiques d'enregistrement peut dans certains cas perturber les tests de données de l'ORIC ; ce qui entraîne l'affichage d'un message d'erreur. Malgré cela, les programmes se chargeront correctement mais le lancement automatique (AUTO-run), ne se fera pas, comme d'habitude. Le chapitre sur le langage machine contient un programme qui résout ce problème, dussiez-vous le rencontrer avec votre magnétophone.

Numérisation http://wsc.n3.net
15/08/2000
Page 44 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net



Chapitre 7
Graphiques et Couleur

Bien, maintenant nous sommes suffisamment familiarisés avec les énoncés les plus simples du BASIC pour nous tourner vers un des thèmes les plus avancés sur les micro-ordinateurs : Les Graphiques. En tant que micro-ordinateur, votre ORIC a de très bonnes capacités au niveau de la couleur et du graphisme. Il y a en tout 4 affichages séparés sur l'écran ou "modes". Ces "modes" sont : **TEXT, LORES 0, LORE 1** et **HIRES**. Dans ce chapitre, nous allons examiner chacun de ces "modes".

mode TEXT

C'est le mode dans lequel votre ORIC rentre lorsque vous le branchez et, comme le nom l'indique, c'est un mode prévu principalement pour l'affichage du texte. Comment affichons-nous le texte ? Et bien, nous avons déjà rencontré l'instruction **PRINT** qui affiche des caractères et des variables sur l'écran, mais cela vaut la peine d'examiner en détails les capacités de la touche **PRINT**. La forme la plus simple de la touche **PRINT** est montrée dans l'exemple ci-dessous, et si vous le tapez suivi de **RUN** vous obtiendrez l'affichage suivant sur votre écran :

```
10 PRINT "FRED"
   FRED
```

Ajoutez maintenant une autre ligne pour obtenir le programme suivant qui, lorsque vous appuyez sur **RUN**, vous donnera le résultat suivant :

```
10 PRINT "FRED"
20 PRINT "A"
   FRED
   A
```

Cela semble assez logique que l'ordinateur utilise une nouvelle ligne pour imprimer le "A" et en général l'ordinateur ira directement au début de la ligne suivante après avoir exécuté l'instruction de la touche **PRINT**. Cependant, il existe quelques formes spéciales de **PRINT** que nous pouvons utiliser pour modifier ce fonctionne-

15/08/2000
Page 45 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

ment. Ajustez une virgule à la fin de la ligne 1Ø et ensuite appuyez sur **RUN**. Cela devrait ressembler au listing ci-dessous et aboutir au résultats suivant :

```
10 PRINT "FRED",
20 PRINT "A"
   FRED    A
```

La virgule a évidemment changé la position de l'information sur l'écran dans la ligne suivante qui se trouve maintenant placée à la fin de l'instruction PRINT à la ligne 1Ø. L'écran peut être vu comme ayant 5 positions automatiques de tabulation espacées de 8 lettres, pour chaque ligne. Lorsqu'une virgule est utilisée pour séparer deux éléments qui vont être affichés ou comme dernier élément dans une instruction **PRINT**, alors la position **PRINT** est avancée au début de la position de tabulation suivante. Cela peut être très pratique lorsque nous souhaitons afficher des tableaux de signes.

Il y a un autre signe que nous pouvons utiliser comme séparateur dans l'instruction **PRINT** qui est le point-virgule (;). Modifiez de nouveau la ligne 1Ø et appuyez sur RUN pour lancer le programme. Cela devrait donner le listing et résultats suivants :

```
10 PRINT "FRED";
20 PRINT "A"
   FREDA
```

Et bien, il semblerait que nous avons modifié le sexe de FRED cette fois ! Le point-virgule implique que la position **PRINT** soit laissée où elle se trouvait lorsque l'on a arrêté afficher. L'exemple ci-dessous montre le résultat produit par deux modèles d'instruction **PRINT**. Remarquez qu'à chaque fois que ORIC doit afficher un nombre, il affiche un espace avant aussi bien qu'après le nombre.

```
10 PRINT 1,2;3
20 PRINT "A","B";"C"
 1       2  3
A       BC
```

Il y a une autre commande que nous pouvons utiliser pour modifier la fonction **PRINT**, modification qui se fait en utilisant la commande **TAB**. Celle-ci est semblable à la fonction **TAB** d'une machine à écrire. En inscrivant **PRINT TAB(n)** ; nous pouvons déplacer la fonction **PRINT** à la position du énième caractère le

long de la ligne. Essayez le programme ci-dessous qui devrait donner le résultat qui suit le listing.

```
10 FOR I=0 TO 5
20 PRINT TAB(I);"HELLO"
30 NEXT I

HELLO
HELLO
HELLO
 HELLO
  HELLO
   HELLO
```

Si l'on regarde le résultat, il semblerait que **TAB(Ø)**, **TAB(1)** et **TAB(2)** n'ont pas eu de résultat perceptible, mais c'est parce que la position du caractère **Ø** et celle du **1** sont réservées par l'ORIC à des codes spéciaux (nous y viendrons dans un moment). Donc la fonction **PRINT** est déjà arrivée à 2 espaces lorsque nous commençons une nouvelle ligne et, comme sur une machine à écrire, on ne peut pas **TAB**uler en arrière. Si nous souhaitons utiliser les positions des deux premiers caractères, **Ø** et 1, nous pouvons appuyer le **CONTROL** et], les touches ensemble. Ceci est une des combinaisons, que l'ORIC reconnaît comme une manette qui inverse l'état d'un interrupteur interne (dans ce cas l'interrupteur de protection de colonne), et comme il était allumé, nous l'éteindrons maintenant. Si nous lançons le programme à nouveau, nous obtiendrons le résultat montré ci-dessous :

```
HELLO
 HELLO
  HELLO
   HELLO
    HELLO
     HELLO
```

Nous en venons maintenant à la question, pourquoi les premières deux colonnes étaient-elles réservées au départ ? Et bien, si vous avez essayé l'exemple ci-dessus, vous l'avez certainement deviné maintenant. Ces colonnes altèrent la couleur du texte sur l'écran. Lorsque vous utilisez ces colonnes les caractères imprimés apparaissent en blanc sur fond noir au lieu de l'habituel noir sur fond blanc, tel l'encre, sur un morceau de papier. Ceci est exactement la raison pour laquelle l'on utilise ces colonnes. La première colonne assure d'habitude la couleur du papier et la seconde la couleur de l'encre.

15/08/2000 Page 46 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

Basculez l'interrupteur de protection à nouveau avec CTRL et] pour que les deux premières colonnes soient protégées de nouveau. Maintenant, nous pouvons essayer de changer les couleurs de l'encre et du papier sur l'écran. L'ORIC associe chaque numéro entre Ø et 7 avec une couleur déterminée et la commande de **PAPER n** (PAPIER) ou **INK n** (ENCRE) établira comme il convient les couleurs du fond ou du premier plan, si toutefois on se trouve dans la fourchette Ø-7. La couleur de chaque numéro est énoncée dans la liste suivante :

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

Essayez de taper **PAPER 1** et appuyez sur **RETURN**. L'écran devrait instantanément changer en un affichage de caractères noirs sur papier rouge. Essayer la commande **INK** de la même façon mais rappelez vous que si la couleur de **INK** est la même que celle de **PAPER** vous ne pourrez pas lire les caractères.

Sur l'ORIC toutes les couleurs sont contrôlées par des caractères spéciaux appelés "attributs". On les appelle parfois attributs collectifs parce qu'ils modifient tout ce qui suit à partir d'eux. Donc, ce que fait l'ORIC, à chaque fois que nous modifions une couleur, est de descendre le long d'une colonne protégée, attribuant une couleur dans cette position sur chaque ligne, ce qui logiquement concerne le reste de la ligne. Tapez le programme suivant et **LIST**ez le sur l'écran. Maintenant lancez le programme et il vous montrera toutes les combinaisons de **INK** et **PAPER** possibles. Rappelez-vous une fois encore que lorsque **INK** et **PAPER** sont de la même couleur, le texte semblera disparaître.

```
10 FOR I=1 TO 7
20 INK I
30 FOR J=0 TO 7
40 PAPER J
50 WAIT 50
60 NEXT J
70 NEXT I
80 INK 0
```


Revenons à la fonction **PRINT**, il y a un autre détail que nous pouvons utiliser pour changer la position à laquelle nous imprimons. Ceci se fait en utilisant le symbole @ (à) avec deux nombres. Le premier nombre est comme le numéro de colonne dans la fonction **TAB**, le deuxième nombre précise sur quelle rangée de l'écran nous allons nous déplacer. La position de la colonne peut être comprise entre Ø et 39 et la position de la rangée peut être comprise entre Ø et 26. Pour un exemple rapide lancez le programme suivant (ou faites-le rentrer en commande directe).

```
10 PRINT @10,10;"HELLO"
```

Comme nous nous y attendions, "HELLO" s'affiche à un quart de l'écran en largeur et à environ au tiers du bas de l'écran. Pas très excitant jusque-là pensez-vous, mais cette capacité de pouvoir afficher n'importe où sur l'écran est la base de la plupart des jeux ! L'exemple suivant devrait vous donner quelque éclaircissement à ce sujet.

```
10 FOR X=2 TO 30
20 PRINT @X,10;"WHIZZ!"
30 NEXT X
```

Cette sorte d'animation est possible dans toute les directions de l'écran, mais nous ne voulons pas voir le curseur clignoter pendant que nous affichons des caractères sur l'écran. Pour éviter cela nous utilisons une des autres manettes, **CTRL-Q**. Ceci allume et éteint le curseur et pour faire cela dans notre programme nous utilisons la fonction **CHR$** qui produit des caractères à partir de leur numéro de code. Nous devrions nous rappeler de le rallumer à la fin de notre programme également, voici donc une version modifiée du listing précédent qui donne ceci :

```
10 PRINT CHR$(17)
20 FOR X=2 TO 30
30 PRINT @X,10;"WHIZZ!"
40 NEXT X
50 PRINT CHR$(17)
```

Le programme suivant est un jeu utilisant toutes les choses que nous avons recontrées jusqu'à maintenant. Vous êtes un chasseur de canards et vous devez tirer vos flèches sur les canards ">" qui volent sur le haut de l'écran. Le programme a été décalé pour mon-

Numérisation http://wsc.n3.net
William Saint-Cricq / wsaintcr@wanadoo.fr Page 47 sur 163 15/08/2000

Numérisation http://wsc.n3.net

trer sa structure, mais les colonnes et espaces initiaux n'affectent pas son déroulement

```
10  : CLS:S=0:PX=20
20  : PAPER 6:PRINT CHR$(17);CHR$(6)
30  : REPEAT
40  :   PRINT @10,0;"SCORE";S
50  :   WAIT RND(1)*100
60  :   DX=2 : REPEAT
70  :     IF KEY$=CHR$(9) AND PX<37 THEN PX=PX+1
80  :     IF KEY$=CHR$(8) AND PX>2 THEN PX=PX-1
90  :     PRINT @ PX,25;" ^ ";
100 :     DX=DX+1 : PRINT@ DX,3;" >"
110 :     IF KEY$="S" THEN 230 ELSE IF KEY$<>" "
          THEN 200
120 :     MX=PX+1 : MY=24
130 :     REPEAT
140 :      PRINT@MX, MY;"!"
150 :      PRINT@MX, MY+1;" ";
160 :      MY=MY-1
170 :   UNTIL MY=3
180 :     IF MX=DX+1 THEN S=S+1:DX=38
190 :     PRINT @MX,MY+1;" " :PRINT @ MX,MY;" "
200 :   UNTIL DX=38
210 :   PRINT@ 39,3;" ";
220 : UNTIL KEY$="S"
230 : PRINT CHR$(17);CHR$(6)
```

Maintenant, vous désirez peut-être savoir comment l'on peut obtenir plus de deux couleurs en même temps sur l'écran, pour donner par exemple aux canards une couleur différente de celle du chasseur. C'est ici que nous revenons à ces attributs collectifs qui changent le reste de la ligne. Nous pouvons produire des symboles en tapant **ESCAPE** suivi de **@, A, B, C, D, E, F** ou **G**. Ceux-ci donnent les attributs pour **INK** respectivement de **0 à 7**. De la même manière **ESC P** à **ESC W** donne les attributs pour **PAPER**. Il est important de bien comprendre que ces symboles sont en fait représentés en caractères sur l'écran et qu'ils changent le reste de la ligne qui suit leur position. Pour réaliser cela de l'intérieur d'un programme nous devons d'abord imprimer le caractère qui signifie **ESCAPE (CHR$(27))** et ensuite le caractère de la lettre que nous désirons faire succéder au code **ESCAPE**. Ceci peut être, soit directement imprimé avec "**Q**", soit en utilisant la fonction **CHR$**. **CHR$(64 + n)** donne l'attribut pour **INKn et CHR$(80 + n)** donne le symbole pour **PAPERn**. Le programme suivant va expliquer ceci

pour les couleurs de **PAPER** et vous pouvez essayer de changer le **80** dans la fonction **CHR$** en 64 pour voir la même chose se passer avec le premier plan ou les couleurs de **INK**

```
5 CLS
10 FOR X=0 TO 7
15 PRINT
20 PRINT CHR$(27);CHR$(80+X);"  BONJOU
R !"
30 NEXT X
```

Il y a d'autres attributs spéciaux qui peuvent être portés sur l'écran de la même manière et ceux-ci produisent des effets variés sur le reste de la ligne. Le tableau ci-dessous montre les caractères qui peuvent succéder à **ESCAPE** et les attributs ou effets qu'il produisent.

ESCAPE @	0	encre noire
ESCAPE A	1	encre rouge
ESCAPE B	2	encre verte
ESCAPE C	3	encre jaune
ESCAPE D	4	encre bleue
ESCAPE E	5	encre magenta
ESCAPE F	6	encre cyan
ESCAPE G	7	encre blanche
ESCAPE H	8	texte standard
ESCAPE I	9	texte alterne
ESCAPE J	10	standard db hauteur
ESCAPE K	11	alterne db hauteur
ESCAPE L	12	standard clignotant
ESCAPE M	13	alterne clignotant
ESCAPE N	14	stnd(db h)clignotant
ESCAPE O	15	alt(db h)clignotant
ESCAPE P	16	papier noir
ESCAPE Q	17	papier rouge
ESCAPE R	18	papier vert
ESCAPE S	19	papier jaune
ESCAPE T	20	papier bleu
ESCAPE U	21	papier magenta
ESCAPE V	22	papier cyan
ESCAPE W	23	papier blanc

Le listing suivant contient de nombreuses nouvelles particularités, nous allons donc parcourir le listing ligne par ligne. La première ligne affiche le caractère le code ASCII 12 à l'écran. Ceci est le caractère d'effacement de l'écran, l'ordinateur prend donc une nouvelle feuille de papier — ce qui est simplement une autre façon

15/08/2000
Page 48 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

de faire **CLS**. La deuxième ligne affiche une autre manette, celui ci **CTRL D** change l'affichage double. Quand cela est fait, tout ce qui est affiché à l'écran est répété au rang en desous. La troisième ligne produit l'attribut pour une double hauteur avec clignotement. La ligne **40** affiche le message sur l'écran, où il est imprimé deux fois. La ligne **50** inverse l'affichage double à nouveau.

```
10 PRINT CHR$(12)
20 PRINT CHR$(4);
30 PRINT CHR$(27);"N";
40 PRINT"UN GRAND BONJOUR CLIGNOTANT"
50 PRINT CHR$(4);
```

Essayez d'enlever les lignes qui provoquent le double affichage ou la ligne qui produit l'attribut. Les effets produits lorsque vous effectuez ces modifications devraient vous aider comprendre le rôle que chacun d'entre eux joue en produisant le message. Remarquez que nous nous sommes arrangés pour avoir la ligne du haut de nos caractères à double hauteur sur la ligne 1 et la moitié de la ligne du bas sur une ligne paire. C'est parce que l'ordinateur doit faire en sorte que si vous utilisez une double hauteur, toutes les moitiés supérieures des caractères sont sur des lignes impaires avec leurs moitiés inférieures sur la ligne paire en-dessous. Essayez d'ajouter une ligne comme celle-ci, pour commencer à afficher sur la mauvaise ligne et voyez ce qu'il se passe lorsque nous le faisons mal.

```
15 PRINT
```

Voici un programme court qui va montrer comme cette disposition peut être utile pour trouver le haut et le bas des lignes

```
10 CLS
20 REPEAT
30 FOR X=1 TO 7
40 PRINT CHR$(27);"J";CHR$(27);CHR$(80
+X)
50 PRINT"      MANUEL DE L'ORIC"
60 NEXT
70 UNTIL KEY$<>""
```


LORES 0

Dans ce mode l'ORIC affiche un écran noir sur lequel les caractères standards peuvent être tracés (PLOTed) d'une manière semblable à celle où ils ont été affichés sur l'écran en mode TEXT. En fait l'écran **LORES0** est similaire à l'affichage texte mais l'attribut au début de l'écran est maintenant celui qui sélectionne l'ensemble des caractères standards. Les couleurs blanc sur fond noir sont les couleurs de défaut que l'on utilise si aucun attribut ne leur a été trouvé.

La commande **PLOT** a deux coordonnées, comme pour **PRINT@**. La première, la coordonnée X, peut aller de 0 à 39, tandis que la seconde, la coordonnée Y peut varier de 0 à 26. L'utilisation de **PLOT** ne change pas la position **PRINT**. La commande **PLOT** n'est pas aussi souple que la commande **PRINT @** étant donné qu'elle nécessite une chaîne de caractères. Donc, si nous désirions tracer (PLOT) un nombre, nous devrions utiliser une commande telle que : **PLOT 10, 10, STR$(1)**.

En utilisant **PLOT** nous pouvons également mettre sur l'écran des caractères parliculiers et à cet égard nous obtenons un énoncé bien différent de la commande **PRINT@**. Nous pouvons utiliser **PLOT** pour mettre des caractères inversés sur l'écran. Pour obtenir des caractères inversés, nous devons utiliser le code ASCII, plus 128 et cela ne peut être fait à partir de **PRINT**. Lorsqu'un caractère est affichés selon le mode inverse, les couleurs utilisées sont les couleurs logiques inversées. Ce qui ne veut pas dire que le caractère apparaîtra comme couleurs **PAPER** sur **INK**, mais qu'il apparaîtra comme l'inverse logique de **INK** sur l'inverse logique de **PAPER**. L'inverse logique d'une couleur se trouve en retranchant au nombre 7 la couleur de code. Donc, si le **PAPER** était 1 il serait affiché comme si PAPER était (7-1) ou 6 et pareillement pour **INK**.

Le listing suivant crée la chaîne de caractères inversés à partir de la chaîne normale et ensuite utilise **PLOT** pour les placer sur l'écran. Juste pour prouver que **PRINT** ne permet pas de caractères inversés, ils sont aussi imprimé en haut de l'écran.

```
10 CLS
20 PAPER 1:INK 2
30 A$="HELLO"
40 FOR I = 1 TO LEN(A$)
50 B$=B$+CHR$(ASC(MID$(A$,I))+128)
60 NEXT
70 PLOT 10,10,A$
80 PLOT 10,11,B$
90 PRINT A$,B$
```


William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Une commande associée est la fonction **SCRN** qui renvoie la valeur affecté à l'emplacement d'écran spécifié par X et Y (comme pour **PLOT**). Cette valeur est habituellement la valeur ASCII du caractère affiché à cette position, mais elle pourrait aussi retourner la valeur de l'attribut figurant là s'il n'y a effectivement rien d'affiché. Le listing suivant explique ces 2 façons d'utiliser **SCRN** :

```
10 LORES0
20 PLOT 10,10,"A"
30 PRINT SCRN(10,10)
40 PRINT SCRN(0,0)
```

Le 8 produit par la seconde utilisation de **SCRN** peut être expliqué en regardant le tableau des effets de **ESC**ape et de symboles donné plus haut. Cette valeur représente simplement l'attribut pour la forme standard. En fait, ce n'est pas important si vous écrivez cela ou non, parce que c'est, bien sûr, la valeur par défaut qui se place de toute façon. La raison pour laquelle ces codes sont mis ici deviendra plus évidente dans le chapitre suivant sur le mode **LORES1**. Comme l'utilisation précédente de **SCRN** l'implique, l'écran est effectivement représenté en mémoire par un ensemble de 112Ø emplacements de mémoire (ce qui est 28*4Ø, étant donné qu'il y a 28 lignes sur l'écran y compris la position de la ligne du haut). Ces emplacements de mémoire s'étendront de # BB8Ø ou décimal 48ØØØ à # BFDF ou 49119 et nous pourrons atteindre ces emplacements individuellement en utilisant **POKE**. Essayer le programme suivant qui remplit tous ces emplacements avec le code ASCII de "A"

```
10 FOR I=#BB80 TO #BFE0
20 POKE I,65
30 NEXT I
```

LORES 1

D'une façon semblable à **LORESØ** qui était le mode **TEXT** avec toute la première colonne initialisée à 8, **LORES1** initialise la première colone à 9, (l'attribut pour l'ensemble des caractères alternés (deuxième jeu de caractères)). Ainsi, nous pouvons afficher une variété de caractères alternés et et ils apparaîtront en combinaisons de 6 petits carrés à l'intérieur d'un bloc de caractères. Essayez de tracer l'alphabet entier sur l'écran. Vous verrez qu'il semble y avoir un modèle pour la disposition des blocs. Il y a en fait une disposition très logique de leurs configurations. Pour comprendre comment ces blocs sont établis, nous devons d'abord retourner aux bases.

A l'intérieur de l'ordinateur tous les nombres sont gardés en système binaire. Pour chaque emplacement nous avons huit chiffres binaires, chacun d'eux peuvent être soit 1 soit Ø. Dans les emplacements sur l'écran ces bits peuvent avoir chacune une signification spéciale. Comme nous l'avons vu, les bits valent 128, 64, 32, 16, 8, 4, 2 et 1. De ce que nous venons de voir, il est évident que le bit le plus élevé (128 ou b7), indique que la vidéo inversée doit être utilisée pour l'affichage de ce caractère déterminé, mais vous devez aussi remarquer que cela ne touche pas le reste de la ligne ! Les deux bits inférieurs suivants, b6 et b5 (valant respectivement 64 et 32), peuvent avoir un effet particulier lorsqu'on les considère ensemble. Si ils sont toutes deux à zéro, alors l'emplacement est un attribut et les bits restant ajouteront un nombre entre Ø et 31 qui peut être consulté dans la table des effets de ESCape ci-dessus pour voir quel effet cela fait. Nous pourrions également avoir l'ensemble des bits les plus élevés rendant le nombre supérieur à 128, mais les bits 5 et 6 étant zéro continueront à indiquer que c'est vraiment un attribut.

Essayez le programme suivant

```
10 LORES1
20 A$=CHR$(145)+"ABCDEFGHIJKLMNOPQRSTU
VWXYZ"
30 PLOT 5,10,A$
```

145 est l'attribut pour **PAPER** rouge avec 128 ajouté pour indiquer l'inverse. Cela signifie que l'espace où se trouve les attributs est montré à l'inverse du rouge c'est à dire : cyan, et le reste de la ligne est imprimé en rouge. Vous avez maintenant remarqué que le rouge ne va pas jusqu'à la fin de la ligne, mais ne parvient qu'au dernier caractère alterné affiché ! Et bien, une rapide vérification avec **SCRN** devrait éclairer la situation, dans le sens où tous les autres emplacements ont été fixés à 16, qui est le symbole pour PAPER noir. Maintenant, regardons comment les bits qui choisissent le caractère sont agencés. Rappelez vous que nous avons dit que ce serait un attribut seulement si b5 et b6 sont tous les deux Ø. Pour l'ensemble des caractères normaux tous les bits de b6 jusqu'à bØ sont ajoutés pour obtenir le code du caractère qui va s'afficher. Cela se passe également dans l'ensemble alterné, mais la façon dont les caractères ont été définis est destinée à permettre à chacun des six blocs d'être représenté par un bit. Il y a 7 bits dans le code, mais nous devons en utiliser un pour empêcher le code d'être un symbole. B5 a été choisi pour être toujours un, et les autres bits sont assignés comme suit : Le bloc du haut à gauche est bØ, celui du haut à droite est b7, celui du milieu à gauche est b2, le milieu droit est b3,

Numérisation http://wsc.n3.net
William Saint-Cricq / wsaintcr@wanadoo.fr Page 50 sur 163 15/08/2000

Numérisation http://wsc.n3.net

celui du bas à gauche est b4 et du bas à droite est b6. Pour obtenir un caractère avec l'ensemble des blocs de droite, nous ajoutons simplement les valeurs des bits représentant chaque bloc, ensuite ajoutez 32 pour le bit 5. De cette façon, nous pouvons considérer les graphiques de l'ORIC comme des graphiques de **RESOLUTION MOYENNE**. Nous pouvons considérer l'écran comme étant composé des blocs Ø à 79 en largeur et des blocs Ø à 8Ø en hauteur. Le programme suivant montre un sous-programme, qui tracera chaque bloc individuel selon ce système de coordonnés, en utilisation. Nous devons évidemment éviter de les tracer aux coordonnés X inférieures à deux étant donné que ceux-ci se trouveraient au même endroit que le premier caractère et effaceraient l'attribut du jeu de caractères.

```
10 DIM P(2,3)
20 FOR I=1 TO 3
30 READ P(1,I),P(2,I)
40 NEXT I
50 DATA 1,2,4,8,16,64
100 LORES1
110 Y=40
120 FOR X=2 TO 79
130 GOSUB 1000
140 NEXT X
150 FOR X=2 TO 79 STEP0.2
160 Y=40+SIN(X/10)*35
170 GOSUB 1000
180 NEXT X
190 END
1000 '.PLOT X,Y...RESOLUTION MOYENNE
1010 CX=INT(X/2):PX=INT(X)-CX*2+1
1020 CY=INT(Y/3):PY=INT(Y)-CY*3+1
1030 P=#BB80+(CY+1)*40+CX
1040 Q=PEEK(P):IF Q<32 THEN Q=32
1050 Q=Q OR P(PX,PY):IF Q>95 THEN Q=Q-
     32
1060 POKE P,Q
1070 RETURN
```


HIRES

Ceci est le mode de graphiques de Haute Résolution. Nous pouvons placer des points n'importe où sur l'écran à une définition de 24Ø points en largeur et 2ØØ points de hauteur. Comme avec tous les autres modes le coin en haut à gauche est Ø,Ø. Il y a plusieurs commandes internes complexes et puissantes pour les graphiques **HIRES** sur votre ORIC. Nous commencerons par considérer les deux commandes les plus simples. La première est le **CURSET**. C'est la commande que nous utilisons pour fixer la position de départ de notre curseur de graphiques. Cette commande a trois paramètres. Les deux premiers sont simplement les coordonnées X et Y du point en question. Le troisième paramètre est celui qui est utilisé par presque toutes les commandes de graphiques **HIRES**, donc nous allons nous en occuper en détails et nous nous y référerons plus tard. Ce dernier paramètre est connu en tant que paramètre premier plan/second plan ou paramètre fb (en anglais : fore/back). Cela concerne la façon dont les points sont tracés sur l'écran. Le paramètre fb peut avoir n'importe laquelle des 4 valeurs de Ø à 3. Les valeurs et la manière dont elles affectent les commandes traçantes sont résumées dans le tableau suivant.

Effet fb	
Ø	Les points sont tracés dans la couleur du second plan (ceci peut être également vu comme non-traçage d'un point)
1	Les points sont tracés dans la couleur du premier plan.
2	Les points sont intervertis ex : s'ils étaient de la couleur du premier plan avant d'être tracés, ils sont ensuite changés dans la couleur du second plan et vice versa.
3	Nul. Les points ne sont pas modifiés bien que la position du curseur graphique soit établi au dernier point rencontré.

La commande de graphiques suivante, est **DRAW**. Cette commande a trois paramètres comme pour **CURSET**, qui sont les paramètres X et Y suivis du paramètre fb. Néanmoins, **DRAW** fonctionne d'une façon très différente de celle de **CURSET**. Les valeurs X et Y déterminées ne sont pas les coordonnées d'un point sur l'écran mais sont relatives à la position actuelle des graphiques. Cela signifie qu'elles sont ajoutées à la position en cours pour obtenir les coordonnées du point vers lequel la ligne sera tracée. Nous devons nous assurer que ce point ne se trouvera pas en dehors de l'écran ou l'ORIC devra s'arrêter et nous signaler qu'il y a une erreur dans notre programme. Le programme suivant démontre la différence entre **CURSET** et **DRAW** en traçant d'abord les points dans les coins de l'écran, et ensuite, après avoir appuyé une touche, en traçant les lignes et en les faisant se rejoin-

William Saint-Cricq / wsaintcr@wanadoo.fr Page 51 sur 163 15/08/2000

dre. Remarquez que nous utilisons **CURSET** pour établir la position de départ avant de tracer les lignes.

```
10 PAPER 0:INK7
20 HIRES
30 CURSET 0,0,1
40 CURSET 0,199,1
50 CURSET239,199,1
60 CURSET239,0,1
70 GET A$
80 CURSET 0,0,3
90 DRAW 0,199,1
100 DRAW 239,0,1
110 DRAW 0,-199,1
120 DRAW -239,0,1
```

Il y a une troisième commande de graphiques, qui est moins souvent utilisée, qui s'appelle **CURMOV**. C'est la version de "mouvement relatif" de **CURSET**. Les paramètres X et Y sont ajoutés à (ou soustraits de) la position de graphiques en cours quand ils sont en position **DRAW**. En fait, cette commande est presque la même chose que d'utiliser **DRAW** avec le paramètre fb établi sur 3. Essayez les deux versions suivantes d'un programme pour étudier les différentes possibilités de **CURSET** et **CURMOV**.

```
10 PAPER 0:INK7
20 HIRES
30 FORI=1 TO 15
40 CURSET I,I,1
50 NEXT I
```

```
10 PAPER 0:INK7
20 HIRES
30 FORI=1 TO 15
40 CURMOV I,I,1
50 NEXT I
```

Pour mettre en évidence la grande définition et la précision de traçage que l'on peut atteindre sur l'écran de l'ORIC, essayez le programme suivant. Cela devrait vous donner une image de télévision qui a des couleurs qui ondulent et où les lignes sont très près les unes des autres. C'est parce que votre ORIC est destiné à exploiter la capacité d'affichage de votre poste de télévision à ses limites et à moins que vous n'ayez un intercepteur/TV très coûteux, c'est la définition d'image la plus correcte qu'il puisse traiter.

```
10 PAPER 0:INK7
20 HIRES
30 FORI=1 TO 239 STEP 2
40 CURSET I,0,1
```



```
50 DRAW 239-I,199,1
60 CURSET I,199,1
70 DRAW -I,-199,1
80 NEXT I
```

Une caractéristique plus approfondie que l'ORIC associe avec le traçage de lignes est la posibilité **PATTERN** (configuration) qui vous permet de préciser le type de ligne tracé, comme des points, des tirets, etc. La commande **PATTERN** a un paramètre entre Ø et 255, et cela définit, par son équivalent binaire, une configuration qui sera utilisé pour tracer n'importe quelle ligne formée de huit points. Le **PATTERN** par défaut est 255 qui est en binaire 11111111 alors tous les points d'une ligne sont tracés. Si nous devions utiliser **PATTERN 85** qui en binaire est **01010101** nous obtiendrions des lignes tracés avec un point sur deux omis.

```
10 HIRES
20 FOR I=0 TO 31
30 FOR J=0 TO 7
40 PATTERN J*32+I
50 CURSET J*30,I*6,3
60 DRAW 24,0,1
70 NEXT J
80 NEXT I
```

Le programme suivant trace un échantillon de la longueur d'une ligne dans chacune des configurations possibles, pour que vous puissiez voir ce qui est réalisable.

```
10 PAPER 0:INK 7
20 HIRES
30 INPUT"PATTERN ";P
40 PATTERN P
50 FORI=0 TO 199 STEP5
60 CURSET I,0,1
70 DRAW 199-I,I,1
80 CURSET I,199,1
90 DRAW -I,I-199,1
100 NEXT I
```

Nous arrivons maintenant aux commandes de graphiques développés disponibles sur votre ATMOS.

La première de ces commandes s'appelle **CIRCLE** et elle a deux paramètres. Le premier paramètre est le rayon du cercle et le

Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr Page 52 sur 163 15/08/2000

Numérisation http://wsc.n3.net

second est le paramètre fb. Le cercle sera tracé en utilisant la position des graphiques en cours comme le centre. Nous devons nous assurer, comme avec **DRAW**, que le cercle ne disparaît pas de l'écran ou nous causerions une erreur. Le programme ci-dessous montre la commande **CIRCLE** en action, en choisissant un centre de coordonnées X et Y de déterminées au hasard. La distance minimum de ce point au bord de l'écran est alors calculée, comme ceci est le rayon maximum que nous puissions utiliser par un cercle autour de ce point. Remarquez que Ø n'est pas une valeur valable pour le rayon, nous n'essayons donc pas de tracer un cercle dans ce cas. Le point central du cercle a été établi avec **CURSET** en utilisant un paramètre fb de 1, pour que vous puissiez voir où il se trouve.

```
10 PAPERØ:INK7
20 HIRES:REPEAT
30 X=INT(RND(1)*239)
40 Y=INT(RND(1)*199)
50 IF Y>99 THEN DY=199-Y ELSE DY=Y
60 IF X>119 THEN DX=239-X ELSE DX=X
70 IF DX>DY THEN R=DY ELSE R=DX
80 CURSET X,Y,1
90 IF R>Ø THEN CIRCLE R,1
100 UNTIL KEY$="S"
```

Le programme suivant montre une utilisation légèrement plus constructive de **CIRCLE** avec les centres non tracés.

```
10 PAPERØ:INK7
20 HIRES:REPEAT
30 FOR X=3 TO 50 STEP 3
40 CURSET X+50,100,3
50 CIRCLE X*2,1
60 NEXT
```

D'une façon similaire à celle dont nous lisons les contenus d'un emplacement particulier d'écran dans les modes **TEXT** avec la fonction **SCRN**, nous pouvons examiner l'état de n'importe quel point sur l'écran avec la commande **POINT**. Cette dernière comporte deux paramètres, logiquement, les coordonnées X et Y du point qui va être examiné. La valeur 1 est retournée par la fonction si le **POINT** est affiché dans la couleur du premier plan et Ø est retourné si il est affiché dans la couleur du second plan. Ce n'est pas la même chose que de lire les contenus d'un emplacement entier comme nous l'avons fait avec la commande **SCRN**.

C'est parce que chacune des 2ØØ lignes de l'écran, est en fait, constituée de quarante emplacements de mémoire de l'écran. Les bits dans ces emplacements peuvent indiquer, comme sur l'écran **TEXT**, qu'un attribut est présent à cet emplacement (bits 5 et 6 tous deux Ø). Si cela n'est pas le cas, alors les six bits de rang inférieur représentent la position d'une rangée de 6 points le long de la ligne, où un binaire 1 correspond à un point de premier plan et un binaire Ø correspond à un point de second plan. C'est comme cela que nous obtenons 24Ø ou 6*4Ø points le long de chaque rangée. Le bit 7 est utilisé pour indiquer qu'un bloc particulier de 6 points va être affiché dans les couleurs inversées logiques et de cette façon nous pouvons utiliser quatre couleurs, bien que nous n'ayons établi qu'une couleur de premier plan. Nous pouvons établir les attributs en utilisant **PAPER** et **INK** tout comme nous l'avons fait en mode **TEXT**, et ces commandes placeront un attribut dans la première et la seconde position de chacune des 2ØØ rangées de l'écran à haute résolution. Bien sûr, cela signifie que nous ne pourrons tracer des points avec des coordonnées X inférieures à 12, parce qu'ils se trouveraient à l'intérieur des emplacements utilisés par les attributs. Nous abordons maintenant la question : comment accédons-nous au bit inverse dans les emplacements de mémoire. Et bien, c'est une des capacités de la commande **FILL** pour des graphiques de haute résolution. Avec ceci nous pouvons remplir **(FILL)** un bloc d'emplacements à l'écran de tant de rangées à la verticale, de tant de segments d'emplacements à l'horizontale, avec n'importe quel nombre que nous choisissons. Nous devons placer le curseur de graphiques sur la ligne du haut et dans le coin de gauche du bloc choisi et ensuite lancer la commande **FILL** avec les trois paramètres suivants. Le paramètre 1 correspond au nombre de rangées verticalement, le paramètre 2 correspond au nombre de segments tracés horizontalement et le paramètre 3 correspond au nombre que nous désirons mettre dans tous ces emplacements. Nous pourrions utiliser cela pour remplir un espace sur l'écran avec une configuration de points, mais dans ce cas nous l'utiliserons pour fixer le bit le plus élevé de tous les emplacements sur le côté droit de l'écran. Remarquez que nous fixons également le bit 6 pour que les emplacements ne soient pas considérés comme emplacements d'attribut, donc 192 dans la ligne 4Ø est 128 + 64.

```
10 HIRES
20 PAPER 2:INK 4
30 CURSET 120,0,3
40 FILL 200,20,192
50 CURSET 120,100,3
60 CIRCLE 90,1
```


William Saint-Cricq / wsaintcr@wanadoo.fr Page 53 sur 163 15/08/2000

Comme nous l'avons mentionné plus haut, l'écran **HIRES** est composé de 200*40 emplacements et nous pouvons les aborder directement en utilisant **POKE**. Ces emplacements sont situés entre # A000 et # BF40 et le programme suivant fixe à 65 tous les emplacements. Cette fois-ci, le 65 est interprété non pas comme le code ASCII pour A mais signale avec le bit 6 qu'il ne s'agit pas d'un attribut, et avec le bit 0 que le point de gauche de chaque ligne de 6 est dans la couleur du premier plan.

```
5 HIRES
10 FOR I=#A000 TO #BFE0
20 POKE I,65
30 NEXT
```

Comme nous pouvons fixer par **POKE** n'importe quelle valeur que nous voulons sur l'écran, nous pouvons utiliser cette possibilité pour établir un affichage de couleurs de Haute Résolution utilisant les huit couleurs. Le programme suivant produit une une striure colorée sinusoïdale de haut en bas de l'écran de cette façon :

```
10 HIRES
20 FOR Y=0 TO 199
30 S=INT(SIN(Y/10)*8)
40 FOR I=0 TO 39
50 P=#A000+Y*40+X
60 Q=(S+X)-INT((S+X)/8)*8+16
70 POKE P,Q
80 NEXT
90 NEXT
```

En utilisant cette méthode pour placer des attributs sur l'écran **HIRES**, nous pouvons produire plus de couleurs avec l'ORIC. Nous mélangeons simplement deux couleurs différentes ensemble sur des lignes alternées, et nous pouvons avoir au moins 36 couleurs différentes. Le programme suivant montre quelques-unes des couleurs qui peuvent être produites. On peut même obtenir plus de teintes, en superposant des configurations colorées en premier plan sur ces fonds rayés.

```
10 PAPER 0:INK7
20 HIRES
30 FORA=0 TO 7
40 FORB=0 TO 7
50 FOR Y=B*20 TO B*20+20
```



```
60 P=#A000+Y*40+A*5
70 IF INT (Y/2)=Y/2 THEN Q=16+A ELSE Q
   =16+B
80 POKE P,Q
90 NEXT
100 NEXT
110 NEXT
```

Jusqu'ici, l'écriture sur l'écran **HIRES** a été négligée. Nous pouvons écrire avec la commande **CHAR**, qui est une des commandes les plus puissantes et vraiment passionnante de votre ORIC. Cette commande nous permet de placer des caractères à une précision d'un point, et non uniquement aux positions des caractères, une possibilité que l'on ne trouve que sur un seul (le plus coûteux) des concurents de l'ORIC.

Les caractères sont placés sur la position graphique avec la commande **CHAR** suivie de trois paramètres. Le premier paramètre est le code ASCII du caractère à être tracé, le second paramètre est l'ensemble de caractères dans lequel le caractère va être pris (0 pour l'ensemble standard et 1 pour l'ensemble alterné), et le troisième paramètre est notre vieil ami, le paramètre fb. Cette capacité vraiment remarquable, qui nous permettra des affichages animés et des jeux de superbe qualité à construire sur le ATMOS, est montrée par le programme ci-dessous, qui utilise une sous-programme pour placer une chaîne de caractères sur l'écran **HIRES**.

```
10 HIRES
20 PAPER 1:INK 3
30 A$="TARTEMOLLE"
40 FORW=1 TO 20
50 CURSET 20,50+W*3,3
60 GOSUB 100
70 NEXT
80 END
100 REM...CHAINE AVEC ESPACES
110 FOR I=1 TO LEN(A$)
120 CHAR ASC(MID$(A$,I)),0,1
130 DRAW W,0,3
140 NEXT I
150 RETURN
```

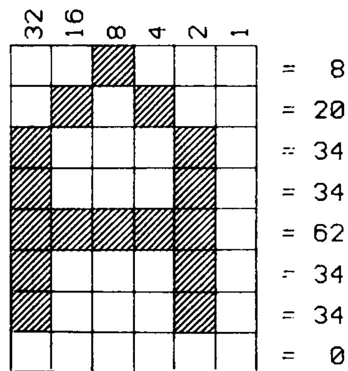

Numérisation http://wsc.n3.net
William Saint-Cricq / wsaintcr@wanadoo.fr   Page 54 sur 163   15/08/2000

Numérisation http://wsc.n3.net

CARACTERES GRAPHIQUES

Après avoir examiné les différents modes qui nous sont disponibles, nous arrivons maintenant à la possibilité de graphique de l'ORIC la plus souple : les caractères. Lorsque nous avons mentionné les jeux dans le chapitre précédent, vous avez peut-être pensé qu'il n'allait pas être très excitant de regarder une chaîne de lettres se déplacer autour de l'écran, quelle qu'en soit la précision. Mais dans ce chapitre nous allons voir que nous ne sommes pas limités à n'utiliser que des lettres. En fait, nous n'avons aucune limite en ce qui concerne les objets que nous utilisons ! D'abord, revoyons ce que nous connaissons sur les caractères.

Nous avons utilisé des caractères dans les affichages **TEXT** et **HIRES**, et maintenant vous vous demandez peut-être comment ils sont fait. La réponse à cette question peut être trouvée en étudiant ce qui se passe exactement lorsque nous mettons un caractère sur l'écran **HIRES**. A ce moment, l'ordinateur doit savoir quels points tracer dans la couleur du premier plan, et lesquels doivent être tracés avec la couleur du second plan à l'intérieur de l'espace autour du caractère. L'ordinateur trouve cette information, en regardant un tableau qui est conservé dans la mémoire. Il y a dans ce tableau huit données pour chaque caractère. Ces données peuvent être considérées comme huit configurations binaires, dont chacun définit une seule ligne horizontale du caractère. En fait, seuls les six bits les plus bas du modèle sont utilisés pour que chaque ligne concorde avec le nombre de points lumineux affichables dans chaque emplacement. Cela vous aidera à comprendre, comment ces configurations se combinent pour définir un caractère, si vous pensez à chaque caractère comme étant une grille de six blocs sur huit.



Dans le dessin ci-dessus, les nombres figurant au-dessus de chaque bloc représentent les valeurs des bits équivalents dans la configuration. Les chiffres sur le côté du bloc représentant la somme des bits de la rangée sur laquelle ils se trouvent (désignés en hachuré). C'est ainsi que les caractères normaux de l'ORIC sont définis. Dans le mode **HIRES**, cette information est utilisée pour décider des points à afficher pour produire un caractère. Dans les modes **TEXT**, cette information est codifiée dans le signal de télévision, pour produire le caractère sur l'affichage à la position convenable du caractère. Contrairement à la plupart des ordinateurs, l'ORIC garde l'ensemble de son tableau de définition de caractères en RAM (mémoire vive). Cela signifie que nous pouvons changer n'importe quel caractère en ce que nous désirons. Ceci représente une possibilité énorme pour les jeux et les affichages animés.

Lorsque vous branchez votre ORIC, ou appuyez sur le bouton de réinitialisation (RESET), l'ordinateur recopie les ensembles de caractères, standards ou alternés, à partir de sa mémoire permanente (ROM) sur la mémoire vive (RAM). Les tableaux comportent 128*8 données pour les définitions de l'ensemble standard, et 112*8 pour l'ensemble alterné. Les tableaux sont toujours gardés en mémoire, juste avant la mémoire d'écran. Lorsque nous passons de **TEXT** à **HIRES**, ces tableaux sont déplacés pour conserver leur position juste avant la mémoire d'écran. En mode **TEXT**, ils commencent à l'emplacement # B400 (pour l'ensemble standard) et à l'emplacement # B800 (pour l'ensemble alterné). En mode **HIRES**, ces tableaux sont déplacés pour commencer à # 9800 pour l'ensemble standard et # 9C00 pour l'ensemble alterné. La position de la première série de huit données dans le tableau, pour définir un caractère, peut être trouvée en ajoutant le produit de 8 par le code ASCII du caractère en question, à l'adresse du début du tableau. Donc, la donnée pour A est conservée à # B400 + 8*65 (ASCII(A)) qui est (en décimal) 46600. Essayez le programme suivant qui introduit **(POKE)** de nouvelles valeurs dans ces emplacements et ensuite affiche un A. Le A doit apparaître comme un *A* italique, parce que nous avons maintenant redéfini la manière dont ORIC dessine un A. Essayez de passer au mode **HIRES** et utilisez **CHAR** pour placer un A sur l'écran, pour vous assurer que les caractères ont été bien recopiés.

```
10 FOR A=0 TO 7
20 READ B
30 POKE 46600+A,B
40 NEXT
50 DATA 2,6,10,14,18,18,34,0
```


15/08/2000
Page 55 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

Pour tirer tout l'avantage de cette possibilité de changer la forme des lettres, nous aurons besoin de calculer de nombreuses représentations différentes en binaire. Les ordinateurs sont supposés nous simplifier la vie, alors utilisons l'ORIC pour tout ce qui est ardu. Le listing suivant est un programme de générateur de caractères qui nous permet de définir n'importe quelle forme que vous désirez, et en conserve les données dans le tableau de caractères de votre choix. Le caractère est affiché à grande échelle à l'intérieur d'une grille et nous utilisons le carré plein de l'ensemble alterné pour remplir les cases. Les cases sont également tracées en taille réelle sur l'écran, de même pour les points en dessous de la grille, et de cette façon nous pouvons lire directement l'équivalent binaire de l'écran de mémoire (en se souvenant que les bits 6 et 7 ne font pas partie de la définition). Pour déplacer le curseur, utilisez les touches fléchées, et pour changer une case appuyez la barre à intervales. Pour Sauvegarder un caractère appuyez sur S, pour Editer un caractère appuyez sur E, et pour arrêter le programme appuyez sur Q. Les options S et E vous demanderont de préciser quel ensemble de caractères (Ø pour standard, et 1 pour alternatif) et quel caractère (le code ASCII) vous désirez sauvegarder ou éditer ainsi que où et d'où. Si vous désirez utiliser la valeur dans un autre programme, vous pourrez la lire sur l'écran où elle est affichée à droite de la grille, lorsqu'un caractère est à l'étude.

```
10 HIMEM #97FF
20 GOSUB 200
30 REPEAT
40 CURSET XC,YC,3:WAIT 10:CHAR 95,1,2
50 WAIT 10:CHAR 95,1,2
60 A$=KEY$
70 UNTIL A$<>""
80 IF A$=CHR$(9) AND XX<6 THEN XX=XX+1
   :XC=XC+6:GOTO 30
90 IF A$=CHR$(8) AND XX>1 THEN XX=XX-1
   :XC=XC-6:GOTO 30
100 IF A$=CHR$(10)AND YY<8 THEN YY=YY+
    1:YC=YC+8:GOTO 30
110 IF A$=CHR$(11)AND YY>1 THEN YY=YY-
    1:YC=YC-8:GOTO 30
120 IF A$=" " THEN GOSUB 300:GOTO 30
130 IF A$="E" THEN GOSUB 400:GOTO 30
140 IF A$="S" THEN GOSUB 500:GOTO 30
150 IF A$<>"Q"THEN GOTO 30
160 STOP
```



```
200 REM...INITIALISE L'AFFICHAGE...
210 HIRES
220 FOR X=100 TO 136 STEP 6:CURSET X,3
    0,1:DRAW 0,64,1:NEXT
230 FOR Y=30  TO 94  STEP 8:CURSET 100
,Y,1:DRAW 36,0,1:NEXT
240 X=119:Y=100:M=#A000+Y*40+INT(X/6)+
    1
250 XC=100:YC=30:XX=1:YY=1
260 GOSUB 900
270 RETURN
300 REM.MODIFICATION D'UN CARRE..
310 CURSET XC,YC,3: CHAR 95,1,2
320 CURSET X+XX,Y+YY,2
330 FOR I=1 TO 8
340 P=PEEK(M+40*I):IF P>63 THEN P=P-6
    4
350 P$=MID$(STR$(P),2):P1=180:P2=I*8+2
    4:GOSUB 600
360 NEXT
370 RETURN
400 REM...EDITION D'UN CARACTERE...
410 GOSUB 700
420 GOSUB 200
430 CURSET X+1,Y+1,3
440 CHAR A,S,1
450 FOR J=X+1 TO X+6
460 FOR K=Y+1 TO Y+8
470 IF POINT(J,K) THEN GOSUB 800
480 NEXT:NEXT
490 GOTO 330
500 REM ..SAUVEGARDE D'UN CARACTERE.
510 GOSUB 700
520 FOR I=1 TO 8
530 P+PEEK(M+40*I):IF P>63 THEN P=P-64
540 POKE(#97FF+S*#400+*8+1),P
550 NEXT
560 GOSUB 200
570 RETURN
600 REM...AFFICHAGE D'UNE LIGNE...
610 CURSET P1,P2,3:FILL 8,3,64
620 CURSET P1,P2,3
630 FOR K=1 TO LEN(P$)
640 CHAR ASC(MID$(P$,K)),P3,1
650 DRAW 6,0,3
660 NEXT
670 RETURN
700 REM....CODE ASCII.......
```


15/08/2000
Page 56 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

```
710 REPEAT
720 INPUT"QUEL ENSEMBLE";S
730 UNTIL S=1 OR S=0
740 REPEAT
750 INPUT"CODE ASCII";A
760 UNTIL A>32 AND A<128
770 RETURN
800 REM.CARRE J,K DANS LA GRILLE.
810 TX=100+6*(J-X-1)
820 TY=30 +8*(K-Y-1)
830 CURSET TX,TY,3
840 CHAR95,1,2
850 RETURN
900 REM  ..AFFICHAGE DES CARACTERES
910 FOR L=32 TO 96 STEP 32:P$=""
920 FOR N=L TO L+31
930 P$=P$+CHR$(N)
940 NEXT
950 P1=20:P2=115+INT(L/3):P3=0
960 GOSUB 600
970 P1=20:P2=155+INT(L/3):P3=1
980 GOSUB 600
990 NEXT
1000 P3=0:RETURN
```

Le dernier programme de ce chapitre, explique l'utilisation d'un caractère défini dans un simple programme d'animation d'un avion suivant un itinéraire précisé en coordonnées polaires. Cela devrait vous donner une idée du pouvoir des caractères définis de votre ORIC et fournir une fin appropriée à ce chapitre qui examine la prouesse du graphisme de l'ATMOS.

```
10 HIMEM #97FF
20 FOR P=0 TO 7
30 M=#B400+8*(ASC("a")+P)
40 FOR A=0 TO 7
50 READ B
60 POKE M+A,B
70 NEXT
80 NEXT
90 REM...DATA POUR a,b,c,d,e,f,g,h
100 DATA 30,30,6,6,38,39,38,0
110 DATA 2,3,18,38,12,24,48,40
120 DATA 0,57,1,63,63,16,0,0
130 DATA 16,8,36,48,24,45,7,2
```



```
140 DATA 0,38,39,38,6,6,30,30
150 DATA 2,4,9,3,6,45,56,16
160 DATA 0,39,32,63,63,2,0,0
170 DATA 16,48,18,25,12,6,3,5
200 HIRES :PAPER 6:INK1
210 OX=180:OY=101
220 R=80:PP=PI/100
230 REPEAT
240 FOR A=0 TO 2*PI STEP PP
250 X=120+COS(A)*R
260 Y=100+SIN(A)*R
270 C=97+INT((A+PI/8)/(PI/4))
280 IF C=105 THEN C=97
290 CURSET OX,OY,3:FILL 8,2,64
300 CURSET X,Y,3:CHAR C,0,1
310 OX=X:OY=Y
320 NEXT
330 UNTIL KEY$<>""
```


William Saint-Cricq / wsaintcr@wanadoo.fr Page 57 sur 163 15/08/2000

Numérisation http://wsc.n3.net

Chapitre 8
Musique et sons

Les possibilités de l'ORIC en ce qui concerne le son et la musique sont considérables. Pour les amateurs de jeu qui recherchent des effets sonores d'emploi facile, l'ORIC apporte les instructions toutes faites : **ZAP** (sifflement) **SHOOT** (coup de feu), **PING** (clochette) ou **EXPLODE** (explosion). Ces commandes produisent exactement le son recherché, et de plus, ce son est émis à un niveau sonore impressionnant. L'ORIC émet ses bruits et ses sons avec une voix forte.

La souplesse et le raffinement des autres commandes sonores de l'ORIC sont aussi dignes d'intérêt :

MUSIC est une commande qui fournit à l'utilisateur une gamme chromatique de type occidental avec laquelle on peut composer. Pour les effets sonores on trouve la commande **SOUND** qui contrôle le générateur de bruit, utile pour simuler des fusées vrombissantes et des explosions passionnantes. Ce chapitre nous entraîne au fond de la musique sur micro-ordinateur. Notre premier arrêt dans ce voyage sera pour les commandes sonores toutes faites.

On trouve quatre instructions BASIC pré-programmées : **ZAP, SHOOT, PING** et **EXPLODE.** Simplement en tapant **ZAP** (et en appuyant sur la touche **RETURN),** le son d'extra-terrestres tirant une salve de leur canon-laser est immédiatement produit. Au contraire, ce programme d'une ligne :

```
10 FOR N=1 TO 100 : SHOOT : WAIT 30 : NEXT
```

vous donnera vraiment l'impression d'une soirée mouvementée à Dodger City. Ces commandes sont idéales pour enrichir les jeux, et peuvent être utilisés à bon escient dans vos futurs Envahisseurs de l'espace, Chasseurs de Dahus ou Agents Secrets.

Les sons préréglés, simples et d'un emploi commode ne sont qu'un préambule aux possibilités de l'ORIC, cependant, et pour une utilisation créatrice des sons, nous nous tournerons vers le trio bruyant **SOUND, MUSIC** et **PLAY.** Dans ce chapitre, nous expliquerons les effets et les écritures de ces instructions et expliquerons comment **PLAY** se combine avec les deux autres. Plongeons dans le monde fascinant de la composition musicale sur micro-ordinateur en étudiant la souplesse de la commande **MUSIC.**

William Saint-Cricq / wsaintcr@wanadoo.fr Page 58 sur 163 15/08/2000

Numérisation http://wsc.n3.net

MUSIC

MUSIC utilise les trois générateurs d'onde rectangulaire disponibles d'une puce spécialiste ès-sons comme source sonore. Chaque son a un canal séparé, et en utilisant **MUSIC** combiné avec **PLAY**, les accords et les morceaux polyphoniques sont possibles. Pour plus de simplicité, nous traiterons du canal 1 seulement, car il fonctionne sans utiliser **PLAY**.

La syntaxe de la commande **MUSIC** est

MUSIC Canal, Octave, Note, Volume

Les quatres nombres désignant le canal, l'octave, la note et le volume doivent être séparés par des virgules. Le canal peut prendre la valeur 1, 2 ou 3, et définit quelle générateur fonctionne. Pour le reste de ce chapitre, nous utiliserons le canal 1.

L'Octave désigne dans quel intervalle le paramètre Note intervient. Ce doit être une valeur entière entre 0 et 6, 2 désignant l'octave commençant par le Do du milieu du piano. Le but de la désignation de l'octave sera plus claire après l'étude du paramètre Note.

Note a une amplitude de valeurs entières entre 1 et 12 inclus. Ce sont les équivalents des notes sur une échelle chromatique partant de Do (N = 1) et finissant en Si (N = 12). En combinant octave et note, on obtient toutes les notes entre Do, deux octaves plus bas que le Do du milieu du piano, et Si quatre octave au-dessus.

Volume s'explique de lui-même, et prend des valeurs entre 1 (silencieux) et 15 (brise-tympans).

Comme rapide démonstration de **MUSIC**, tapez et effectuez le programme **"GAMME CHROMATIQUE"** qui joue toutes les notes des sept octaves, c'est-à-dire de **MUSIC 1, 0, 1, 10** (Do grave) à **MUSIC 1, 6, 12, 10** (Si aigu)

```
1 REM ...GAMME CHROMATIQUE...
5 CLS
10 PAPER 5
20 FOR O=0 TO 6
30 FOR N=1 TO 12
40 MUSIC 1,O,N,10
50 WAIT 30
60 NEXT N
70 NEXT O
75 END
```

William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Numérisation http://wsc.n3.net

L'élégance de la commande **MUSIC** devient réellement évidente quand vous établissez une relation entre la musique du monde réel et la musique sur ordinateur. La façon dont l'octave et la note se combinent est le reflet exact de la thèse occidentale de musique. Pour illustrer cela, jetez un coup d'œil sur ce diagramme d'un clavier ordinaire de piano. Tout comme dans la notation habituelle où 12 symboles désignent les 12 notes de la gamme, l'ORIC associe 12 valeurs aux notes. Les notes ne peuvent être 13 ou 14, puisqu'il n'y a que 12 notes dans la gamme.

Pour jouer les notes de la gamme majeure de Do, (c'est-à-dire les notes blanches du piano de Do à Do) vous utiliserez

Note = 1, 3, 5, 6, 8, 1Ø, 12, 1

Le programme "CLAVIER" montre ce principe en utilisant la ligne supérieure de l'ORIC comme un clavier de piano (Ø devient 1Ø, "—" devient 11 et "=" devient 12) :

1	2	3	4	5	6	7	8	9	Ø	—	=
Do	Do # Ré b	Ré	Ré # Mi b	Mi	Fa	Fa # Sol b	Sol	Sol # La b	La	La # Si b	Si

```
10 REM.....CLAVIER.......
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,8
85 WAIT 20 :PLAY 0,0,0,0
90 GOTO 20
```

GETA$ attend qu'une touche soit enfoncée, cependant que la ligne 3Ø prend en compte la valeur de la touche enfoncée. A est la valeur de cette touche, sauf pour **Ø**, —, =, qui donnent respectivement 1Ø, 11, 12. L'instruction **PLAY** est utilisée dans ce programme : **PLAY Ø, Ø, Ø, Ø,** simplement pour déconnecter le canal 1 quand le programme est terminé. Enfoncez la touche "/" pour arrêter le programme.

En raison des limites du clavier de l'ORIC, vous ne pourrez sans doute pas jouer "le Vol du Bourdon" avec ce programme. Cependant, il est possible d'acquérir une excellente technique sans bouger le moindre doigt (ah, il faut quand même enfoncer la touche **RETURN** !). Pour ce faire, utilisons l'ordinateur dans ce qu'il connaît le mieux, se souvenir. On peut conserver des informations

15/08/2000

musicales dans l'ORIC de deux façons différentes : la première manière est d'utiliser l'instruction de données **DATA** :

```
1 REM....SONS...........
5 READ A
10 MUSIC 1,3,A,10
20 IF A= 11 THEN RESTORE
30 WAIT 25
40 GOTO 5
50 FOR Y=B*20 TO B*20 +20
60 P=#A000+Y*40+A*5
70 IF INT (Y/2)=Y/2 THEN Q= 16+A ELSE
   Q=16+B
80 POKE P,Q
90 NEXT
100 DATA 3,5,6,8,10,6,10,10,9,5,9,9,8,
    4,8,11
110 NEXT
```

Le programme **NOTES** utilise l'instruction **DATA** pour stocker les valeurs des notes (A). Dans la ligne 5, les valeurs sont lues **(READ)** puis jouées dans l'enoncé de **MUSIC**. La ligne 25 sert simplement à répéter le programme automatiquement.

L'énoncé en **DATA** présente l'avantage d'être économe avec l'espace-mémoire, et est donc recommandé si on a besoin d'une fanfare de ce type dans un programme de jeux. Son inconvénient tient dans la difficulté rencontrée pour changer les données. Ce sont souvent temps perdu et essais infructueux pour éditer la mélodie. Pour une utilisation plus simple des données, nous devons voir la seconde méthode possible de stockage des données musicales : dans un tableau.

Le programme ci-dessous utilise le clavier pour entrer les valeurs des notes dans un tableau de 1ØØ notes A(1ØØ). La ligne 13 crée le tableau et la partie suivante du programme est le vieux programme "CLAVIER". La ligne 85 utilise **WAIT 2Ø**, suivie de **PLAY Ø, Ø, Ø, Ø**, pour donner à chaque note une longueur constante de 1/5ᵉ de seconde (sans cela, la note continuerait tant que la touche est enfoncée). La ligne 75 nous permet d'abandonner l'entrée des données quand le nombre de notes est suffisant, et la variable G compte le nombre de termes de la suite.

```
2 REM ....SEQUENCER .........
4 CLS
5 PRINT"Un sequencer de 100 notes"
6 PRINT"========================="
```


Page 60 sur 163

William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

```
7 PRINT
8 PRINT"Entrez les notes avec la ligne
      superieure du clavier"
9 PRINT"Apres la derniere note,appuyez
sur /"
10 CLEAR
12 DIMA(100)
14 G=1
16 FOR N=1 TO 100
20 GETA$
30 A(N)=VAL(A$)
40 IF A$="-"THEN A(N)=11
50 IF A$="="THEN A(N)=12
60 IF A$="0"THEN A(N)=10
70 IF A$="/"THEN 400
75 IF A(N)=0 THEN 20
80 MUSIC 1,3,A(N),8
85 WAIT 20:PLAY 0,0,0,0
87 G=G+1
90 NEXT N
400 CLS
405 PRINT"Fixez la vitesse d'execution
  "
410 INPUT S
411 PRINT
412 PRINT"Depart......................:
   tapez D"
413 PRINT"Stop........................:
   tapez S"
414 PRINT"Entree d'une autre sequence:
   tapez E"
415 PRINT"Autre vitesse d'execution..:
   tapez A"
430 GET A$
440 IF A$="D" THEN 500
450 IF A$="E" THEN RUN
460 IF A$="A" THEN 400
500 FOR N=1 TO G
505 IF N=G THEN N=1
510 MUSIC 1,3,A(N),10
512 PLOT 10,10,"No.  :"+STR$(N)
513 PLOT 10,12,"Note:"+STR$(A(N))
520 WAIT S
521 IF KEY$="S" THEN 430
522 PLOT 10,10,"No.  :          "
523 PLOT 10,12,"Note:           "
```


La plupart du reste du programme est précisé par les instructions et l'explication dans le texte pour lancer et arrêter la suite de notes, sauf en ce qui concerne les lignes **500** à 530.

A la ligne **500** nous voyons que l'utilisation de G permet de reproduire la séquence indéfiniment, puisque N est remis à 1 lorsque le cycle est achevé, c'est-à-dire lorsque N = G. La vitesse d'exécution de la boucle **FOR...NEXT** est contrôlée par la ligne **520 WAIT S**.

WAIT est une commande importante pour la musique : la durée des notes est aussi importante que la hauteur des notes, aussi dans un programme musical sérieux, nous devrons prendre en compte la durée des notes. Dans ce programme nous pourrons rendre les notes plus longues simplement en les entrant plus d'une fois. Cette methode est utilisée dans les synthétiseurs du commerce, tels le Roland 81101. Ce type de travail répétitif convient parfaitement pour la musique répétitive ou mécanique, mais ne peut être utilisée lorsque l'on doit jouer des variations.

Voici un programme permettant une composition musicale plus subtile, qui permet de conserver à la fois la hauteur et la longueur de la note. Étudiez ce programme : **"COMPOSITION"**.

```
1 REM .....COMPOSITION MUSICALE...
10 T=1
20 CLS
30 DIM L(50)
40 DIM O(50)
50 DIM A(50)
55 PRINT"POUR JOUER VOTRE AIR,TAPEZ 0
  COMME    VALEUR DE NOTE"
60 FOR N=1 TO 50
90 PRINT
100 INPUT"NOTE   entre 1 et 12";A(N)
110 IF A(N)=0 THEN 500
120 INPUT"OCTAVE entre 1 et 7 ";O(N)
125 MUSIC 1,O(N),A(N),10:WAIT 150:PLAY
    0,0,0,0
130 INPUT"LONGUEUR ";L(N)
150 NEXT
500 PLAY 1,0,0,0
504 T=T+1
506 IF T=6 THEN PLAY 0,0,0,0:STOP
510 FOR P=1 TO N-1
520 MUSIC 1,O(P),A(P),10
530 WAIT 20*L(P)
535 IF P=N-1 THEN 500
540 NEXT P
```


William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Numérisation http://wsc.n3.net

Ce programme utilise trois tableaux A( ), O( ) et L( ) pour conserver les notes, l'octave et la longueur. Les notes sont entrées par **INPUT**, ce qui fait que la touche **RETURN** doit être enfoncée. Pour sortir de l'entrée des données et jouer la mélodie, le programme comporte la ligne :

102 IF A(N) = 0 THEN GOTO 500

Il faut entrer 0 comme réponse à **NOTE** ? pour cesser d'entrer les données.

Cette méthode pour entrer la hauteur et la longueur de la note se rapproche des méthodes utilisées par certains synthétiseurs de type professionnel. Notez aussi que l'ORIC permet de jouer en polyphonie, c'est-à-dire qu'il peut jouer plusieurs notes à la fois.

PLAY

La commande **PLAY** a deux fonctions distinctes : la première est de mettre en service ou hors service le canal sonore correspondant à l'un des trois générateurs de son ou de bruit, qui peuvent être utilisés simultanément. La seconde est de déterminer la forme de l'enveloppe du son. La syntaxe de la commande est la suivante :

PLAY Canal Son, Canal Bruit, Enveloppe, Période

La commande **PLAY** permet la combinaison de trois canaux pour le son ou le bruit, utilisant les valeurs 1 à 7 pour "canal son" ou "canal bruit"

0 : Ni son ni bruit dans aucun canal
1 : canal 1 en service
2 : canal 2 en service
3 : canaux 1 et 2 en service
4 : canal 3 en service
5 : canaux 3 et 1 en service
6 : canaux 3 et 2 en service
7 : canaux 3, 2 et 1 en service.

Nous avons jusqu'à présent utilisé **MUSIC** sans **PLAY** dans nos précédents programmes. Cela tient au fait que le canal 1 est un cas spécial, et fonctionne tant que la commande **PLAY 0, 0, 0, 0** n'a pas été donnée. Cela ne s'applique pas aux canaux 2 ou 3, pas plus qu'aux canaux de bruits. Dans tous nos programmes, désormais, la commande **PLAY** est essentielle pour que le programme fonctionne. Comme exemple de **PLAY**, faites tourner le programme "**ACCORD**" qui joue un accord de Do-majeur de trois notes jusqu'à ce que vous coupiez le son par **CTRL C**.

```
90 REM ....ACCORD DE TROIS NOTES ...
95 PLAY 7,0,0,0
100 MUSIC 1,4,1,5
110 MUSIC 2,4,5,5
120 MUSIC 3,4,8,5
140 STOP
```

Le programme utilise **PLAY** ainsi :

PLAY 7, 0, 0, 0

les trois canaux "son" sont en service, les trois canaux "bruit" sont hors service. On peut réduire l'accord à deux notes en changeant la commande **PLAY** afin que deux canaux seulement fonctionnent :

Canaux
(1 + 2) Play 3, 0, 0, 0
(3 + 2) Play 6, 0, 0, 0
(3 + 1) Play 5, 0, 0, 0

Un autre exemple de l'utilisation de **PLAY** est "ANDROIDES", un morceau musical composé spécialement pour ce livre. **PLAY** 3, 0, 0, 0 est utilisé, ainsi que deux instructions **MUSIC**, créant ainsi un duo entre les sources sonores 1 et 2.

```
5 REM    ANDROIDES
10 A=1
50 REPEAT
60 A=A+1
90 READ X,Y
95 PLAY 7,0,0,0
100 PLAY 3,0,0,0
110 MUSIC 1,1,X,15
120 MUSIC 2,4,Y,12
140 WAIT 15
150 UNTIL A=192
160 RESTORE
170 GOTO 10
1000 DATA 1,8,8,8,1,8,8,8,1,8,8,8,1,8,
     8,8,1,8,8,8,1,8,8,8,3,6,10,5,3,3,10,3
1010 DATA 1,10,8,10,1,10,8,10,1,8,8,8,
     1,8,8,8,1,8,8,8,1,8,8,8
1020 DATA 3,3,10,5,3,6,10,6,1,10,8,10,
```


15/08/2000
Page 62 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

```
1,10,8,10
 1030 DATA 1,8,8,8,1,8,8,8,1,8,8,8,1,8,
8,8,3,3,10,5,3,3,10,3
 1040 DATA 1,1,8,1,1,1,8,1,1,1,8,1,1,1,
8,1,1,1,8,1,1,1,8,1,1,1,8,1,1,1,8,1
 1050 DATA 1,8,8,8,1,8,8,8,1,8,8,8,1,8,
8,1,8,8,8,1,8,8,8,3,6,10,5,3,3,10,3
 1060 DATA 1,10,8,10,1,10,8,10,1,8,8,8,
1,8,8,8,1,8,8,8,1,8,8,8
 1070 DATA 3,3,10,5,3,6,10,6,1,10,8,10,
1,10,8,10
 1080 DATA 1,8,8,8,1,8,8,8,1,8,8,8,1,8,
8,8,3,3,10,5,3,3,10,3
 1090 DATA 1,1,8,1,1,1,8,1,1,1,8,1,1,1,
8,1,1,1,8,1,1,1,8,1,1,1,8,1,1,1,8,1
 1095 DATA 3,3,10,3,3,3,10,3,5,5,12,5,5
,5,12,5
 2000 DATA 6,6,1,6,6,6,1,6,1,1,8,1,1,1,
8,1
 2010 DATA 3,3,10,3,3,3,10,3,5,5,12,5,5
,5,12,5
 2020 DATA 6,6,1,6,6,6,1,6,1,1,8,1,1,1,
8,1
 2030 DATA 3,3,10,3,3,3,10,3,5,5,12,5,5
,5,12,5
 2040 DATA 6,6,1,6,6,6,1,6,1,1,8,1,1,1,
8,1
 2050 DATA 3,3,10,3,3,3,10,3,5,5,12,5,5
,5,12,5
 2060 DATA 6,6,1,6,6,6,1,6,8,8,3,8,8,8,
3,8
```

Ceci est un autre exemple de l'utilisation de l'instruction **DATA** pour conserver des informations musicales. Dans ce cas, **DATA** consiste en un couple de nombres, le premier est la basse, le second la mélodie.

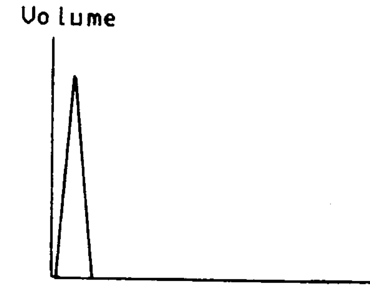
Remarquez que les lignes **1000** à **1040** sont identiques au lignes **1050** à **1095**, ainsi vous pourrez vous épargner du temps et vous dispenser de les taper en utilisant **CTRL A**, qui permet de recopier les lignes. De même, les lignes **1090** et **2000** peuvent être recopiées pour donner les lignes **2010**, **2020**, et **2030**, **2040**.

Vous pouvez rencontrer d'autres problèmes avec les **DATA** : une erreur de copie, et ce thème pour un western spatial devient un morceau de jazz d'avant-garde !

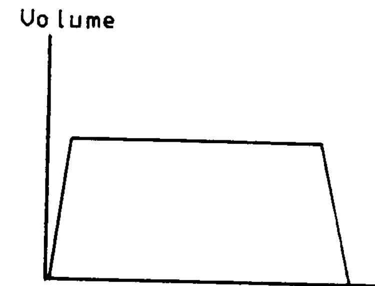
Revenons à **PLAY** : vous vous êtes sûrement demandé ce que signifiaient les deux derniers paramètres, Enveloppe et Période. Ceux d'entre vous familiarisés avec le synthétiseur, ou férus d'acoustique, sont habitués au concept d'enveloppe sonore, mais

pour les autres, nous allons en quelques lignes vous expliquer de quoi il s'agit. L'enveloppe d'un son est simplement le graphe du volume en fonction du temps. Comme exemples, nous prendrons le son d'un tambour contre, disons, James Galway jouant de la flûte.

coup de tambour



flûte



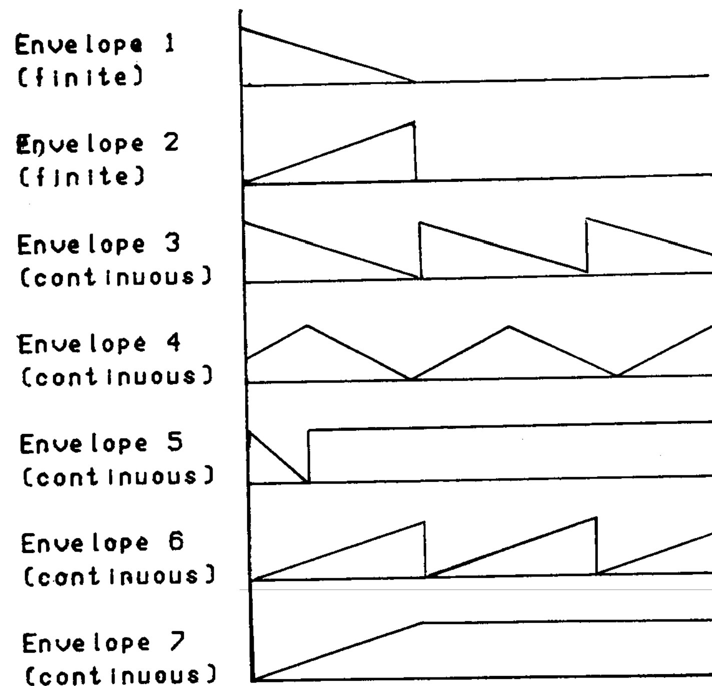
Les instruments à percussion, tels le tambour et les marimbas ont une attaque rapide, c'est-à-dire qu'ils atteignent rapidement le niveau sonore maximum, puis déclinent rapidement tandis que la flûte, le violon, etc. ont un attaque plus douce et maintiennent leur volume pendant un temps plus ou moins long (jusqu'à ce que M. James Galway manque de souffle, par exemple).

L'ORIC possède un arsenal d'enveloppes dans lequel nous pourrons choisir, sept en tout, en souhaitant qu'elles couvrent toutes les occasions. Même si ce n'est pas entièrement vrai, le système est certainement plus simple à utiliser qu'un système de définition complète de l'enveloppe.

William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Les graphes de chaque enveloppe ont les formes données ci-dessous. Notez que les formes 1 et 2 sont limitées (ont un point d'arrêt), cependant que les formes 3 à 7 sont continues, et joueront la note tant qu'elle n'est pas arrêtée. L'enveloppe Ø existe, mais c'est exactement la même que l'enveloppe 1 montrée ci-dessous.



La période peut prendre des valeurs entre Ø et 32767, et contrôle la durée du son ou du bruit dans le cas d'enveloppes limitées. Pour les enveloppes continues, la période contrôle la longueur de l'onde, et le mieux pour comprendre cela est d'étudier le programme "Essai d'enveloppes" et d'essayer différentes valeurs de la période.

```
5 REM.... ESAIS D'ENVELOPPES .......
8 CLS
10 INPUT"QUEL CANAL (1,2 ou 3)";C
20 IF C<1 OR C>3 THEN 10
30 INPUT "QUEL TYPE D'ENVELOPPE (1 A 7
)";E
```



```
40 IF E<1 OR E>7 THEN 30
50 INPUT "QUELLE PERIODE OU DUREE (0 A
32767)";T
60 IF T<0 OR T> 32767 THEN 50
70 CLS
80 PRINT"CANAL................";C
90 PRINT"TYPE D'ENVELOPPE......";E
100 PRINT"PERIODE OU DUREE......";T
110 MUSIC C,3,4,0
120 PLAY C,0,E,T
130 PRINT"Tapez 'RETURN' si le son per
siste"
140 GOTO 5
```

Dans la ligne 11Ø du programme, vous remarquerez que la valeur du volume est ramenée à Ø, avec **MUSIC T, 3, 4, Ø**. Donner à cette variable la valeur Ø permet de donner à l'instruction **PLAY** le contrôle de l'enveloppe. Toute autre valeur ferait que le paramètre enveloppe ne serait pas pris en compte.

Pour l'enveloppe de type 1, une période de 1ØØØ donnera un effet brusque du genre banjo, tandis qu'une période de 327ØØ donnera une attaque nette puis un son déclinant lentement. Pour l'enveloppe de type 6, une valeur de période égale à 19 donne un effet mécanique intéressant. Des heures d'amusement pour toute la famille, voilà ce que réserve la recherche des sons les plus fous avec le programme "Essai d'enveloppes".

Un autre usage important de la commande **PLAY** que vous devez garder à portée de la main est **PLAY Ø, Ø, Ø, Ø**, surtout si votre magnifique composition musicale se termine en un ennuyeux gémissement que vous ne pouvez arrêter. Plutôt que d'utiliser la touche d'initialisation ou de débrancher l'alimentation, essayez ce qui précède. Un autre point à garder à l'esprit est que le déclic du clavier arrêtera normalement une note persistante, mais peut aussi perturber votre programme, alors souvenez-vous qu'il peut être supprimé (et rétabli) en utilisant **CTRL F**.

SOUND

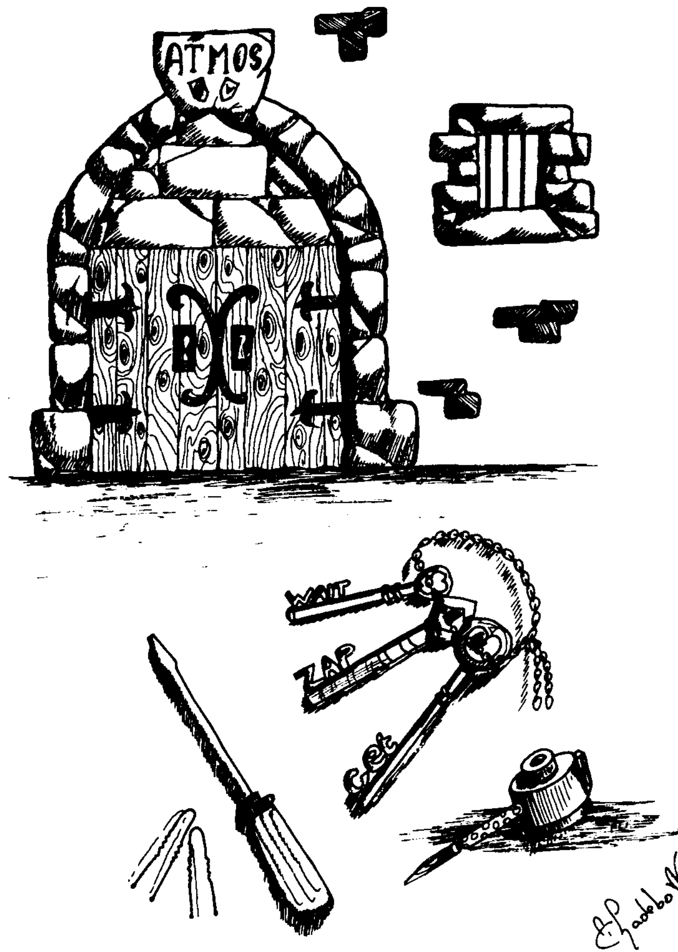
SOUND est fondamentalement une version moins élaborée de **MUSIC**, où les sons des canaux ne sont pas aménagés en demi-tons. Cette instruction comporte en plus cependant un générateur de bruit, permettant la réalisation idéale d'effets sonores. L'écriture de la commande **SOUND** est la suivante :

SOUND canal, hauteur du son, volume

Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr Page 64 sur 163 15/08/2000

Numérisation http://wsc.n3.net



Chapitre 9
Les mots clés du BASIC

Dans cette partie, chacun des mots-clés du BASIC est étudié avec un exemple de son utilisation. Les informations sont données avec l'écriture correspondant à chaque commande, en utilisant les symboles suivants :

v nom de variable numérique
v$ nom de variable chaîne
i,j nombres entiers
n nombre décimal (qui peut aussi être un entier), ou résultat d'une expression numérique
c expression conditionnelle logique (comme A>B ou **TRUE**)
a$ chaîne de caractère
ln : numéro de ligne de programme
adr adresse d'un emplacement de mémoire

Pour rendre les commandes graphiques plus claires, nous utilisons aussi les symboles suivants :

fb paramètre "premier plan/arrière plan" (Ø – 3). Voir chapitre 7 pour plus de détails.
s caractères
x,y coordonées d'un point de l'écran.

D'autres symboles pourront être utilisés si nécessaire. Ils seront définis dans le cours de la définition du mot-clé. Le code BASIC donné est la valeur décimale que l'ORIC utilise pour stocker les mots-clés du BASIC en octets dans la mémoire.

ABS

Code BASIC : 216
écriture : ABS (n)

Renvoie la valeur absolue du nombre ou de l'expression entre paranthèses. Par exemple **ABS** (– 19) est 19. Cette fonction peut être utilisée quand il est nécessaire d'obtenir un résultat positif lors d'un calcul sur deux variables. Par exemple, dans le calcul de **A – B**,

15/08/2000
Page 65 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

La valeur du **canal** peut être choisie entre 1 et 6 : 1, 2 et 3 font référence aux **trois générateurs** de son (ainsi que dans l'instruction **MUSIC**). Les **valeurs 4, 5, 6** précisent que le générateur de bruit est mêlé avec. Il **n'y a qu'un** générateur de bruit, qui produira du bruit par les canaux **1, 2 ou 3** selon que la valeur 4, 5 ou 6 est spécifiée.

La hauteur du son ou du bruit désirée (c'est-à-dire la fréquence) n'est pas, comme pour la musique, préprogrammé en demi-tons, ce qui est moins pratique pour des recherches musicales. La valeur peut être prise entre **Ø** et 65536, en sachant que 65536 est très grave, et qu'une valeur inférieure à 5 fait hurler les chiens. Le canal de bruit peut aussi être choisi quand la hauteur varie. C'est une astuce inhabituelle mais utile, que montre le programme **"VAGUES"**

```
180 REM.....VAGUES.....
200 PLAY0,1,0,0
205 Z%=RND(1)*20
210 FOR I=0 TO 31
220 SOUND 4,I,7
230 WAIT Z%
240 NEXT I
250 GOTO 205
```

Comme vous pourrez le constater, le bruit donne l'impression de changer de hauteur (il n'est pas mélangé au son). Quand il s'agit d'un bruit, la hauteur peut prendre les valeurs Ø à 31, ou tout multiple de ces valeurs (32-63, 64-95, etc.) pour un "glissement" complet. Des programmes de ce type sont particulièrement intéressants s'ils sont associés au graphiques mais nous vous laissons le soin d'évoquer des images de vagues se brisant sur littoral !

Le dernier paramètre de **SOUND** est, bien entendu, le volume. Le volume est identique à son homologue sous l'instruction **MUSIC**, et peut prendre des valeurs entières de 1 à 15. Comme précédemment, la valeur Ø laisse le contrôle de l'enveloppe à l'instruction **PLAY**. Tout comme pour **MUSIC**, **SOUND** doit être utilisé avec **PLAY**, sauf si le canal 1 est seul utilisé. Dans **VAGUES**, le canal 1 est choisi en utilisant **PLAY Ø, 1, Ø, Ø**, qui déconnecte le générateur de son du canal 1, et utilise ce canal 1 pour le bruit. Le bruit est choisi dans l'instruction **SOUND** en donnant au paramètre de canal la valeur 4 **(SOUND 4, I, 7)**. Fixer à zéro le volume permet d'utiliser les sept enveloppes comme indiqué dans **MUSIC**.

Un autre possibilité de **SOUND** avec **PLAY** est la production de plusieurs bruits simultanés. Cela ne donne pas des accords comme

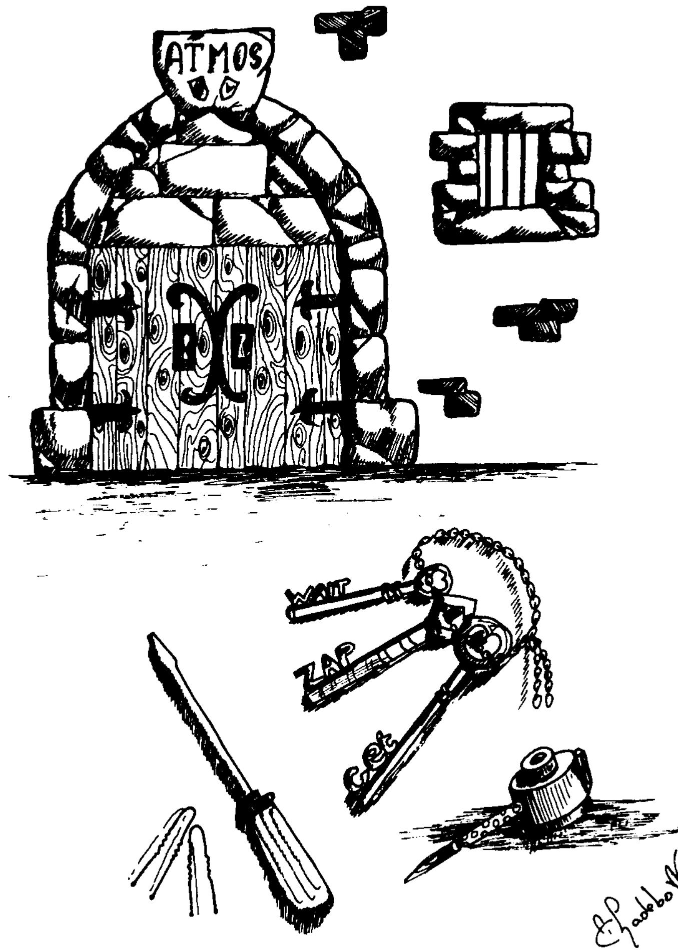
en musique, mais peut être utilisé pour créer des effets complexes, tels que dans ce programme :

```
10 REM....TOUS AUX ABRIS !.....
20 FOR P=31 TO 1 STEP-1
30 SOUND 2,P+100,15
40 SOUND 1,P*10,15
50 SOUND 3,P*2+6,10
60 SOUND 4,P,15
70 PLAY 7,1,0,0
80 WAIT 50
90 IF P=1 THEN FORI=1 TO 5:EXPLODE:WAIT 50:NEXT
100 NEXT
```

Le programme utilise 4 fois l'instruction **SOUND**. Les trois instructions de son changent la hauteur à différentes valeurs dans certains intervalles, cependant que **SOUND 4, P, 15** choisit le bruit du canal 1 à plein volume. **PLAY 7, 1, Ø, Ø** met en service les trois générateurs de son, mais un seul canal de bruit. La fission nucléaire s'achève sur notre vieille connaissance **EXPLODE**. Cela nous ramène au point de départ de notre voyage dans le monde sonore de l'ORIC. Le reste vous concerne. Les programmes cités ne sont que des exemples de ce que l'on peut faire avec votre ORIC aux talents variés. Nous vous souhaitons de longues heures de créations sonores et musicales pour amuser vos amis et importuner vos voisins.

William Saint-Cricq / wsaintcr@wanadoo.fr Page 66 sur 163 15/08/2000

Numérisation http://wsc.n3.net



Chapitre 9
Les mots clés du BASIC

Dans cette partie, chacun des mots-clés du BASIC est étudié avec un exemple de son utilisation. Les informations sont données avec l'écriture correspondant à chaque commande, en utilisant les symboles suivants :

v	nom de variable numérique
v$	nom de variable chaîne
i,j	nombres entiers
n	nombre décimal (qui peut aussi être un entier), ou résultat d'une expression numérique
c	expression conditionnelle logique (comme A>B ou **TRUE**)
a$	chaîne de caractère
ln :	numéro de ligne de programme
adr	adresse d'un emplacement de mémoire

Pour rendre les commandes graphiques plus claires, nous utilisons aussi les symboles suivants :

fb	paramètre "premier plan/arrière plan" (Ø – 3). Voir chapitre 7 pour plus de détails.
s	caractères
x,y	coordonées d'un point de l'écran.

D'autres symboles pourront être utilisés si nécessaire. Ils seront définis dans le cours de la définition du mot-clé. Le code BASIC donné est la valeur décimale que l'ORIC utilise pour stocker les mots-clés du BASIC en octets dans la mémoire.

ABS

Code BASIC : 216
écriture : ABS (n)

Renvoie la valeur absolue du nombre ou de l'expression entre paranthèses. Par exemple **ABS** (– 19) est 19. Cette fonction peut être utilisée quand il est nécessaire d'obtenir un résultat positif lors d'un calcul sur deux variables. Par exemple, dans le calcul de **A – B**,

William Saint-Cricq / wsaintcr@wanadoo.fr Page 67 sur 163 15/08/2000

la différence ne sera positive que si A est supérieur à B. Cependant **ABS (A – B)** donnera toujours une valeur positive.

```
10 REM ..... ABS .....
20 FOR C=-500 TO 1000 STEP 150
30 PRINT ABS(C);
40 IF C<0 THEN PRINT"avant J.C." ELSE
PRINT"apres J.C."
50 NEXT
60 END
```

Cet exemple écrit les dates de 500 av. J.C. à 150. Sans **ABS**, cette boucle donnerait des valeurs av. J.C. négatives, c'est-à-dire – 350 av J.C.

```
10 REM .....ABS 2 ........
20 A=COS(PI)
30 IF A=-1 THEN PRINT "COS(PI)=";A
40 IF ABS(A+1)<1E-9 THEN PRINT"COS(PI)
="A"selon la precision de l'ORIC"
50 PRINT"D'ou "A"=-1 !"
```

ABS, est aussi utilisé pour des égalités défectueuses dues aux erreurs inévitables de conversion de décimales en binaire et vice-versa, dans l'ordinateur, que les incertitudes d'arrondissement peuvent provoquer. Dans le programme ci-dessus, l'ordinateur calcule (cos(PI) qui est – 1, mais n'est pas conservé exactement dans la mémoire de l'ordinateur. Utiliser **ABS** en ligne 40, s'assure que la valeur calculée est correcte dans la limite des erreurs de conversion.

Mot-clé associé : SGN.

AND

Code BASIC : 209
écriture : i AND i
c AND c

Le mot-clé basic **AND** est un opérateur logique qui correspond à la conjonction française "et". Quand il est combiné avec des expressions logiques, il donnera pour réponse vrai (= – 1) ou faux (= 0), selon que respectivement, les deux expressions sont vraies ensemble ou pas. Il est souvent utilisé en ce sens, en liaison avec

l'instruction **IF** pour s'assurer, que deux conditions sont remplies, avant qu'une instruction soit exécutée. Le premier exemple montre l'utilisation de **AND** dans ce cas, où l'instruction **GOTO** à la fin de la ligne 40 ne sera exécutée que si à la fois, A > 12 et A < 20.

```
10 REM...... AND .......
20 CLS:PRINT:PRINT
30 INPUT "QUEL EST VOTRE AGE ";A
40 IF A>12 AND A<20 GOTO 70
50 PRINT" VOUS N'ETES PAS UN(E) TEENAG
ER"
60 END
70 PRINT" VOUS ETES UN(E) TEENAGER"
80 END
```

Nous pouvons aussi utiliser **AND** pour combiner des représentations binaires de nombres dans la mémoire. Quand la commande est utilisée dans ce cas, chaque paire de bits correspondant au deux nombres en question est traitée comme une combinaison, de la même façon que ci-dessus, avec le bit du nombre résultant à 1 si les bits correspondants sont tous les deux à 1. Ainsi, **12 AND 9 (0000** 1100 **AND 0000** 1001) donnera **8 (0000 10000)**

Mots-clés associés : FALSE, NOT, OR, TRUE.

ASC

Code Basic : 236
écriture : ASC (a$)

Chaque caractère connu de l'ORIC est associé à un nombre code ASCII (prononcer à-SKI). Ainsi quand nous écrivons la ligne :

10P$ = "M"

l'ORIC, ne stocke pas exactement la lettre M, mais son équivalent ASCII, (dans ce cas 77).

Comme le montre le programme ci-dessous, **ASC** peut aussi être utilisé pour contrôler **INPUT**. La ligne 40, permet de voir si le premier caractère de la chaîne que vous avez entré, tombe entre les codes **ASCII** 65 et 90. Puisque les codes 65 – 90 correspondent aux

Numérisation http://wsc.n3.net
William Saint-Cricq / wsaintcr@wanadoo.fr Page 68 sur 163 15/08/2000

Numérisation http://wsc.n3.net

lettres capitales, l'ordinateur affichera seulement une entrée commençant par une majuscule.

```
10 REM ......ASC..........
20 CLS
30 INPUT"TAPEZ UNE CHAINE QUELCONQUE";
K$
35 CLS:PRINT:PRINT
40 IF ASC(K$)>64 AND ASC(K$)<91 THEN P
RINT K$
50 WAIT 100
60 GOTO 10
70 NEXT
80 END
```

Comme ce sont des fonctions de gestion de lignes, ce type de programme acceptera seulement “,”,“:”,“«»”,“”, ou “”, s'ils sont entre guillemets. Si cela vous intéresse, **ASCII** est une abréviation pour American Standard Code for Information Interchange et une liste complète est fournie à l'annexe 1.

Mots-clés associés : CHR$, HEX$, STR$, VAL.

ATN

Code BASIC : 229
écriture : ATN (n)

Cette fonction calcule un angle dont on connaît la tangente. Il faut remarquer que l'angle donné par ORIC est exprimé en radians. La valeur calculée appartient toujours à l'intervalle principal de $-Pi/_2$ à $Pi/_2$.

Ainsi **ATN(1)** donnera $Pi/_4$ de Ø.785398162 pour l'angle bien que n'importe quel angle de la forme $Pi/_4+n*Pi$ (où n est un entier) aura aussi une tangente de 1.

Cela peut être utile dans le programmes graphiques pour calculer l'angle d'une droite à partir du quotient des différences des coordonnées x et y de ces extrémités.

```
20 HIRES
30 INPUT"SOMMETS X1,Y1,X2,Y2";X1,Y1,X2
,Y2
```



```
40 CURSET X1,Y1,3
50 DRAW X2,Y2,1:WAIT 100
60 ANGLE=ATN((Y2-Y1)/(X2-X1)):TEXT
70 PRINT"L'ANGLE VAUT"ANGLE"RADIANS"
80 PRINT"OU"ANGLE*180/PI"DEGRES"
90 END
```

Mots-clés associés : COS, PI, SIN, TAN.

CALL

Code BASIC : 191
écriture : CALL-adr

Cette commande appelle le programme en langage machine, qui démarre à l'adresse de l'emplacement mémoire. Le retour en BASIC, se fait en utilisant le code **RTS**, pour arrêter le programme.

Utiliser cette commande peut entraîner la perte de votre programe si l'adresse n'est pas correcte et n'est pas en fait le début du programme machine. Voyez le chapitre utilisation du code machine, pour une connaissance complète de cette commande.

Essayez la commande suivante comme exemple de danger potentiel et du pouvoir de ce mot-clé.

CALL DEEK (#FFFC)

Cela transmet le contrôle au point froid, relançant l'ORIC comme s'il venait juste d'être allumé. De même appeler l'adresse donnée par **DEEK (#FFFA)** mettra l'ORIC au point chaud comme si vous aviez enfoncé le bouton reset.

Cela peut être utile pour retrouver les caractères originaux dans le déroulement d'un programme ou pour éviter d'avoir à retourner l'ORIC.

Mots-clés associés : DEEK, DOKE, PEEK, POKE, USR.

15/08/2000 Page 69 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

CHAR

Code BASIC : 176
écriture : CHAR n,s,fb.

Pour les possesseurs d'un **ORIC**, interessés par le graphique ou les jeux, **CHAR** est un don du ciel, puisqu'il permet de placer des caractères n'importe où sur l'**écran** haute résolution. **CHAR** écrit au point courant et, utilisé avec **CURMOV** et **CURSET**, permet au programmeur de placer du texte ou des caractères pré-définis à la place d'un simple point. Dans l'écriture ci-dessus, n (32 – 127) représente le code **ASCII**, s (**Ø ou 1**) est le paramètre qui détermine si l'écriture est normale ou inverse, et fb (Ø – 3) est le paramètre premier plan/arrière plan, correspondant à l'un des effets suivants :

Ø couleur du fond
1 couleur du premier plan
2 couleurs inverses
3 nul.

Si vous utilisez **CHAR,** dans d'autres cas que l'affichage haute résolution, l'ordinateur délivrera un message d'erreur. Il faut noter, que les coordonnées maximales du curseur sont x = 234 et y = 192.

```
1 REM...... CHAR .........
5 HIRES:C=0
10 FOR A=0 TO 150 STEP 50
20 CURSET 40+A,90,3
30 DRAW 20,0,2:DRAW 0,20,2
40 DRAW -20,0,2:DRAW 0,-20,2
50 CURMOV 7,6,3
60 CHAR 49+C,0,2
70 C=C+1
80 NEXT
```

Mots-clés associés : **CURMOV, CURSET, HIRES, PLOT.**

CHR$

Code BASIC : 237
écriture : CHR$ (i)

Cette fonction est intéressante pour programmer sur ORIC, pour plusieurs raisons :
CHR$(i) renvoie le caractère dont le code est contenu dans les paranthèses. Le nombre doit être un entier de l'intervalle Ø – 255. Cependant, l'usage de cette fonction dépasse la reproduction de l'ensemble des caractères et un rapide coup d'œil sur la liste complète des codes ACSII, dans l'annexe 1, nous révélera les possibilités de **CHR$**. Par exemple, le programme ci-dessous utilise **CHR$**, pour fixer les attributs qui modifient le fond et la couleur du caractère, pour le reste de l'instruction **PRINT**.

```
10 REM ...CHR$(i)
20 PRINT CHR$(129);
30 PRINT CHR$(150);
40 PRINT "ROUGE SUR CYAN"
50 END
```

Deux heures d'entraînement avec cette fonction, utilisée avec les codes des intervalles Ø – 31 et 128-151, vous révélera des possibilités de programmation que vous n'avez jamais imaginées de la part de votre ORIC.

Les possesseurs d'une imprimante ORIC trouveront aussi **CHR$** utile, en le combinant avec la commande **LPRINT**. Voyez les codes de contrôle dans l'annexe 7. Dans ce cas, **CHR$** permet d'écrire des caractères n'appartenant pas à la police de caractères habituels et permet de modifier le format d'écriture. **CHR$** est aussi utilisé avec des imprimantes ordinaires, pour renvoyer des codes de contrôle pour diverses fonctions de l'imprimante.

Mots-clés associés : **ASC, HEX$, STR$, VAL.**

CIRCLE

Code BASIC : 173
écriture : CIRCLE n,fb

Cette commande dessine un cercle en mode haute résolution, dont le centre est la position courante du curseur. C'est pourquoi

William Saint-Cricq / wsaintcr@wanadoo.fr Page 70 sur 163 15/08/2000

Numérisation http://wsc.n3.net

CIRCLE est habituellement utilisé en liaison avec la commande **CURSET**. Si une partie du cercle quitte l'écran, l'ORIC affichera un message d'erreur.

Le premier paramètre, entier, est le rayon en points (1 – 99) et fb vaut Ø à 3. Dans cet exemple, quatre ensembles de cercles sont dessinés, à quatre positions de curseur différentes. L'ORIC les efface avec fb fixé à Ø (fond).

```
10 REM....CIRCLE.......
20 HIRES
30 FOR A=0 TO 30 STEP 10
40 FOR B=1 TO 0 STEP -1
50 CURSET 80+A,100,0
60 CIRCLE A+9,B
70 NEXT B
80 NEXT A
90 END
```

Mots-clés associés : CURMOV, CURSET, DRAW, HIRES, PATTERN.

CLEAR

Code BASIC : 189
écriture : CLEAR

Cette commande efface les valeurs de toutes les variables d'usage courant. Le programme suivant écrit cinq variables tant numériques que chaînes, la ligne 8Ø annule les variables et les lignes 9Ø à **1ØØ**, affichent les valeurs nulles (Ø pour variables numériques, et chaîne vide pour variables chaînes).

```
1 REM ..... CLEAR .....
5 CLS
10 A=5
20 B=10
30 C=20
40 PRINTA,B,C
50 C$="OR"
60 B$="IC"
70 PRINT C$+B$
80 CLEAR
90 PRINTA,B,C
100 PRINT C$+B$
```

Mots-clés associés : NEW, RUN.

CLOAD

Code BASIC : 182
écriture :

CLOAD "nom de fichier" (,S)
CLOAD "" (,S)
CLOAD "nom de fichier", J (,S)
CLOAD "nom de fichier", V (,S)

Cette commande vous permet de charger un programme BASIC d'un ficher en langage machine, à partir d'une cassette. Elle peut prendre plusieurs écritures :

CLOAD ""

Si la cassette contient seulement un programme, ou si vous êtes certain de l'emplacement d'un programme particulier, cette écriture peut être utilisée, il charge alors le premier programme complet qu'il trouve.

Comme toutes les écritures **CLOAD**, la commande doit être accompagnée par le **,S** facultatif, si le programme a été sauvegardé en mode lent.

CLOAD "nom de fichier"

Si vous recherchez un programme particulier dont le nom correct est connu, l'écriture ci-dessus peut être utilisée dans laquelle "nom de fichier" est n'importe quel nom, jusqu'à 16 caractères de longueur.

Cependant, il est préférable de garder les noms de fichiers aussi courts et mémorisables que possible, parce que si vous ajoutez ou retranchez un caractère, fût-ce un espace, le programme ne sera pas chargé.

Si vous oubliez un nom de fichier, vous devrez utiliser la première écriture, **CLOAD ""** et fouiller dans la cassette entière jusqu'à ce que vous trouviez le programme que vous cherchez.

L'ORIC délivrera des messages vous informant des démarches. Il affichera **SEARCHING** (je cherche) sur la ligne spéciale, jusqu'à ce que le programme spécifié soit rencontré, et ensuite **LOADING** (je charge). Un nom de fichier est affiché quand un programme est rencontré sur la bande magnétique. Les noms de programme sont suivis par B, pour indiquer un fichier en BASIC, et un bloc mémoire sauvegardé, par C, pour un fichier en code machine.

Les fichiers trouvés, mais que l'on ne doit pas charger, sont signalés par le message **FOUND** (trouvé) NOM DE FICHIER, suivi

15/08/2000 Page 71 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

de **B** ou **C** et le fichier convenable donne LOADING... NOM DE FICHIER (B ou C).

L'ORIC affichera aussi les message **ERRORS FOUND**, si des erreurs sont détectées au chargement et bloquera le lancement automatique (AUTO) si celui-ci a été spécifié dans la commande **CSAVE.**

Si vous sauvegardé un bloc d'emplacements-mémoire sous un nom de fichier particulier, vous avez dû spécifier le début et la fin de ce bloc dans votre instruction **CSAVE**. Cependant, quand vous chargez un tel fichier, vous avez besoin seulement du nom de fichier.

CLOAD peut être utilisé avec 2 finalités supplémentaires :

CLOAD "nom de fichier", J

L'écriture ci-dessus vous permet d'adjoindre un second programme basic à la fin d'un programme chargé auparavant. Cependant, il est nécessaire que tous les numéros de ligne du 2e programme soient plus grands que le plus élevé des numéros de ligne du premier programme ; si ce n'est pas le cas, le programme ne pourra pas être lancé car les lignes recopiées du deuxième programme, coexisteront avec leur contrepartie, dans le premier.

CLOAD "nom de fichier", V

Cette dernière écriture vous permet de vérifier, que votre programme (ou bloc mémoire) a été correctement sauvegardé.

Si vous tapez avec l'écriture ci-dessus et chargez le programme normalement, votre programme d'origine se place dans la mémoire de l'ORIC.

Quand le programme commencera à être chargé, l'ORIC affichera **VERIFYING** NOM DE FICHIER, en haut de l'écran, et si le chargement s'effectue sans problème, vous obtiendrez : **Ø VERIFY ERROR DETECTED** (aucune erreur de vérification).

Cela indique que le programme a été sauvegardé sans problème, et il est à l'abri d'un effacement de mémoire. Néanmoins, il est toujours prudent de faire plus d'une copie d'un programme. Si le programme est erroné d'après cette vérification, votre listing d'origine est toujours à l'abri dans la mémoire de l'ORIC et vous devez essayer de charger et vérifier à nouveau.

Si après plusieurs essais, l'ORIC refuse de vous fournir le message : **Ø** erreur détectée, (bien que vous ayez ajusté le volume et la tonalité sur votre magnétophone), vous serez obligé de conclure que la sauvegarde n'a pas été correcte et vous devrez sauvegarder à nouveau.

Il faut se souvenir que **CSAVE** et **CLOAD** se font en mode rapide (vitesse : 2 400 bauds) sans l'instruction finale ,S qui précise le

transfert de données en mode lent (Slow) vitesse 300 bauds.

Si un programme a été sauvegardé en mode lent, on ne peut le charger qu'en mode lent donc le ,S final doit figurer dans l'écriture.

Mots-clés associés : CSAVE, RECALL, STORE

CLS

Code BASIC : 148
écriture : CLS

Cela efface l'écran et le met à la couleur de fond courante. En règle générale, il est préférable de commencer tout programme, de texte avec **CLS** sauf si vous avez une bonne raison de vouloir que le programme reste visible sur l'écran.

Dans l'exemple l'ORIC efface deux fois l'écran. La première fois en ligne **2Ø**, quand l'écran est libéré en préparation à l'affichage et ensuite en **7Ø**, lorsque l'écran est rempli avec du texte. (L'usage correspond à celui de **CTRL-L**).

```
10 REM......CLS......
20 PAPER 0:INK 5:CLS
30 FOR A=0 TO 134
40 PRINT"ORIC";
50 NEXT
60 WAIT 100
70 CLS
80 WAIT 100
90 GOTO 20
```

Mots-clés associés : HIRES, LORES, TEXT

CONT

Code BASIC : 187
écriture : CONT

Cette commande directe est utilisée pour faire repartir un programme après un arrêt. Par exemple, pendant qu'un programme tourne, vous utilisez sans doute **CTRL-L** pour arrêter le processus

Numérisation http://wsc.n3.net
William Saint-Cricq / wsaintcr@wanadoo.fr Page 72 sur 163 15/08/2000

Numérisation http://wsc.n3.net

afin d'examiner l'affichage sur l'écran ou l'état d'une variable particulière. Dans la plupart des cas vous ne voulez pas relancer le programme au début mais continuer à partir de l'endroit où vous vous êtes arrêté.

C'est là que **CONT** intervient habituellement, et est entré en commande directe. Notez que **CONT** ne vous permet pas de repartir si vous avez changé une partie du programme. Ce n'est qu'une commande directe, ce qui fait que, écrit dans le corps d'un programme **CONT** le "plantera".

Mots-clés associés : RUN, STOP

COS

Code BASIC : 226
écriture : COS(n)

Calcule le cosinus d'un angle n. Il est important de se rappeller que le nombre entre parenthèses est exprimé non pas en degrés mais en radians.

Pour convertir des radians en degrés, puisque 2*Pi radians = 360°, nous multiplions par 180/Pi. L'exemple ci-dessous, alterne les courbes sinus et cosinus.

```
10 REM....COSINUS ET SINUS......
20 HIRES:PRINT:PRINT:PRINT
30 INPUT "COS ou SIN ";A$
40 INPUT"VALEUR (1 a 99)";V
50 CURSET 0,100,3:DRAW 239,0,1
60 FOR A=40960 TO 49079 STEP 40
70 POKE A,INT(RND(1)*2)+16
80 NEXT
90 FOR A=-PI TO PI STEP 0.02
100 IF A$="COS" THEN B=COS(A)
110 IF A$="SIN" THEN B=SIN(A)
120 CURSET A*38+120,(B*V)+99,1
130 NEXT
140 PRINT"COURBE "+A$:WAIT 100
150 INPUT"CONSERVEZ-VOUS CETTE COURBE
(O/N)";M$
160 IF M$="O" THEN 30
170 GOTO 20
```

Mots-clés associés : ATN, PI, SIN, TAN

CSAVE

Code BASIC : 183
écriture :

CSAVE "nom de fichier" (,S)
CSAVE "nom de fichier" ,A addr, E addr (,S)

Cette commande est utilisée pour sauvegarder un programme ou une série d'emplacements de mémoire sur cassette. Elle peut prendre plusieurs écritures :

CSAVE "nom de fichier"

C'est l'usage le plus courant dans lequel "nom de fichier" est n'importe quel nom jusqu'à 16 caractères de longueur.

L'usage de cette instruction sauvegardera le programme BASIC courant, qui sera stocké sous le nom spécifié. Comme toutes les commandes **CSAVE**, elle peut être suivie par virgule S **(,S)** facultatif, cela sauvegardera le fichier en mode lent (slow).

Bien que l'ORIC sauvegarde correctement dans les modes lent et rapide, il vaut mieux garder finalement une copie de chaque programme important en mode lent.

CSAVE "nom de fichier", AUTO

Cela marche exactement comme le premier exemple sauf que lorsque le programme a été chargé à nouveau dans l'ORIC, il se mettra en route **AUTO**matiquement.

Il est aussi possible de sauvegarder le contenu d'un bloc d'emplacement mémoire en utilisant A suivi par l'adresse de départ (en héxadécimal ou décimal) pour spécifier le début d'un bloc et E (pour end) pour spécifier l'adresse de fin.

Prenez l'exemple d'un écran en texte ou en haute résolution que vous voulez conserver. L'écriture **CSAVE** pour cela est comme suit (48K, en 16K retrancher 32768 aux adresses).

Pour l'affichage **HIRES** 6haute résolution) :
CSAVE "nom de fichier", A40960,E48000.

Pour l'affichage **TEXT** ou **LORES** (basse résolution) :
CSAVE "nom de fichier", A48000,E49119.

Mots-clés associés : CLOAD, RECALL STORE.

15/08/2000
Page 73 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

CURMOV

Code BASIC : 171
écriture : CURMOV x,y,fb

Cette commande place le curseur à une nouvelle position en mode haute résolution. Les valeurs de x et y ne sont pas intrinsèques, mais relatives à la position courante du curseur. fb vaut 0, 1, 2 ou 3.

CURMOV est utile pour simuler un mouvement sur l'écran et plus encore dans la création de nombreux programmes graphiques.

Le simple exemple ci-dessous place un cercle sur l'écran et utilise le paramètre fb pour l'effacer ensuite.

```
10 REM.....CURMOV.......
20 HIRES:INK 5:PAPER 4
30 FOR C=0 TO 100 STEP 20
40 FOR B=1 TO 0 STEP -1
50 CURSET 20,50+C,0
60 FOR A=0 TO 30
70 CIRCLE 10+A,B
80 CURMOV 5,0,0
90 NEXT A
100 NEXT B
110 NEXT C
```

Mots-clés associés : CIRCLE. CURSET. DRAW, HIRES.

CURSET

Code BASIC : 170
écriture : CURSET x, y, fb.

Détermine la position intrinsèque x, y du curseur en mode haute résolution. Pour tenir dans l'intervalle, x doit être compris entre 0 et 239, cependant que y doit être compris entre 0 et 199. Comme ailleurs, fb est un entier de 0 à 3. Puisque la commande **CIRCLE** place le centre du cercle à la position courante du curseur, ce court programme, ci-dessous, est la parfaite démonstration de cette commande en action.

```
10 REM.....CURSET.......
20 HIRES:INK 5:PAPER 4
30 FOR C=0 TO 100 STEP 20
```



```
40 FOR B=1 TO 0 STEP -1
50 CURSET 20,50+C,0
60 FOR A=0 TO 30
70 CIRCLE 10+A,B
80 CURMOV 5,0,0
90 NEXT A
100 NEXT B
110 NEXT C
```

Mots-clés associés : CIRCLE, CURMOV, DRAW, HIRES.

DATA

Code BASIC : 145
écriture : DATA n, n, n...
DATA a$, a$, a$...

Cette instruction précède et définit une liste de données (**DATA**) qui associée avec **READ**, peuvent être lues (read) comme des variables.

La liste peut prendre la forme de nombres, de mots, ou de lettres, et si vous voulez conserver des espace liminaires, les données doivent être écrites entre guillemets.

L'instruction **DATA** peut être placée en n'importe quel point du programme, sans s'occuper de l'endroit où les données seront lues.

Si vous voulez placer une virgule dans une instruction **DATA** comme l'une des données, elle doit être mise entre guillemets. Il est essentiel que la donnée soit introduite dans une variable appropriée, c'est-à-dire numérique pour les nombres, chaîne pour les mots, les lettres et les symboles. Les données sont lues exactement dans l'ordre dans lequel elles apparaissent dans une instruction **DATA**. Ce n'est qu'en utilisant la commande **RESTORE** que l'on peut renvoyer le pointeur au début d'une ligne de **DATA**.

Dans l'exemple, les données, en ligne 70 et 90, sont lues en ligne 20 et affichées en ligne 30.

```
10 REM.........DATA.........
20 FOR I=1 TO 2:FOR J=1 TO 2:READ A$,A
30 PRINTA$,A
40 NEXT
50 NEXT
60 END
70 DATA "SALUT",1,"LES",2
80 DATA  COPAINS,3,"D'ORIC",4
```

Mots-clés associés : READ, RESTORE.

William Saint-Cricq / wsaintcr@wanadoo.fr Page 74 sur 163 15/08/2000

Numérisation http://wsc.n3.net

DEEK

Code BASIC : 231
écriture : DEEK (adr)

Cette fonction nous permet d'examiner la valeur stockée en mémoire, dans les 2 emplacements adr et adr + 1. Il calcule automatiquement la valeur (Ø-65535) stockée dans le double octet spécifié. L'écriture ordinaire que le processeur 65Ø2 utilise pour stocker les adresses, est un couple d'octets dont le premier est de poids faible. L'action de **DEEK** est de prendre la valeur stockée dans le second octet, de poids fort, à l'adresse adr + 1 et de le multiplier par 256 puis d'ajouter la valeur de l'octet stocké à l'emplacement adr. Dans les explications de **CALL**, nous expliquons comment **DEEK** peut être utilisé pour examiner les adresses stockées dans la mémoire morte de l'ordinateur.

Comme exemple plus poussé de l'utilisation de **DEEK**, on peut utiliser **DEEK** (# A6) pour examiner la valeur courante de **HIMEM** qui n'est pas accessible normalement.

Mots-clés associés : CALL, DOKE, PEEK, POKE, USR.

DEF

Code BASIC : 184
écriture : DEF FN V(z) = exp
DEF USR = adr

DEF FN est utilisé pour **DEF**inir une Fonction Numérique. L'ORIC est équipé de plusieurs fonctions numériques pré-programmées, mais la commande **DEF FN** vous permet de construire votre propre fonction dans un programme, essentiellement **DEF FN** peut être utilisé pour éviter de reproduire une ligne qui effectue les mêmes calculs chaque fois que ce calcul est nécessaire.

Dans l'écriture ci-dessous V (nom de variable d'une seule lettre) complète le nom par lequel la fonction peut être rappelée (en utilisant la commande **FN**, plus loin dans le programme).

Le z entre parenthèses est le nom qui est utilisé pour l'argument de la fonction, et est considéré comme une variable muette.

L'expression suivant le signe =, est une expression utilisant z entre parenthèses, la variable muette. C'est le calcul qui devra être fait sur l'argument introduit par l'appel de **FN**. Dans l'exemple ci-dessous, la boucle détermine la valeur de A qui est utilisée comme argument dans la fonction.


```
10 REM.......DEF.......
20 DEF FN M(Z)=Z/6*2
30 FOR A=10 TO 100 STEP 10
40 PRINT FN M(A)
50 NEXT
```

DEF USR est utilisé pour définir l'adresse de départ d'un programme machine. Voir la partie **USR**

```
 10 REM.......DEF USR........
 20 DEF USR=5120
 30 FOR I=0 TO 17
 40 READ A:POKE 5120+I,A
 50 NEXT
 60 DATA 162,5,189,12,20,157,128,187,20
2,208,247,96
 70 DATA 0,#48,#45,#4C,#4C,#4F
 80 REPEAT
 90 DUMMY=USR(0)
 100 WAIT 50
 110 FOR I=48001 TO 48006
 120 POKE I,32:WAIT 10
 130 NEXT
 140 WAIT 50
 150 UNTIL FALSE
```

Mots-clés associés : FN, USR

DIM

Code BASIC : 147
écriture : DIM v (i, j...)
DIM v% (i, j...)
DIM v$ (i, j...)

L'instruction **DIM** est utilisée pour **DIM**ensionner des tableaux. Les tableaux permettent de stocker des nombres décimaux entiers ou des chaînes, comme éléments d'un tableau à une dimension (une liste) ou dans un tableau à plusieurs dimensions.

Une simple liste de 6 entiers, par exemple, a un espace réservé en mémoire avec l'instruction **DIM A% (5).**

La variable du tableau A dans ce cas, doit être une lettre unique pour tous les tableaux. Le tableau A% (J) a 6 éléments numérotés

William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Numérisation http://wsc.n3.net

de 0 à 5, auxquels peuvent être assignés des valeurs, avec des instructions telles que : **LET** A% (3) = 276.

Les tableaux de chaîne sont désignés par un $ après le nom de variable, par exemple D$(3) et un tableau de nombres décimaux, a juste un nom de variable d'une seule lettre telle que T(8).

L'ORIC permet l'utilisation de tableau avec 11 éléments ou moins (indiqués 0 à 10), sans utiliser l'instruction **DIM**.

Cela se fait simplement en donnant une valeur à l'un des éléments du tableau. L'usage d'une instruction telle quel **LET N(5) = 6** fabrique automatiquement un tableau N(10) et assigne la valeur 6 au 6e élément N(5). Si nous voulons utiliser des tableaux qui ont un plus grand nombre d'éléments alors l'instruction **DIM** est nécessaire.

Les tableaux sont généralement **DIM**ensionés au début du programme et lorsqu'un tableau particulier a été fabriqué, par **DIM**, il ne doit pas être modifié dans la suite du programme. Sinon, l'erreur **REDIM'D ARRAY ERROR** (tableau redimensionné) est affichée.

Le programme suivant utilise **DIM** pour classer le tableau de chaînes **D$(7)**, assigne une valeur à chaque élément en utilisant **READ** et **DATA**, puis affiche la liste.

```
 10 REM .... DIM .....
 20 DIM D$(7)
 30 FOR I=1 TO 7
 40 READ D$(I)
 50 NEXT
 60 DATA "TIQUE.","EN ","ORDRE AL","CLA
SSEMENT ","DONNEES ","PHABE","DE "
 70 FOR A=1 TO 6
 80 FOR B=A TO 7
 90 IF D$(A)<D$(B) THEN 110
 100 T$=D$(A):D$(A)=D$(B):D$(B)=T$
 110 NEXT
 120 NEXT
 130 FOR I=1 TO 7
 140 PRINT D$(I);
 150 NEXT
 160 END
```

Des tableaux de plus d'une dimension peuvent être utilisés. Dans l'exemple suivant un tableau d'entrées à 2 dimensions, N%(3,4) est dimensionné, les valeurs assignées aux éléments, et ensuite le tableau est affiché avec 4 colonnes et 5 lignes.


```
10 REM .....DIM
20 DIM N%(3,4)
30 FOR A=0 TO 3
40 FOR B=0 TO 4
50 N%(A,B)=A*10+B
60 NEXT
70 NEXT
80 FOR K=0 TO 4
90 FOR J=0 TO 3
100 PRINTN%(J,K),
110 NEXT
120 PRINT
130 NEXT
```

Les tableaux de chaînes admettent pour chaque élément une longueur maximale de chaîne de 255 caractères. Le nombre maximum de dimensions permis dans un tableau est 255 aussi, et bien que vous puissiez avoir des tableaux multi-dimensionnés, la limite du nombre d'éléments du tableau que vous pouvez pratiquement utiliser est déterminée par la mémoire de votre ORIC.

Rappelez-vous que les tableaux consomment rapidement de la mémoire et n'utilisez pas de tableaux non nécessaires.

Mots-clés associés : aucun.

DOKE

Code BASIC : 138
écriture : DOKE adr, i

Cette commande est utilisée pour stocker un double octet d'une valeur entière (dans l'intervalle 0-65535) en mémoire. Le 1er paramètre est l'adresse du 1er octet du couple d'octet dans lequel nous désirons stocker la valeur et le 2e paramètre entier de l'écriture est la valeur à stocker. L'écriture dans laquelle elle est stockée est la représentation habituelle du 6502, poids faible, poids fort. (voir chapitre code-machine). Si nous voulons stocker 770 dans la mémoire, nous devons utiliser la commande :

DOKE 30000, 770

Cela a pour effet de stocker 2 à l'emplacement 30000 et 3 à l'emplacement 30001 puisque 770 vaut 3*256 + 2.

Mots-clés associés : CALL, DEEK, POKE, USR

15/08/2000
Page 76 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

DRAW

Code BASIC : 132
écriture : DRAW x, y, fb

C'est l'une des commandes graphiques de l'ORIC qui ne peuvent être utilisées qu'en mode graphique haute résolution. **DRAW** est utilisé, associé à **CURMOV** et **CURSET**, et dessine (draw) une ligne continue à partir de la position courante du curseur, jusqu'à un point situé x points le long de l'axe des x, et y points sur l'axe des y. Ce qui fait que les coordonnées x et y spécifiées dans l'instruction **DRAW** ne sont pas intrinsèques (c'est-à-dire ne représentent pas les coordonnées de l'écran au sens propre du terme) mais sont relatives à la position du curseur.

Aussi x doit appartenir à l'intervalle Ø-199 et y à l'intervalle Ø-239. Noter que si votre instruction **DRAW** fait sortir la ligne de l'écran, un message d'erreur sera affiché.

Comme d'habitude, le code fb vaut entre Ø et 3 (voir le paragraphe spécial **CHAR**).

Bien que **DRAW** produise normalement une ligne continue sur l'écran, associé avec **PATTERN**, la ligne peut être tracée en pointillés de différentes espèces (voir **PATTERN**).

Pour une illustration animée de **DRAW**, voyez l'exemple pour haute résolution.

```
1 REM.......DRAW.......
5 HIRES:PAPER6:INK1
10 FOR A=0 TO 230 STEP 10
20 CURSET A,0,0
30 DRAW 0,199,1
40 WAIT 20:PING
45 NEXT
50 FOR Y=0 TO 190 STEP 10
60 CURSET 20,Y,0
70 DRAW 219,0,1
75 WAIT 20:PING
80 NEXT
```

Mots-clés associés : CIRCLE, CURMOV, CURSET, HIRES, PATTERN.

EDIT

Code BASIC : 129
écriture : EDIT ln

Cette instruction vous permet d'amener la ligne spécifiée par ln, sous la position courante du curseur pour l'édition.

Bien qu'il soit possible d'éditer une ligne particulière sur l'écran après l'avoir listée, puisque les insertions doivent être tapées sur les lignes au-dessus ou en-dessous de la ligne éditée, il est très facile d'être sûr de ce que vous ajoutez quand la ligne est séparée du reste du programme.

En d'autres cas, les touches de mouvement du curseur sont utilisées pour placer le curseur au début de la ligne à modifier, et **CTRL-A**, pour copier les parties de ligne que vous voulez conserver. L'insertion de caractères est pleinement expliquée dans le chapitre 2, mais essentiellement, cela oblige à taper les caractères nécessaires en position correcte sur la ligne au-dessus ou en-dessous de la ligne éditée, à inverser le mouvement du curseur pour réentrer dans la ligne au point désigné, et enfin à recopier d'autres parties correctes jusqu'à ce qu'enfin, on appuie sur **RETURN**.

Mots-clés associés : LIST.

END

Code BASIC : 128
écriture : END

Cette commande peut être utilisée pour arrêter un programme si vous voulez éviter le message **BREAK** in ln, que l'ORIC donnera si la commande **STOP** est utilisée. Bien qu'elle soit généralement placée comme dernière instruction d'un programme, il y a plusieurs circonstances dans lesquelles son usage permet de "planter" avec élégance quand une donnée incorrecte est entrée.

L'exemple montre comment utiliser **END**, dans ce but.

```
10 REM.....END......
20 REPEAT
30 INPUT"NOMBRE";NB
40 IF NB<+0 THEN END
50 PRINT "LE LOGARITHME NEPERIEN DE"NB
"EST"LN(NB)
60 UNTIL FALSE
```

Mots-clés associés : STOP

William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Numérisation http://wsc.n3.net

EXP

Code BASIC : 225
écriture : EXP(n)

C'est une autre des nombreuses fonctions mathématiques de l'ORIC **EXP(n)** renvoie e(= 2.7183) à la puissance n. Il est souvent utilisé en association avec LN (loge), puisque **EXP [LN(n)]** "neutralise" le logarithme.

```
10 REM......EXP.......
20 HIRES
30 CURSET 30,30,0
40 DRAW 0,150,1:DRAW 190,0,1
50 FOR N=0 TO 5 STEP .025
60 X=N*40+30
70 Y=180-EXP(N)
80 CURSET X,Y,1
90 NEXT
100 END
```

Mots-clés associés : LN, LOG

EXPLODE

Code BASIC : 164
écriture : EXPLODE

C'est un des nombreux sons pré-définis possibles sur l'ORIC. Ils sont surtout utilisés dans les jeux d'arcade mais peuvent aussi être utilisés dans des programmes d'inspiration plus sérieuse.
Des retardements **(WAIT)** peuvent aussi être utilisé pour augmenter leurs possibilités. **WAIT** doit être utilisé si des explosions répétées sont nécessaires.

```
10 REM.......EXPLODE..........
20 HIRES:PAPER3:INK4
30 FOR B=0 TO 95 STEP 4
40 CURSET 120,2+B,0
50 GOSUB 200
60 CURSET 120,190-B,0
70 GOSUB 200
```



```
80 NEXT
90 EXPLODE
100 FOR K=1 TO 10
110 PAPER 4:INK 3
120 WAIT K
130 PAPER 3:INK 4
140 WAIT K
150 NEXT
160 WAIT 200
170 GOTO 20
180 END
200 FOR I=1 TO 3
210 CHAR 100,1,1:CURMOV 6,0,0
220 NEXT
230 RETURN
```

Mots-clés associés : PING, SHOOT, ZAP

FALSE

Code BASIC : 240
écriture : FALSE

Si vous utilisez votre ORIC pour traiter des données, il y a certaines circonstances pour lesquelles il doit décider si quelque chose est vrai ou faux.

```
1 REM........FALSE........
10 FOR A=11 TO 20
20 IF A<10 THEN PRINT TRUE ELSE PRINT
FALSE
30 NEXT
40 END
```

Le résultat du programme ci-dessus n'est pas spectaculaire, mais très compréhensible en termes d'ORIC. Puisque le 10 de la ligne 20 ne peut jamais être égal à A (puisque la boucle commence à 11), l'ORIC écrit simplement 10 zéros. C'est parce que zéro est la représentation de l'ordinateur pour faux.
Bien qu'il soit possible de remplacer **FALSE** par zéro dans un

William Saint-Cricq / wsaintcr@wanadoo.fr Page 78 sur 163 15/08/2000

Numérisation http://wsc.n3.net

programme, dans une jungle complexe de codes, cette fonction peut servir à clarifier un listing. Ainsi :

```
10 REM ....FALSE 2.......
20 A=10:B=1
30 REPEAT
40 PRINT (A>B);:B=B+1
50 UNTIL FALSE
```

écrira 9 fois “ – 1” (représentation de **TRUE**) avant que l'instruction de la ligne 40 (A > B) devienne fausse quand A = B (c'est-à-dire 10 = 10).

A partir de ce point, l'ORIC sortira une ligne sans fin de zéros. Aussi, bien que **FALSE** en ligne 50 puisse être remplacé par zéro sans que l'exécution du programme soit modifiée, il paraît clair que l'usage de **FALSE**, clarifie considérablement ce qui se passe dans le programme.

Mots-clés associés : TRUE.

FILL

Code BASIC : 175
écriture : FILL i_1, i_2, i_3

C'est une autre des commandes graphiques de l'ORIC, elle vous permet de remplir des emplacements déterminés de l'écran haute résolution avec une valeur particulière. L'écran de l'ORIC se compose de 200 lignes avec 40 emplacements par ligne.

Suivant l'écriture de la commande ci-dessus, **FILL** remplit i_1 lignes de i_2 segments avec une valeur de i_3 à la position courante du curseur. Donc i_1 est dans l'intervalle 1-200, i_2 est dans l'intervalle 1-40, et i_3 doit être entre 0 et 255.

FILL est habituellement utilisé pour colorier différentes régions de l'écran avec des couleurs déterminées.

Son rôle dans l'ensemble des commandes graphiques possibles sur l'ORIC est pleinement expliqué dans le chapitre 7, cependant, le programme montre **FILL** en action.

```
1 REM .....FILL
5 PAPER 0
10 HIRES:PRINTCHR$(17)
20 REPEAT
```



```
30 X%=RND(1)*231
40 Y%=RND(1)*175+7
50 A%=RND(1)*(230-X%)/6+1
60 D%=RND(1)*(182-Y%)+8
70 CURSET X%+6,Y%-7,0:FILL D%+7,A%,16
80 CURSET X%,Y%,0:FILL D%,A%,(17+RND(1
)*7)
90 UNTIL FALSE
```

Mots-clés associés : CURSET, CURMOV, HIRES.

FN

Code BASIC : 196
écriture : FN v(n)

FN suivi par un nom de variable peut seulement être utilisé dans un programme si elle a été précédée par **DEF FN** puisqu'elle se réfère à une fonction qui a été définie auparavant dans le programme. Pour une description complète de l'usage de fonctions définies, voyez le chapitre 4 et le paragraphe **DEF**.

L'exemple suivant montre **FN** en action transformant des radians en degrés.

```
10 REM......FN.......
20 DEF FNDEG(R)=R*180/PI
30 REPEAT
40 INPUT"RADIANS";A
50 PRINT"="FNDEG(A)"DEGRES"
60 PRINT
70 UNTIL A=999
```

Mots-clés associés : DEF.

William Saint-Cricq / wsaintcr@wanadoo.fr Page 79 sur 163 15/08/2000

Numérisation http://wsc.n3.net

FOR... TO... (STEP) NEXT

Code BASIC : 141... 195... (283)... 144
écriture : FOR v = n TO n (STEP n)
NEXT v

L'activité la plus importante d'un ordinateur consiste en la répétition de tâches simples. En BASIC la constitution d'une boucle est l'un des moyens par lesquels une telle répétition peut être effectuée.

Les boucles **FOR... NEXT** sont les exécutions les plus courantes d'une telle constitution. Une boucle **FOR... NEXT** oblige l'ordinateur à effectuer les instructions qu'elle comporte, un certain nombre de fois, par exemple :

```
10 FOR A=1 TO 3
20 PRINT A
30 NEXT A
```

La boucle ci-dessus affichera les nombres 1, 2, 3. Étudions-la et voyons comment elle fonctionne. Tout d'abord, l'ordinateur prend la variable A et lui assigne la valeur 1, qu'elle écrit en ligne 2Ø. Il continue alors jusqu'à ce qu'il rencontre le mot **NEXT** et alors retourne ligne 1Ø. Le processus est alors répété, avec la valeur de A incrémentée de 1 à chaque passage de la boucle jusqu'à A = 3. A ce point, le processus est terminé et le programme continue et exécute la ligne suivant l'instruction **NEXT**. Dans l'exemple ci-dessus, la variable est incrémentée par pas de 1, qui sera toujours le cas si le programme ne précise rien d'autre. Vous pouvez utiliser le mot facultatif **STEP** (pas) pour modifier le pas.

```
10 FOR A=5 TO 20 STEP 5
20 PRINT A
30 NEXT A
```

La boucle ci-dessus écrira 5,1Ø, 15, 2Ø. Vous pouvez avoir remarqué qu'en ligne 3Ø le nom de la variable a été omis après **NEXT**. Le BASIC de l'ORIC permet une telle omission, mais pour raisons de clarté, il est d'usage de laisser le nom de la variable surtout si votre programme contient des boucles à l'intérieur de boucles.

Un pas peut avoir des valeurs négatives. Ainsi :

```
10 FOR A=20 TO 5 STEP -5
20 PRINT A
30 NEXT A
```

est convenable.

Le pas peut aussi être non entier. On peut utiliser :

```
10 FOR A=1 TO 2 STEP .2
20 PRINT A
30 NEXT A
```

Les valeurs initiales, finales et le pas peuvent aussi être des variables non entières où des expressions calculées.

```
5 A=56:C=PI/2
10 FOR X=2*4 TO A/3 STEP C
20 PRINT X
30 NEXT X
```

Mots-clés associés : REPEAT, UNTIL.

FRE

Code BASIC : 218
écriture : FRE (Ø)
FRE ""

Cette instruction a deux écritures pour deux fonctions distinctes. Dans l'écriture ci-dessus, le premier exemple renvoie le nombre d'octets de mémoire encore disponible à un moment donné, dans le développement d'un programme. Le deuxième exemple d'écriture consititue un ramassage de déchets. C'est simplement un moyen de rendre plus soigné le stockage des chaînes (partant juste d'en dessous de **HIMEM** et décroissant).

Elle est utile si vous transformez des chaînes qui doivent devenir plus courtes quand elles prennent leurs nouvelles valeurs.

En effet lorsque leur valeur a été changée la taille de la mémoire laissée pour la nouvelle chaîne plus courte restera la même. (L'ORIC stocke réellement des espaces en surplus).

Si les nouvelles chaînes sont plus longues elles devront être stockées ailleurs laissant la place entière allouée auparavant à l'ancienne valeur vide. En utilisant **FRE ""** vous rangez à nouveau les chaînes et vous débarrassez des espaces inutiles.

William Saint-Cricq / wsaintcr@wanadoo.fr Page 80 sur 163 15/08/2000

L'exemple ci-dessous montre le ramassage des déchets au travail.

```
10 REM......FRE........
20 PRINT FRE(0)
30 A$="A":B$="B"
40 REPEAT
50 A$=A$+A$:B$=B$+B$
60 UNTIL LEN(A$)=128
70 PRINTFRE(0)
80 PRINT"MAINTENANT,ON NETTOIE"
90 A$="":B$=""
100 PRINT FRE("")
```

Mots-clés associés : aucun.

GET

Code BASIC : 190
écriture : GET V$
GET V

"Enfoncer n'importe quelle touche pour continuer" est l'une des indications les plus utilisées dans des programmes inter-actifs. Bien, si le programme est écrit en basic ORIC, c'est très agréable que **GET** suivi par une variable chaîne, retienne le programme jusqu'à ce que vous soyiez prêt à poursuivre. Cela ressemble à **INPUT**.

GET arrête le programme et ne permettra à l'ORIC de continuer que si une touche est enfoncée. Dans l'instruction

100 GET A$

le caractère de la 1re touche enfoncée sera stockée en **A$**. Cependant, contrairement à **INPUT**, **GET** ne nécessite pas **RETURN**, mais automatiquement continue à la ligne suivante du programme dès qu'une touche est enfoncée.

Cependant, **GET A$** ne permet de placer qu'un seul caractère dans **A$**. **GET** suivi par un nom de variable numérique peut être utilisé pour obtenir un nombre de un chiffre, à partir du clavier, mais donnera une erreur si une touche non numérique est enfoncée. Comme le montre cet exemple **GET** est utile pour les programmes nécessitant une simple entrée au clavier.


```
1 REM.....GET.........
5 HIRES:C=RND(1)*6+1:INK C:PAPER 0
10 FOR A=10 TO 50 STEP 10
20 CURSET 50+(A*2),96,3
30 CIRCLE 10+A,2
40 NEXT
50 PRINT"FRAPPEZ UNE TOUCHE POUR RECOM
MENCER"
60 GET A$
70 GOTO 5
```

GET diffère de la commande **KEY$**. En effet sur la commande **GET**, le programme s'arrête jusqu'à ce que l'utilisateur ait pressé une touche au clavier alors que **KEY$** recherche une entrée éventuelle mais passe à l'exécution de la ligne de programme suivante qu'une touche ait été ou non pressée.

GOSUB

Code BASIC : 155
écriture : GOSUB ln
GOSUB v

C'est l'une des commandes les plus importantes du BASIC. **GOSUB** est semblable à **GOTO** en cela qu'elle oblige l'ORIC à exécuter les instructions à partir d'un numéro de ligne au lieu de poursuivre les lignes de programme à la suite. Cependant, contrairement à **GOTO**, cette instruction conserve une adresse de retour dans la pile de mémoire de l'ordinateur. Cela signifie que **GOSUB** ln obligera le programme à sauter jusqu'au numéro de ligne ln, à exécuter toutes les opérations suivantes, jusqu'à ce que l'instruction **RETURN** (retour) soit rencontrée. A ce point, le programme retournera à l'instruction suivant la ligne contenant le **GOSUB** d'origine. Le numéro de ligne dans l'expression **GOSUB** peut aussi être une variable ou le résultat d'un calcul. La partie du programme atteinte (ou "appelée") par une instruction **GOSUB** est appelée sous-programme.

La destination de **RETURN** peut seulement être modifiée avec l'usage de la commande **POP** (voir **POP** dans ce chapitre).

Une utilisation intelligente des sous-programmes peut grandement améliorer l'efficacité et la clarté d'un programme.

Numérisation http://wsc.n3.net
William Saint-Cricq / wsaintcr@wanadoo.fr Page 81 sur 163 15/08/2000

Numérisation http://wsc.n3.net

```
1 REM......GOSUB........
10 CLS:C=0
20 PRINT"DONNEZ UN NOMBRE <22"
30 INPUT N
40 IF N>30 THEN GOSUB 250
50 IF C=1 THEN 10
60 GOSUB 100
70 PRINT F:GOTO 310
100 REM...Sous-Programme 1...
110 IF N<>1 THEN 140
120 F=1
130 GOTO 180
140 N=N-1
150 GOSUB 100
160 F=F*(N+1)
170 N=N+1
180 RETURN
190 REM...fin du sous Programme.
200 GOTO 310
250 REM...Sous-Programme 2...
260 CLS:C=1
270 PRINT"SUIVEZ LES INSTRUCTIONS !REC
OMMENCEZ"
280 WAIT 300
290 RETURN
300 REM...fin du sous Programme..
310 END
```

Mots-clés associés : GOTO, ON, RETURN.

GOTO

Code BASIC : 151
écriture : GOTO ln
GOTO v

Normalement, la suite des opérations d'un programme BASIC se fait dans l'ordre des numéros d'instruction, c'est-à-dire du plus petit au plus grand. L'utilisation de **GOTO** interrompt cette suite et envoie le contrôle des opérations au numéro de ligne précisé.

On s'inquiète en général de l'utilisation soigneuse de **GOTO**, car c'est une commande particulièrement puissante. **GOTO** est trop souvent utilisée pour raccommoder un programme mal structuré. De tels programmes deviennent difficile à suivre et les erreurs résistent à la recherche.

En bref, si vous trouvez que vous utilisez beaucoup de fois l'instruction **GOTO** dans un programme, c'est certainement qu'il est mal conçu.

Le BASIC de l'ORIC permet d'utiliser pour numéro de la ligne à laquelle **GOTO** donne le contrôle une variable. Par exemple :

```
60 GOTO A
70 GOTO A+B
```

GOTO peut aussi être utilisé comme commande directe. Par exemple, **GOTO 90** exécute le programme à partir de la ligne **90**, gardant en mémoire les valeurs primitivement assignées aux variables. En cela, il diffère de **RUN 90** qui efface les variables avant exécution.

```
10 REM......GOTO.......
20 CLS :GOTO 40
30 END
40 GOTO 1000
50 PRINT"UN PROGRAMME ";
60 GOTO 90
70 PRINT"UNE PIETRE ";
80 GOTO 200
90 PRINT"QUI MONTRE "
100 GOTO 70
200 PRINT"UTILISATION ";
210 GOTO 1020
300 PRINT"GOTO !"
310 GOTO 30
1000 PRINT:PRINT:PRINT:PRINT"VOILA ";
1010 GOTO 50
1020 PRINT"DE ";
1030 GOTO 300
```

Mots-clés associés : GOSUB, ON.

GRAB

Code BASIC : 159
écriture : GRAB

Une grande partie de l'espace mémoire de l'ORIC est réservé pour la haute résolution. Si vous composez un programme BASIC très long qui ne nécessite pas de haute résolution, il est possible de récupérer l'emplacement mémoire réservé pour la haute résolution en utilisant l'instruction **GRAB**, qui est normalement entrée comme une commande directe (**GRAB** peut cependant être utilisée dans le

William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

corps d'un programme quand une mémoire additive est nécessaire pour stocker des tableaux).

Une fois que la commande a été effectuée, vous ne pouvez plus utiliser la haute résolution tant que la section de mémoire correspondante (octets #98ØØ à #B4ØØ sur l'ORIC 48K et #18ØØ à #34ØØ sur le 16K) n'a pas été reconvertie au mode haute résolution (**HIRES**). Cela peut être réalisé par **RELEASE**, entré en commande directe.

```
 1 REM......GRAB........
 5 CLS
 10 PRINT"COMBIEN DE MEMOIRE ?":PRINT F
RE(0)
 20 PRINT"MAINTENANT,ON RECUPERE DE LA
MEMOIRE"
 30 GRAB
 50 PRINTFRE(0)
 60 PRINT"MAINTENANT,ON LA RESTITUE"
 70 RELEASE
 80 PRINTFRE(0)
 90 PRINT"COMPRIS ?"
```

Mots-clés associés : HIRES, RELEASE.

HEX$

Code BASIC : 220
écriture : HEX$(i)

Le système numérique hexadécimal est l'un des préférés par les ordinateurs. Contrairement au système décimal avec lequel la majorité des humains sont familiarisés, qui est basé sur le nombre 1Ø, le système hexadécimal opère en base 16, dont les chiffres sont

Ø,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

A-F en héxadécimal correspondent à 10 – 15 en décimal. **HEX$(i)** permet de convertir l'entier décimal entre parenthèses en son équivalent **HEX**adécimal. Ainsi **HEX$(15)** donnera #F (le signe # signifie hexadécimal). Le nombre décimal entre parenthèses peut être tout nombre entier entre Ø (#Ø) et 65535 (#FFFF). L'ORIC affichera le message "**BAD SUSCRIPT**" (écriture incorrecte) **si** vous tentez de convertir un nombre négatif ou une fraction.

Comme il est peu probable que vous utilisiez cette fonction si vous n'êtes pas familiarisé avec le système hexadécimal, il vaut mieux comprendre comment convertir un hexadécimal en décimal équivalent. Alors que le système décimal utilise la position de chaque chiffre pour représenter des puissances de 1Ø (par exemple $522 = (5*10^2) + (2*10^1) + (2*1)$, vous ne serez pas étonné d'apprendre que le système hexadécimal suit le même principe, sauf que la position de chaque chiffre indique des puissances de 16. Ainsi :

$$46E = (4*16^2) + (6*16) + (14*1) = \text{décimal } 1134$$

Mots-clés associés : STR$, VAL.

HIMEM

Code BASIC : 158
écriture : HIMEM adr

Cette commande vous donne la possibilité de réserver une partie de la mémoire de l'ordinateur pour stocker des programmes en langage machine.

Normalement, l'ORIC partage la mémoire disponible de façon aussi efficace que possible pour stocker vos programmes, leurs variables, et la mémoire nécessaire pour les affichage graphiques. Dans ces conditions, l'utilisateur n'a pas de contrôle sur la manière dont la machine partage sa mémoire. Toutefois, quand il utilise des sous-programmes en langage machine, le programmeur doit allouer une partie de la mémoire au stockage exclusif du code machine, afin qu'il soit préservé du stockage du code BASIC déterminé par l'ordinateur.

Pour cela **HIMEM** fixe l'adresse de l'emplacement mémoire le plus élevé disponible pour cette portion de programme écrit en BASIC. **HIMEM** doit être fixée au tout début du programme, et la commande doit toujours être utilisée avec un maximum de prudence.

```
 1 REM......GRAB........
 5 CLS
 10 REM......HIMEM.........
 20 PRINT"HIMEM VAUT HABITUELLEMENT"DEE
K(#A6)
 30 GRAB
```


Numérisation http://wsc.n3.net
William Saint-Cricq / wsaintcr@wanadoo.fr Page 83 sur 163 15/08/2000

Numérisation http://wsc.n3.net

```
 40 PRINT"APRES 'GRAB' ON OBTIENT  "DEE
K(#A6)
 50 RELEASE
 60 PRINT"APRES 'RELEASE' ON A     "DEE
K(#A6)
 70 HIMEM 3000
 80 PRINT"ON PEUT FIXER N'IMPORTE QUELL
E VALEUR"
 90 PRINT"PAR EXEMPLE"DEEK(#A6)
```

Mots-clés associés : GRAB, RELEASE.

HIRES

Code BASIC : 162
écriture : HIRES

Cette commande convertit l'affichage texte **(TEXT)** ou basse résolution **(LORES)** en affichage haute résolution. C'est cet affichage qu'utilise effectivement l'ORIC pour ses possibilités graphiques soignées. Quand la commande est lancée, l'affichage initial des 24 lignes devient un tableau noir de 240 × 200 points. Les trois lignes du bas de l'écran restent dans le mode texte, afin que l'utilisateur de l'ordinateur puisse afficher des messages de façon habituelle.

Comme tous les affichages graphiques, **HIRES** utilise une grande partie de la mémoire de l'ORIC, qui peut être récupérée pour un programme BASIC par la commande **GRAB**. Cependant, si cette possibilité a été utilisée, vous ne pourrez plus travailler en mode haute résolution tant que vous n'aurez pas utilisé **RELEASE** pour rendre à nouveau la mémoire disponible pour **HIRES**.

Quand vous élaborez un programme qui utilise **HIRES**, vous ne pourrez pas lister ou éditer le programme de façon normale, puisqu'il ne s'affichera pas sur l'écran en mode haute résolution. Chaque fois que vous lancerez le programme, vous devrez revenir à l'affichage mode texte **(TEXT)** avant d'étudier la liste d'instructions.

Les commandes dont la liste est ci-dessous, parmi lesquelles **FILL**, ne fonctionnent qu'en mode haute résolution, tandis que le programme qui suit utilise plusieurs d'entre elles à bon escient.


```
20 CURSET 14,10,3
30 DRAW 0,180,1:DRAW 220,0,1
40 CURMOV -210,-10,3:DRAW -19,19,1
50 FOR B=1 TO 6
60 READ H:GOSUB 100
70 NEXT B
80 GOSUB 400:END
90 REM....LIGNE D'ECRITURE....
100 X=(B*5+2)*6-5:CURSET X,190,3
110 DRAW 0,-H,1:DRAW 10,-10,1:DRAW 12,
0,1:DRAW -10,10,1
120 CURSET X+23,180-H,3:DRAW 0,H+10,1
130 CURSET X,190-H,3
150 FILL H,2,B+16
160 CURSET X+13,190-H,3:FILL H,1,16
200 RETURN
300 DATA 100,75,55,124,150,155
400 REM AFFICHER DU TEXTE AVEC 'CHAR'
410 A$="VENTES DE CARAMEL MOU"
420 FOR I=1 TO LEN(A$)
430 CURSET I*6+40,10,3
440 CHAR ASC(MID$(A$,I)),0,1
450 NEXT
460 PATTERN 51:CURSET 40,20,3:DRAW 150
,0,1
470 RETURN
```

Mots-clés associés : CHAR, CIRCLE, CURMOV, CURSET, DRAW, FILL, LORES, TEX, PAPER, INK.

IF... THEN... (ELSE)

Code BASIC : 153, 201, 200
écriture :
IF c THEN instructions ELSE instruction

L'écriture **IF... THEN... (ELSE)** est appelée structure de décision. Elle est utilisée pour vérifier des conditions et contrôler les actions de l'ORIC qui en découlent. Le **ELSE** est entre parenthèses ci-dessus (sans parenthèses s'il est utilisé dans un programme) car

15/08/2000 Page 84 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

c'est un élément facultatif de cet écriture. Les "instructions" peuvent être une suite d'instructions BASIC quelconques, tant qu'elles tiennent dans une ligne. Si la condition c est vraie, les instructions suivant **THEN** sont exécutées. Le contrôle passe alors à la ligne suivante, sauf si **ELSE** a été utilisé, auquel cas les instructions suivant **ELSE** sont effectuées.

Cette structure de décision est l'un des éléments les plus puissant du BASIC. Son emploi est très clair dans cet exemple :

```
 1 REM.....IF...THEN...ELSE......
 5 CLS
 8 N$="MADEMOISELLE "
 10 INPUT"DONNEZ VOTRE NOM";Z$
 20 CLS
 30 INPUT"ETES-VOUS UN HOMME OU UNE FEM
ME (H/F)";S$
 40 IF S$="F" THEN INPUT"ETES-VOUS MARI
EE (O/N) ";S$
 45 CLS:PRINT @ 5,10;
 50 PRINT"BONJOUR ";
 60 H$="MONSIEUR "
 70 O$="MADAME "
 80 N$="MADEMOISELLE "
 90 IF S$="H" THEN PRINT H$;ELSE IF S$=
"O" THEN PRINT O$;ELSE PRINT N$;
 100 PRINT Z$
```

GOSUB doit être utilisé en écriture ordinaire avec une syntaxe **IF... THEN... ELSE. GOTO** peut être omis, et le numéro de ligne indiqué seul, et **GOTO** peut remplacer **THEN**.

Mots-clés associés : AND, GOTO, NOT, OR, ON.

INK

Code BASIC : 178
écriture : INKi

Cette instruction fonctionne en modes haute et basse résolution. Elle définit la couleur de premier plan de l'écran entier en fonction de la valeur de i, dont les valeurs possibles sont énumérées dans le

tableau de couleurs ci-dessous :

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

INK modifie la couleur de tout ce qui est "écrit" sur l'écran ; on ne peut l'utiliser pour donner des couleurs différentes à tel ou tel caractère (voir le chapitre "graphismes" et la rubrique **CHR$** pour savoir comment obtenir ce résultat).

```
1 REM.....INK........
5 CLS
10 HIRES
20 A=0:B=0
25 HIRES
30 FOR V=0 TO 25
40 B=B+1:A=A+3
50 IF B>7 THEN B=1
60 INK B
70 CURSET 110,100,3
80 CIRCLE A,1
90 NEXT V
100 GOTO 20
```

Mots-clés associés : LORES, HIRES, PAPER, TEXT, CHR$.

INPUT

Code BASIC : 141
écriture : INPUT v, v$,...
INPUT "message", v, v$...

Cette instruction permet à l'ordinateur de recevoir des informations provenant de l'extérieur. Elle interrompt l'exécution du programme dans lequel elle figure, et qui ne se poursuit qu'une fois que l'utilisateur a *entré* (c'est le sens du mot *input*) un mot (alphabétique) ou un nombre. Il existe plusieurs présentations possibles pour cette instruction. Par exemple :

```
10 INPUT N$
```


William Saint-Cricq / wsaintcr@wanadoo.fr Page 85 sur 163 15/08/2000

interrompt le programme en faisant apparaître sur l'écran un point d'interrogation. L'utilisateur doit alors frapper l'entrée appropriée et appuyer sur la touche **RETURN**. Mais comme, en règle générale, seul le programmeur connaît de façon précise la nature de la donnée requise par le programme, il est nécessaire d'avoir recours à un message explicatif. On peut l'obtenir de deux manières différentes :

```
10 PRINT @ 13,10;"QUEL EST VOTRE NOM"
20 INPUT N$
```

ou bien

```
10 INPUT "QUEL EST VOTRE NOM ";N$
```

Le seul avantage de la première solution est de vous permettre de positionner le message à l'endroit voulu de l'écran (à condition que vous aviez un ORIC VI.I), alors que dans le second cas, le message s'affiche à la position actuelle du curseur.

Mots-clés associés : GET,KEY$.

INT

Code BASIC : 215
écriture : INT(n)

INT sert à convertir en nombre entier un nombre à décimale. Notez que le nombre obtenu en utilisant **INT** est toujours inférieur à la valeur fournie au départ. Par exemple :

```
10 Z=INT(47.667)
20 PRINT Z
```

affichera 47 à l'écran. Mais faites attention lorsque les valeurs sont négatives. Par exemple :

```
10 X=INT(-7.667)
20 PRINT X
```

donnera bien entendu : – 11.

Mots-clés associés : aucun

KEY$

Code BASIC : 241
écriture : v$ = KEY$

Cette instruction constitue un des moyens par lequel l'ORIC peut recevoir des informations provenant de l'extérieur. Comme GET, KEY$ demande une réponse frappée au clavier ; cependant, le programme se poursuit, que l'on appuie ou non sur une touche.

Comme KEY$ reçoit la valeur de la touche frappée, quelle qu'elle soit, cette instruction est précieuse dans les jeux de type "jeux d'arcade", où elle permet d'utiliser les touches curseur pour contrôler les mouvements qui se produisent à l'écran. Comme le montre l'exemple ci-dessous, KEY$ est également utile lorsqu'on attend de l'utilisateur du programme une réponse particulière :

```
 1 REM......KEY$...........
 5 CLS:X=2
 10 PRINT"Pour se deplacer a gauche...Z
"
 11 PRINT"Pour se deplacer a droite...M
"
 15 PRINT"Pour arreter................S
 20 REPEAT
 30 V$=KEY$
 40 IF V$="M" THEN X=X+3:PLOT X-3,10,"
 "
 50 IF V$="Z" THEN X=X-3:PLOT X+3,10,"
 "
 60 IF X<2 THEN X=2
 70 IF X>35 THEN X=35
 80 PLOT X,10,"<*>"
 90 UNTIL V$="S"
 100 PRINT"EXEMPLE TERMINE"
```

Mots-clés associés : GET,INPUT

Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Numérisation http://wsc.n3.net

LEFT$

Code BASIC : 244
écriture : v$ = LEFT$(a$,i)

De la même manière que **MID$ et RIGHT$**, cette instruction permet d'extraire d'une chaîne certains caractères. Sa présentation est la suivante :

LEFT$(a$,i)

a$ étant une variable définie au préalable et i étant égal au nombre de caractères consécutifs que l'on désire extraire de la partie gauche de la chaîne. Ainsi :

```
10 REM.......NEW........
20 A$="ORIC LEFT$"
30 FOR I=1 TO LEN(A$)
40 B$= LEFT$(A$,I)
50 PRINT B$
60 NEXT I
70 END
```

extraira et affichera tous les **LEFT$** possibles à partir de "**ORIC LEFT$**". Si i est supérieur au nombre de caractères de la chaîne, l'ORIC restitue l'ensemble de la chaîne.

Mots-clés associés : RIGHT$, MID$, LEN

LEN

Code BASIC : 233
écriture : LEN(a$)

Cette instruction restitue le nombre de caractères dans une chaîne donnée. Par exemple :

10 J = LEN("NOMBRE")
20 PRINT J

affichera 6 à l'écran, puisqu'il y six lettres dans la chaîne "NOMBRE".

10 H = LEN("AFFICHE A L'ECRAN")
20 PRINT H

affichera 17 à l'écran, puisque dans ce cas la chaîne comporte 15 caractères et deux espaces.

Cette instruction est utile lorsque votre programme doit afficher à l'écran des chaînes entrées par l'utilisateur du programme au moyen de **INPUT**, chaînes dont la longueur est évidemment inconnue. On peut également employer **LEN** dans une structure **FOR...NEXT**, où une tâche doit être effectuée pour chaque caractère d'une chaîne. Ainsi :

10 FOR A = 1 TO LEN(X$)

constitue une instruction valide. La chaîne vide ou nulle "" a une longueur **LEN** égale à 0.

Mots-clés associés : MID$, RIGHT$

LET

Code BASIC : 150
écriture : LET:v = n
LET v$ = a$

Nous nous sommes efforcés dans ce livre de montrer qu'il ne suffit pas de produire des programmes qui tournent : ils doivent également être lisibles pour un non-initié. Dans le BASIC Oric, **LET** est une instruction facultative, mais elle peut servir à rendre un listing plus clair.

LET sert à affecter une valeur à une variable. Par exemple :

10 LET A = 10

Cela veut dire que dans un emplacement de mémoire donné qui sera désigné ultérieurement par A, la valeur 10 sera conservée. Nous avons donc créé un monstre bizarre appelé variable constante : une variable dont la valeur reste identique. Ceci dit, nous pouvons modifier la valeur de la variable A au moyen d'une instruction telle que :

20 LET A = A + 10

William Saint-Cricq / wsaintcr@wanadoo.fr Page 87 sur 163 15/08/2000

Cela peut paraître absurde en termes mathématiques, mais en termes BASIC, cela veut simplement dire que nous ajoutons 1Ø à l'emplacement-mémoire A, auquel avait été affectée initialement la valeur 1Ø. La valeur de A est donc maintenant égale à 2Ø (1Ø + 1Ø). Le BASIC Oric nous autorise cependant à omettre **LET** :

3Ø A = A + 1Ø

mais au risque de paraître insister lourdement, répétons que son utilisation permet d'obtenir des listings plus lisibles.

Mots-clés associés : aucun

LIST

Code BASIC : 188
écriture : LIST i-i

Ce mot de commande est important à toutes les étapes de la mise au point d'un programme, car il permet au programmeur de "lister" une ligne, une série de lignes ou l'ensemble du programme. Examinons les différentes présentations possibles :

LIST

Si l'on frappe **LIST** seul, on obtient l'affichage à l'écran de l'ensemble du programme. Comme un programme de taille moyenne dépasse déjà largement les dimensions de l'écran, vous allez vouloir interrompre le défilement du programme sur l'écran pour pouvoir le consulter section par section. Il vous suffira d'utiliser la barre d'espacement située en bas du clavier. Le défilement ("scrolling") ainsi interrompu reprendra dès que vous aurez appuyé à nouveau sur la barre d'espacement.

LIST 4Ø

Vous obtenez ainsi l'affichage à l'écran de la ligne 4Ø.

LIST 4Ø-8Ø

Les lignes 4Ø à 8Ø (incluse) s'affichent à l'écran.

Mots-clés associés : EDIT,LLIST

LLIST

Code BASIC : 142
écriture : LLIST i-i

Ce mot de commande s'emploie exactement comme **LIST**, mais au lieu d'afficher à l'écran les lignes spécifiées, il les liste sur l'imprimante.

Mots-clés associés : LPRINT, LIST

LN

Code BASIC : 224
écriture : LN(n)

Il s'agit là d'une des précieuses fonctions mathématiques de l'ORIC. Elle permet d'obtenir des logarithmes népériens, ou plus exactement, des logarithmes de base e.

LN(n) est le logarithme népérien de n. Il est "neutralisé" par : **EXP(LN(n))**. Le cas échéant, on peut procéder aux opérations sur les logarithmes népériens de la même manière qu'avec les logarithmes décimaux. Par exemple :

EXP(LN(x) + LN(y))

donne le produit de x par y.

Mots-clés associés : EXP, LOG

LOG

Code BASIC : 232
écriture : LOG(n)

Cette fonction mathématique permet de calculer les logarithmes décimaux. L'argument n doit être supérieur à Ø, car dans le cas contraire, **LOG(n)** serait un nombre imaginaire. Si vous vous intéressez

Numérisation http://wsc.n3.net
William Saint-Cricq / wsaintcr@wanadoo.fr Page 88 sur 163 15/08/2000

à ces nombres, référez-vous aux ouvrages portant sur le calcul infinitésimal, mais sachez que l'Oric ne peut les traiter.

```
 1 REM.....LOG......
 5 CLS
 10 INPUT"DONNEZ UN NOMBRE ENTRE 1 ET 1
0";N
 20 FOR A=1 TO 10
 40 PRINT"LE LOGARITHME DECIMAL DE DE"N
*A
 50 PRINT"EST"LOG(N*A)
 60 NEXT
```

Mots-clés associés : EXP, LN

LORES

Code BASIC : 137
écriture : LORES 0
LORES 1

Ce mot de commande donne accès à deux des quatre écrans de l'ORIC. Les deux présentations possibles **(LORES 0 et LORES 1)** correspondent à chacun des écrans à basse résolution, mais ils ont tous les deux le même format : vingt-sept lignes de quarante caractères.

L'écran **LORES 0** est similaire à l'écran **TEXT**, à ceci près qu'il génère un écran noir sur lequel les caractères s'affichent en blanc. L'emploi normal de la déclaration **PRINT** provoque le défilement de tout l'écran jusqu'à son remplacement par l'écran **TEXT**. Les caractères doivent donc être positionnés au moyen des instructions **PRINT @** ou **PLOT**.

LORES 1 agit exactement de la même façon, mais en affichant à l'écran le deuxième jeu de caractères. Le programme suivant illustre l'utilisation des deux écrans.

```
 1 REM........LORES .......
 10 LORES0 :C=0
 20 PLOT 2,24,"ENSEMBLE DES CARACTERES
EN LORES0"
 30 FOR A = 32 TO 126
```



```
 40 X=RND(1)*36+1:Y=RND(1)*22+1
 50 C$=CHR$(A)
 60 PLOT X,Y,C$
 70 WAIT 50
 80 NEXT
 90 IF C=1 THEN WAIT 500:CLS:END
 100 PRINT:PRINT"ET MAINTENANT EN MODE
LORES1"
 110 WAIT 300:GOSUB 130
 120 GOTO 30
 130 CLS:LORES1:C=1
 140 RETURN
```

Mots-clés associés : TEXT, HIRES

LPRINT

Code BASIC : 143
écriture : LPRINT a$
LPRINT i

Ce mot de commande agit de la même manière que la déclaration **PRINT**, mais ce qui est affiché à l'écran est alors imprimé sur papier par l'imprimante. Il est cependant impossible d'utiliser une instruction **LPRINT@**. En utilisant les codes de commande énumérés à l'Annexe 1, on peut employer **LPRINT** pour réguler la sortie en direction de diverses imprimantes. On peut modifier la longueur des lignes du document imprimé en ayant recours à l'instruction **POKE** # 256,i où i représente la longueur de ligne voulue.

Mots-clés associés : CHR$, LLIST, PRINT

MID$

Code BASIC : 246
écriture : MID$(a$,i_1, i_2)

De même que **RIGHT$** et **LEFT$**, cette instruction sert à extraire d'une chaîne définie au préalable des caractères particuliers. Son nom (de l'anglais "middle", qui veut dire "milieu") pourrait prêter

Numérisation http://wsc.n3.net
William Saint-Cricq / wsaintcr@wanadoo.fr Page 89 sur 163 15/08/2000

à confusion : en fait, elle permet au programmeur d'extraire des caractères situés n'importe où dans la chaîne. Les caractères extraits constituent ce que l'on appelle une sous-chaîne commençant au caractère i_1 et ayant une longueur de i_2 caractères. Si la sous-chaîne ainsi définie est d'une longueur supérieure à celle de la chaîne, le programme affichera la chaîne entière à partir du point spécifié par i_1. Si le point i_1 n'existe pas dans la chaîne, rien ne sera affiché. Si i_2 n'est pas indiqué, l'instruction affichera le reste de la chaîne à partir du point spécifié par i_1.

Dans le programme suivant, on utilise **MID$** pour se débarrasser des espaces situés en tête d'un affichage numérique.

```
 10 REM.....MID$.....
 20 A$="    ORIC    "
 30 IF ASC(A$)=32 THEN A$=MID$(A$,2):GO
TO 30
 40 IF ASC(RIGHT$(A$,1))=32 THEN A$=MID
$(A$,1,LEN(A$)-1):GOTO 40
 50 PRINT"*"A$"*"
```

Mots-clés associés : LEFT$, RIGHT$

MUSIC

Code BASIC : 168
écriture : MUSIC c, o, n, v
c = canal 0-3
o = octave 0-7
n = note 1-12
v = volume 0-15

Il s'agit d'une des trois instructions sonores principales de l'ORIC. Comme son nom l'indique, elle permet de générer des sons musicaux. L'utilisation des instructions sonores de l'ORIC est relativement complexe ; si vous comptez profiter pleinement des possibilités multiples et élaborées qu'elles vous offrent, nous vous conseillons de vous reporter au chapitre consacré au son. Nous pouvons cependant présenter ici les paramètres de l'instruction **MUSIC**.

Elle implique quatre variables, et elle est généralement associée à l'instruction **PLAY** (qui crée la "forme" du son et détermine le

nombre de canaux son fonctionnant à un moment donné). Comme on le voit ci-dessus dans la présentation de l'instruction, c définit le canal à utiliser, tandis que o détermine quelle octave doit être utilisée parmi les huit disponibles (0-7). La troisième variable, n, détermine la note qui sera générée parmi les douze notes d'une octave donnée (1-12). Le dernier paramètre, v, détermine le volume du son produit (0-15).

A moins que vous n'ayiez une raison sérieuse de vouloir que la dernière notre de votre chef-d'œuvre musical résonne jusqu'à la fin des temps, il vous faut fermer tous les canaux en utilisant l'instruction **PLAY 0,0,0,0** pour réduire l'ordinateur au silence.

```
1 REM.....MUSIC.....
10 FOR O=0 TO 6
20 FOR N= 1 TO 12
30 MUSIC 1,O,N,10
40 PLAY 3,0,7,0
50 WAIT 30
60 NEXT N
70 NEXT O
80 PLAY 0,0,0,0
90 EXPLODE
```

Mots-clés associés : PLAY, SOUND

NEW

Code BASIC : 193
écriture : NEW

Ce mot de commande détruit le programme BASIC en cours et les variables stockées dans la mémoire de l'ORIC. Il est recommandé de frapper **NEW** avant de commencer un nouveau programme : on évite ainsi toute perturbation due à des instructions superflues laissées par le programme précédent.

Mots-clés associés : RUN

Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr Page 90 sur 163 15/08/2000

Numérisation http://wsc.n3.net

NOT

Code BASIC : 202
écriture : NOT i
NOT c

Le mot-clé **NOT** est un opérateur logique dont le rôle correspond largement à sa signification en anglais courant ou à celle du mot français "non". Il inverse la valeur VRAI(– 1) ou FAUX(Ø) assignée par l'ORIC à une expression logique. On peut l'utiliser, par exemple, dans le cadre d'une instruction IF...THEN :

IN NOT (condition) **THEN** (action)

Il est facile de voir l'utilité de ce type de construction dans un programme de jeux. Par exemple :

IF NOT DEAD THEN GOTO (étape suivante du jeu)

où DEAD constitue un indicateur ou drapeau qui détermine si le programme doit ou non entrer en séquence de fin de partie.

Cette instruction peut aussi prendre la forme **NOT i** ou **NOT c, i** étant un nombre entier et c une expression logique. Elle peut alors s'intégrer au type de construction suivant :

IF NOT X THEN X = 6

Si x était égal à Ø (FAUX), l'expression conditionnelle **NOT X** est VRAIE, et x recevra la valeur 6. **NOT 7** donnerait la réponse 248 ; en effet, dans le cadre d'une opération sur les nombres entiers, on envisagera les bits correspondants. Si un bit fait partie du nombre entier initial, le bit correspondant ne fera pas partie de la réponse, et vice versa. C'est ainsi que 7 (ØØØØØ111) devient 248 (11111ØØØ).

```
1 REM......NOT.......
5 CLS:A=1
10 INPUT "DONNEZ UN ENTIER";N
20 REPEAT
25 A=A+1
30 IF NOT (A=N) THEN PRINT A
40 WAIT 20:CLS
50 UNTIL A=N
60 PRINT A:EXPLODE
```

Mots-clés associés : AND, IF, OR

ON

Code BASIC : 18Ø
écriture : ON i GOTO $1n_1,1n_2$...
ON i GOSUB $1n_1,1n_2$...

Cette instruction doit être associée à **GOTO** ou à **GOSUB** ; elle facilite les branchements multiples dans un programme. On l'emploie fréquemment comme structure de contrôle dans les programmes du type "menu", dans lesquels l'utilisateur se voit proposer un certain nombre d'options dont les conséquences sont traitées par différentes parties du programme. Prenons par exemple les lignes de programmes suivantes :

```
1 REM....ON.......
5 CLS
10 INPUT"TAPEZ 1,2 OU 3";N
20 ON N GOTO 100,200,300
50 REM
100 PRINT"LIGNE 100 POUR N=1":GOTO 10
150 REM
200 PRINT"LIGNE 200 POUR N=2":GOTO 10
250 REM
300 PRINT"LIGNE 300 POUR N=3":GOTO 10
```

Si l'utilisateur frappe 3 à la ligne 1Ø, le programme va à la ligne 3ØØ, c'est-à-dire au troisième numéro de ligne spécifié en ligne 2Ø. Si N = 2, le programme saute au deuxième numéro de ligne, et ainsi de suite.

Si **N** est supérieur au nombre de numéros de ligne spécifiés, le programme reprend à la ligne qui suit celle qui contient l'instruction **ON...GOTO**. Signalons cependant qu'un **INPUT** négatif produira un message d'erreur. De toute façon, il est évidemment nécessaire de vérifier les **INPUT**. Les valeurs décimales sont arrondies automatiquement.

ON...GOSUB fonctionne exactement de la même manière, mais comme le programme saute à un sous-programme, une instruction **RETURN** est indispensable ; elle renvoie le programme à la ligne qui suit l'instruction **ON... GOSUB**.

Mots-clés associés : GOSUB, GOTO, RETURN

William Saint-Cricq / wsaintcr@wanadoo.fr Page 91 sur 163 15/08/2000

Numérisation http://wsc.n3.net

OR

Code BASIC : 210
écriture : i OR i
c OR c

Ce mot-clé fait partie des opérateurs logiques de l'ORIC. Pour comprendre comment fonctionne cet opérateur, il est utile d'étudier la table suivante :

VRAI OU VRAI = VRAI
VRAI OU FAUX = VRAI
FAUX OU VRAI = VRAI
FAUX OU FAUX = FAUX

Le mot anglais "or" signifie "ou" en français, et l'opérateur **OR** joue un rôle similaire à la construction "ou bien... ou bien" du langage courant, c'est-à-dire qu'il suffit qu'une des deux conditions formulées soit remplie pour qu'une réponse **VRAI (–1)** soit obtenue.

On peut également utiliser **OR**, de façon similaire à **AND**, pour opérer sur les équivalents binaires de nombres entiers, chaque paire de bits correspondants étant envisagée conjointement. Par exemle :

13 (00001101) OR 24 (00010100) donne 29 (00011101)

Essayez l'instruction suivante :

PRINT 13 OR 24

et vous devriez obtenir 29.

Une instruction **IF...THEN** comportant **OR** prend la forme suivante :

IF (A = 0) OR (B = 1) THEN 300

Si l'une ou l'autre des conditions, ou les deux, sont vraies, le programme sautera à la ligne 300.

```
1 REM.....OR.......
5 CLS
10 INPUT"AGE DU MARI          ";AM
20 INPUT"AGE DE L'EPOUSE      ";AE
30 INPUT"REVENUS DU MARI      ";RM
40 INPUT"REVENUS DE L'EPOUSE ";RE
50 IF(AM>21 AND RM>5000 )OR (AE>21 AND
RE>5000 ) THEN 100
60 PRINT"PRET REFUSE "
70 END
100 PRINT"PRET ACCORDE"
```

Mots-clés associés : AND, NOT, FALSE, TRUE

PAPER

Code BASIC : 177
écriture : PAPER i

Cette instruction définit la couleur de fond de l'écran entier. Elle fonctionne en modes haute et basse résolution, mais on ne peut l'utiliser pour définir le fond de telle ou telle partie de l'écran. L'instruction **PAPER** doit être associée à i, qui spécifie un des codes couleur suivants :

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

Pour définir la couleur de fond de sections particulières de l'écran, voir les rubriques **CHR$** et **FILL**, ainsi que le chapitre "graphismes".

```
1 REM.....PAPER...........
10 HIRES:INK 0
20 A=0:B=0
30 FORV=0 TO 25
40 A=A+1:B=B+3
50 IF A>7 THEN A=0
60 PAPER A:WAIT 20
70 CURSET 110,100,3
80 CIRCLE B,1
90 NEXT V
100 GOTO 20
```

Mots-clés associés : INK

William Saint-Cricq / wsaintcr@wanadoo.fr

PATTERN

Code BASIC : 174
écriture : PATTERN i

Il s'agit là encore d'une instruction graphique pour la haute résolution **(HIRES)**. Comme on doit l'associer aux instructions **DRAW** ou **CIRCLE**, on ne peut l'utiliser que dans le mode haute résolution.

Normalement, aussi bien **DRAW** que **CIRCLE** génèrent un affichage en ligne continue. Mais en introduisant l'instruction **PATTERN**, on décompose les lignes ainsi produites en points ou en tirets dont la disposition est déterminée par le nombre entier i compris entre Ø et 255. Le programme ci-dessous illustre la portée de **PATTERN** :

```
10 REM.....PATTERN........
20 HIRES:INK 0: PAPER 1
25 CURSET 0,0,3
30 FOR A=1 TO 65
40 PATTERN 170-A
50 DRAW  230,0,1
60 CURMOV -230,3,0
70 NEXT
```

Mots-clés associés : HIRES, DRAW, CIRCLE

PEEK

Code BASIC : 23Ø
écriture : PEEK (adr.)

L'instruction **PEEK** examine l'emplacement mémoire spécifié par adr. et restitue le contenu de cet emplacement. La valeur restituée est comprise entre Ø et 255. Par exemple, avec **PEEK (#256),** on obtient la longueur de ligne de l'imprimante (8Ø). Pour comprendre l'utilité de cette instruction, il faut voir qu'en appliquant à cette adresse l'instruction **POKE** (se reporter à la rubrique **POKE)** affectée d'une valeur différente de 8Ø, on obtient une modification de la longueur de ligne. Pour plus de détails sur l'instruction **PEEK,** voir le chapitre 5.


```
1 REM....PEEK...........
10 REPEAT
20 PRINT CHR$(20)
30 GOSUB 100
40 UNTIL FALSE
50 END
100 IF PEEK(48039)=83 THEN PRINT"MAJUS
CULES"ELSE PRINT"minuscules"
110 RETURN
```

Mots-clés associés : CALL, DEEK, DOKE, POKE, USR

PI

Code BASIC : 238
écriture : PI

Il s'agit de la constante trigonométrique. La valeur restituée est de 3.14159265. Dans le bref programme ci-dessous, on utilise **PI** pour calculer la surface d'un cercle.

```
1 REM.......PI........
5 CLS
10 FOR V= 1 TO 5
20 R=RND(1)*30
30 A=PI*(R^2)
40 PRINT"SI LE RAYON DU DISQUE EST"R
50 PRINT"SON AIRE EST"A
60 PRINT:PRINT
70 NEXT
```

PING

Code BASIC : 166
écriture : PING

Il s'agit d'un des sons préprogrammés de l'ORIC, que l'on peut utiliser comme message ou faire figurer dans un programme de jeux. Pour utiliser des **PING** multiples, il faut les associer à l'ins-

Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr Page 93 sur 163 15/08/2000

Numérisation http://wsc.n3.net

truction **WAIT**. On peut tester l'instruction à l'aide de **CTRL-G** (ou en essayant d'introduire au clavier plus de quatre-vingt caractères pour une seule instruction).

```
1 REM.......PING...........
100 PAPER 1:INK0:CLS:PRINT
110 PRINT CHR$(140)"FAITES VOTRE CHOIX
"
120 PRINT:PRINT:PRINT
130 PRINT CHR$(147)"1....JOURNAL":PING
140 WAIT 30
150 PRINT CHR$(148)"2....HOROSCOPE":PI
NG
160 WAIT 30
170 PRINT CHR$(146)"3....CALENDRIER":P
ING
```

Mots-clés associés : EXPLODE, SHOOT, ZAP

PLAY

Code BASIC : 169
écriture : PLAY t, s, e, d
t = canal timbre Ø-7
s = canal son Ø-7
e = enveloppe Ø-7
d = durée Ø-32767

PLAY fait partie des instructions sonores assez complexes de l'ORIC, qui mettent à votre disposition, une fois que vous vous y êtes habitué, une gamme de possibilités musicales et sonores largement supérieure à ce que propose la concurrence. Mais il est assez délicat d'acquérir la maîtrise complète de ces instructions, surtout si vous n'avez pas d'expérience dans le domaine musical, aussi cela vaut-il la peine de passer un bon moment à étudier le chapitre 7. Pour l'instant, nous nous contenterons de clarifier la présentation de l'instruction.

Votre ORIC est doté de trois canaux son et timbre et **PLAY** a pour fonction de déterminer la combinaison de ces canaux. Si l'on se réfère à la présentation indiquée en tête de rubrique, on voit que t et s, qui déterminent l'activation des canaux, sont compris entre Ø et 7. Les conséquences de la combinaison des canaux ne sont pleinement compréhensibles qu'après un peu d'expérimentation, mais le tableau suivant pourra ultérieurement servir de référence. La colonne de gauche représente la valeur de t ou de s, et la colonne de droite indique quelle combinaison de canaux est mise en action par cette valeur.

t ou s	combinaison de canaux
Ø	pas de canaux
1	1
2	2
3	1 et 2
4	3
5	1 et 3
6	2 et 3
7	1,2 et 3

Le troisième paramètre de **PLAY**, e, est sans doute le plus difficile à comprendre : ce nombre entier détermine la "forme" du son produit par l'ORIC. Là encore, il est conseillé de se reporter au chapitre sur le son, où l'on trouvera une explication complète de cette caractéristique. Signalons toutefois que la variable e (Ø-7) produit lorsqu'on lui donne la valeur 1 ou 2 des "enveloppes sonores" d'une longueur déterminée, tandis que les autres valeurs génèrent des sons continus de différents types. Le programme ci-dessous, espérons-le, éclaircira les conséquences des différentes présentations de **PLAY**. Enfin, d(Ø-32767) définit la durée de l'enveloppe sonore.

Quand vous utilisez dans le cours d'un programme les instructions **MUSIC** ou **SOUND**, il vous faudra désactiver les canaux son, ce que vous pourrez faire au moyen de l'instruction **PLAY Ø,Ø,Ø,Ø.**

```
1 REM.....PLAY.......
10 CLS:PRINT:PRINT
20 INPUT"FORME DU SON (1 A 7)";E
30 IF E<0 OR E>7 THEN 20
40 INPUT"DUREE DU SON (0 A 65535)";D
50 PRINT"ENVELOPPE SONORE No"E"DE DURE
E"D
60 SOUND 1,1500,0
70 PLAY 1,0,E,D
80 PRINT"ENFONCEZ UNE TOUCHE POUR ARRE
TER"
90 GET A$
100 PLAY0,0,0,0:GOTO 10
```

Mots-clés associés : MUSIC, SOUND

William Saint-Cricq / wsaintcr@wanadoo.fr Page 94 sur 163 15/08/2000

PLOT

Code BASIC : 135
écriture : PLOTx,y,n
PLOTx,y,a$

On utilise cette instruction pour positionner les caractères sur les écrans à basse résolution.

Dans la première écriture indiquée ci-dessus, n doit être une expression numérique qui restituera le caractère ASCII approprié, représenté par cette expression. Le code ASCII comportant non seulement le jeu de caractères standard, mais aussi les caractères spéciaux ou "attributs" (voir chapitre 7 et annexe 1), on peut utiliser **PLOT** pour produire à l'écran des affichages intéressants.
Par exemple :

PLOT 11,11,12:PLOT 13,11,"ORIC"

fera apparaître sur l'écran un "ORIC" clignotant. L'expérimentation vous révélera certainement les possibilités de cette instruction ainsi utilisée, et le chapitre sur les graphismes donne une explication complète de la façon la plus efficace de tirer profit des attributs.

Utilisé avec la deuxième présentation, **PLOT** positionne la chaîne de caractères sur l'écran. Qu'il s'agisse de chaînes de caractères ou d'expressions numériques, les coordonnées se situent entre Ø et 39 pour les x et Ø et 26 pour les y. Quand x = Ø et y = Ø, **PLOT** positionne le caractère à la première position de caractère en haut à gauche de l'écran. L'instruction fonctionne dans les mode **TEXT, LORES Ø** et **LORES 1**, mais pas sur l'écran **HIRES**.

```
 1 REM ......PLOT ......
 5 CLS:LORES0
 10 PLOT 2,2,"UTILISATION DE 'PLOT' EN
MODE LORES0"
 20 FOR A=1 TO 23
 30 X=RND(1)*33+4:Y=RND(1)*22+3
 40 PLOT X,Y,A:PLOT X+2,Y,"ORIC"
 50 WAIT 50:NEXT
```

Mots-clés associés : CHAR, PRINT

POINT

Code BASIC : 243
écriture : POINT(x,y)

Cette instruction permet de tester un point de l'écran **HIRES** spécifié par les coordonnées qui la suivent (x entre Ø et 239, y entre Ø et 199) en indiquant s'il a la couleur du fond ou du premier plan. Si le point détient la couleur du fond, l'instruction **POINT** fournit la réponse Ø, et s'il a la couleur du premier plan, elle indique – 1.

POINT s'utilise fréquemment dans les programmes de jeux pour détecter les collisions et le programme ci-dessous donne un exemple simple du rôle joué par cette instruction dans ce contexte.

```
1 REM......POINT........
5 HIRES:PAPER4:INK0
10 FOR B=5 TO 175
20 CURSET 200,B,2:CHAR 124,0,2
30 NEXT B
40 A=5
50 REPEAT
60 A=A+8
70 CURSET A,90,0
80 CHAR 127,0,1:WAIT 5
90 CHAR 127,0,0
100 C= POINT(A+11,90)
120 UNTIL C=-1
130 EXPLODE
```

Mots-clés associés : HIRES

POKE

Code BASIC : 185
écriture : POKE adr,i

Cette instruction permet au programmeur de mettre la valeur de i dans l'emplacement mémoire représenté par l'adresse adr. L'adresse est comprise entre Ø et 65535, et puisque chaque emplacement comporte huit bits la valeur de i se situe entre Ø et 255. Adr. et la valeur de i peuvent être donnés en notation décimale ou hexdécimale.

Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr Page 95 sur 163 15/08/2000

Numérisation http://wsc.n3.net

Avec **POKE**, vous pouvez entrer des programmes en code machine et bricoler les variables système de votre ORIC. Et même si les conséquences d'un **POKE** inconsidéré peuvent paraître désastreuses, autant qu'on puisse en juger, il ne risque d'en découler pour votre machine aucun dégât irréversible. Le programme suivant, destiné à modifier l'affichage de la ligne d'état, vous donnera un exemple de la puissance de cette instruction.

```
10 REM.....POKE......
20 FOR A=1 TO 8
30 POKE 46599+A,A
40 WAIT 100
50 NEXT
60 WAIT 1000
70 CALL DEEK(#FFFA)
```

Quant à l'instruction ci-dessous, elle affichera un "A" dans le coin supérieur gauche de l'écran, normalement inaccessible :

POKE 48000,65

Mots-clés associés : DEEK, DOKE, PEEK

POP

Code BASIC : 134
écriture : POP

Cette instruction agit sur **GOSUB...RETURN** exactement comme **PULL** sur une boucle **REPEAT... UNTIL.** A bien des titres, il faut considérer **POP** comme une solution de dernier recours, puisqu'elle force un programme à sortir d'un sous-programme imbriqué, ce qui ne devrait jamais être nécessaire dans un programme correctement organisé et structuré.

En cas d'appel de sous-programme, l'adresse vers laquelle doit se faire le **RETURN** est normalement stockée dans une zone de la mémoire connue sous le nom de pile. Chaque nouveau sous-programme ajoute une adresse à la pile, et chaque fois qu'un sous-programme arrive à un **RETURN** il récupère son adresse de retour sur le mode **LIFO** : "dernier entré premier sorti". L'instruction **POP** est le seul moyen légal de se soustraire à ce fonctionnement.

Imaginez un programme appelant un sous-programme 1, lequel appelle à son tour un sous-programme 2. Dans certaines circonstances, lorsqu'un ensemble de conditions spécifiques sont remplies dans le sous-programme 2, il se peut qu'il faille revenir au programme principal, et non au sous-programme 1. L'instruction **POP** vous permet de le faire : elle contourne en effet l'adresse de retour placée en première position sur la pile de l'ORIC, et dans l'exemple proposé, récupère l'adresse de retour du sous-programme 1, qui remettra en action le programme principal. Le programme ci-dessous illustre le déroulement de cette procédure.

```
10 REM.....POP.....
20 FOR I=-5 TO 5:PRINTI,
30 GOSUB 100
40 PRINT:NEXT
50 END
100 GOSUB 200
110 PRINT X
120 RETURN
200 IF I>0 THEN X=LOG(I) ELSE POP
210 RETURN
```

POS

Code BASIC : 219
écriture : POS(0)
POS(1)

Utilisée pour détecter la position du curseur résultant d'une instruction **PRINT** directe, cette instruction indique la position horizontale du curseur. On ne peut l'utiliser pour obtenir les positions résultant de **PLOT** ou de **PRINT@** . **POS(0)** donne la position du curseur sur l'écran tandis que **POS(1)** donne la position horizontale de la tête d'impression lors de l'exécution d'une instruction **LPRINT**.

```
1 REM.....POS......
80 PRINT "0";
90 REPEAT:PRINT" ";:UNTIL POS(0)=15
100 PRINT  "* ICI 15"
```

Mots-clés associés : PRINT

William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Numérisation http://wsc.n3.net

PRINT

Code BASIC : 186
écriture : PRINT plist
PRINT@ x,y ; plist
plist = liste des éléments "printés"

Les instructions **PRINT** agissent sur la position du curseur à l'écran, cette position définissant le point à partir duquel se fera l'affichage. **PRINT** (que l'on peut remplacer par ?) est suivi d'une liste d'éléments à afficher sur l'écran. **PRINT@** x,y ; définit par deux coordonnées une position d'affichage sur les écrans **TEXT** et **LORES**.

PRINT peut s'utiliser seul, sans le faire suivre d'éléments à afficher ; une ligne vide apparaît alors à l'écran. La liste qui suit **PRINT** peut comporter des nombres, des variables numériques, des expressions, des chaînes et des expressions de chaîne qui peuvent soit se suivre directement, soit être séparées par des virgules et des points-virgules. On peut également inclure les instructions d'affichage **TAB** et **SPC** (voir les rubriques correspondantes dans ce chapitre).

Lorsqu'on place un point-virgule après les éléments affichés par **PRINT**, la position d'affichage reste fixée juste après le dernier caractère, et l'élément qui suit s'affiche donc à la suite. Si le point-virgule est à la fin d'une liste régie par l'instruction **PRINT**, toute nouvelle instruction **PRINT** continuera à afficher sur la même ligne. Quand il n'existe aucune ambiguïté dans la séquence à afficher, on peut omettre le point-virgule, et chaque élément apparaîtra sur l'écran immédiatement à la suite du précédent (mais rappelez-vous que les nombres sont précédés et suivis d'un espace). On peut donc avoir une ligne **PRINT AX"TIMES"23** sans en séparer les éléments par des points-virgules, puisque l'ORIC interprètera les éléments correctement ; par contre, on ne peut pas dire **PRINT AX12** si l'on veut afficher la valeur affectée à la variable **AX**, puis 1, puis 2 : en effet, l'ORIC interprétera cette instruction comme s'il s'agissait d'afficher la valeur d'une variable appelée **AX12.** Puisque le résultat est le même, il est conseillé d'utiliser un point-virgule sauf si vous êtes certain que l'ORIC ne fera aucune erreur d'interprétation sur la séquence placée sur la ligne.

Si l'on met une virgule entre les éléments suivant un **PRINT**, la position du curseur se déplace jusqu'au début du nouveau champ d'affichage. L'écran de l'ORIC est divisé dans le sens de la largeur en huit champs dont chacun couvre huit positions d'affichage, et la virgule a pour effet de déplacer le curseur d'au moins un espace vers la droite, et de là jusqu'au début du champ suivant, ce qui garantit que les éléments ainsi présentés seront séparés par au moins un espace. Pour opérer un déplacement supérieur à la dimension d'un champ, on peut utiliser plusieurs virgules.

On peut placer des caractères de fonction et des attributs dans une instruction **PRINT** en utilisant **CHR$(i)**, où i est le code **ASCII** du caractère. La même instruction peut servir à placer d'autres caractères sur les écrans **TEXT** et **LORES**, mais les codes de fonction et les attributs n'affichent rien à l'écran ; ils occupent simplement une position dans la zone de mémoire correspondant à l'écran de sorte qu'un espace apparaît à l'écran.

Vous trouverez dans ce manuel de nombreux exemples d'utilisation de **PRINT.** Le programme ci-dessous comporte différents types de présentation de cette instruction.

```
10 REM.....PRINT
20 B=2
30 A$="C'EST LA VIE"
40 PRINT 1,2,3
50 PRINTA$
60 PRINT 2*B OR NOT 2*B
70 PRINT
80 PRINT 1;2;3
```

Les instructions **PRINT@** prennent la forme **PRINT@** x,y; x étant un numéro de colonne compris entre Ø et 39, de gauche à droite sur l'écran, et y étant un numéro de ligne compris entre Ø et 27, de haut en bas sur l'écran. Les deux valeurs sont séparées par une virgule et doivent être suivies par un point-virgule précédant la liste d'éléments à afficher, disposée de la même manière qu'avec un **PRINT.** x et y peuvent être des variables ou des expressions à calculer ; elles seront tronquées (arrondies à la valeur inférieure) s'il ne s'agit pas de nombres entiers. Si les valeur ne sont pas comprises entre les limites indiquées ci-dessus, vous recevrez un message d'erreur **:ILLEGAL QUANTITY** (quantité illégale).

Le programme donné en exemple dessine un cercle en utilisant les fonctions **SIN** et **COS** pour calculer les positions d'affichage x et y.

```
10 REM.....PRINT @ ...........
20 R=13:XC=20:YC=13:CLS
30 FOR A=0 TO 2*PI STEP PI/50
40 X=XC+COS(A)*R
50 Y=YC+SIN(A)*R
60 PRINT @ X,Y;"O"
70 NEXT
```

Mots-clés associés : LPRINT, SPC, TAB

15/08/2000 Page 97 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

PULL

Code BASIC : 136
écriture : PULL

Cette instruction agit sur une boucle **REPEAT... UNTIL** exactement comme **POP** sur un **GOSUB... RETURN**. Pour parler franc, on doit considérer aussi bien **POP** que **PULL** comme des instructions de "rapiéçage" d'un programme mal bâti. Si vous avez organisé et structuré correctement votre programme,vous n'aurez jamais à vous échapper d'une boucle ou d'un sous-programme. Mais personne n'est parfait ; et **PULL** peut quand même vous permettre de vous sortir d'un mauvais pas.

Supposons que vous ayez programmé une boucle **REPEAT** destinée à afficher les nombres compris entre – 5 et 10 ainsi qu'un compte à rebours de chaque nombre jusqu'à zéro (voir le programme ci-dessous). **PULL** nous servira à sortir de la boucle intérieure dans le cas des nombres négatifs, puisque la décrémentation ne les amènera évidemment jamais à zéro. (Cet exemple, bien sûr, est quelque peu artificiel, puisqu'il serait évidemment plus simple d'utiliser l'instruction **IF** pour éviter le recours à la boucle **REPEAT**).

```
10 REM.....PULL.....
20 A=9
30 REPEAT
40 :    B=A
50 :    REPEAT
60 :    :    PRINT B;
70 :    :    B=B-1
80 :    :    IF B<0 THEN PULL:GOTO 100
90 :    UNTIL B=0
100 :   A=A-1
110 :   PRINT
120 UNTIL A=-5
```



```
 9  8  7  6  5  4  3  2  1
 8  7  6  5  4  3  2  1
 7  6  5  4  3  2  1
 6  5  4  3  2  1
 5  4  3  2  1
 4  3  2  1
 3  2  1
 2  1
 1
 0
-1
-2
-3
-4
```

Mots-clés associés : POP, REPEAT, UNTIL

READ

Code BASIC : 149
écriture : READ v,v...
READ v$,v$...

On utilise toujours **READ** avec l'instruction **DATA,** et il faut veiller à ce que **READ** soit suivi par une variable appropriée (v pour les données numériques et v$ quand les données entrées en **DATA** sont des chaînes de caractères). L'instruction "lit" de façon séquentielle les données entrées en **DATA**, et le programme ne doit jamais lui demander de lire plus de **DATA** que n'en comportent les instructions correspondantes (cela provoquerait un message d'erreur : **OUT OF DATA**, c'est-à-dire "manque de données"). La place de l'instruction **READ** dans le programme est importante, alors que les instructions **DATA** peuvent être n'importe où.

Une fois que toutes les données déclarées ont été lues, le pointeur interne de l'ordinateur peut revenir au début de la ligne **DATA** au moyen de l'instruction **RESTORE**.

```
1 REM.....READ.........
5 CLS
10 FOR A=1 TO 5
20 READ V,V$
30 PLOT V,10,V$
40 NEXT
50 DATA 5,HOMME,11,FEMME,17,ENFANT,24,
CHIEN,30,MAISON
```

Mots-clés associés : DATA, RESTORE

William Saint-Cricq / wsaintcr@wanadoo.fr Page 98 sur 163 15/08/2000

Numérisation http://wsc.n3.net

RECALL

Code BASIC : 131
écriture :
RECALL v,"nom fichier" (,S)
RECALL v$,"nom fichier" (,S)
RECALL v%,"nom fichier" (,S)

Cette instruction est associée à **STORE**, qui stocke un tableau dans un fichier sur cassette magnétique. Avec **RECALL**, on peut recharger en mémoire le contenu d'un tableau sauvegardé sur cassette au moyen de **STORE** ; la procédure à suivre est la même qu'avec **CLOAD**. Avant d'utiliser **RECALL**, il faut déclarer ("dimensionner") le tableau destiné à recevoir le tableau ainsi rappelé, sous peine de voir apparaître un message d'erreur **OUT OF DATA.** La variable du tableau doit être du même type que celle du tableau originel (nombre entier, chaîne ou réelle) et d'une taille identique ou supérieure.

Pour obtenir une transmission lente des données stockées sur la cassette, il suffit d'ajouter à l'instruction une virgule suivie d'un S (,S). Ce type de transmission doit avoir été également utilisé avec l'instruction **STORE.** On trouvera à la rubrique **STORE** un programme illustrant l'emploi de ces instructions.

Mots-clés associés : CLOAD, CSAVE, DIM, STORE

RELEASE

Code BASIC : 160
écriture : RELEASE

Une fois que le mot de commande **GRAB** a été utilisé pour libérer la zone de mémoire allouée à l'affichage **HIRES**, afin de pouvoir entrer un long programme en **BASIC**, on ne peut utiliser cet affichage tant qu'on n'a pas rétabli au moyen de **RELEASE** l'organisation normale de la mémoire. Cette instruction restitue l'affectation d'origine des octets 9800 à B400 (sur l'ORIC 48K) ou des octets 1800 à 3400 (sur le modèle 16K).

Mots-clés associés : HIRES, GRAB

REM

Code BASIC : 157
écriture : REM

L'instruction **REM** représente un outil de travail indispensable pour tout programmeur sérieux, quel que soit son niveau d'expérience, bien que son absence ou sa présence dans un programme n'ait aucune conséquence sur l'exécution du programme. **REM** veut dire **REM**arque en abrégé, et l'instruction **REM** permet au programmeur d'ajouter, au fil d'un programme, des commentaires qui expliquent la fonction de tel ou tel groupe d'instructions. L'ORIC ne tient pas compte de l'ensemble de la ligne en tête de laquelle est placée une instruction **REM.**

On peut considérer que les programmes très courts s'expliquent d'eux-mêmes, et que les **REM** n'y sont donc pas nécessaires. Nous vous conseillons cependant de prendre dès le début l'habitude de les utiliser, même si vous ne tapez que quelques lignes de programme. Dans le cours de l'élaboration d'un programme, on a souvent l'impression qu'il est parfaitement clair, mais lorsque, plus tard, vous voudrez le reprendre, vous vous épargnerez des heures de travail inutile si vous avez pris la peine d'y inclure sous forme de **REM** des commentaires qui peuvent d'ailleurs être succincts. Vous pouvez remplacer cette instruction par une apostrophe si vous vous lassez de taper **REM** à chaque fois que vous avez besoin d'un pense-bête.

```
10 REM......REM...........
20 'l'apostrophe est interpretee
30 ' comme l'instruction REM
40 ' Pour utiliser ' dans une ligne
70 ' utiliser CHR$(39)
80 PRINT "'"
90 PRINT CHR$(39)
```

Mots-clés associés : aucun

William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

REPEAT

Code BASIC : 139
écriture : REPEAT

Quand elle est associée à **UNTIL**, cette instruction crée une boucle qui force l'ORIC à répéter une série d'instructions jusqu'à ce qu'une condition donnée ait été remplie. A la différence des boucles **FOR...NEXT**, il n'y a pas de compteur à incrémenter ; si un compteur est nécessaire, il faut le programmer à l'intérieur de la boucle (comme dans l'exemple ci-dessous).

Si vous commettez le péché capital de sortir d'une boucle **REPEAT...UNTIL** par une instruction **GOTO** (pratique extrêmement mal vue de tous les programmeurs à l'exception des plus larges d'esprit), vous devrez impérativement revenir à la boucle, sous peine de supporter les conséquences d'une pile altérée. Le seul moyen de sortir d'une boucle **REPEAT...UNTIL** avant qu'elle soit achevée est le recours à l'instruction **PULL** (voir la rubrique correspondante dans ce chapitre).

```
10 REM.....REPEAT...UNTIL........
15 HIRES:INK4:PAPER0
20 A=0
25 REPEAT
40 CURSET 55+A,110-A,3
50 DRAW 2+A,0,1:DRAW 0,A*2,1
55 A=A+1
60 UNTIL A>60
```

Mots-clés associés : FOR, NEXT, PULL, UNTIL

RESTORE

Code BASIC : 154
écriture : RESTORE

Il arrive que des données introduites par **DATA** doivent être lues plus d'une fois dans le cours du même programme. Mais chaque fois qu'une donnée est lue par l'ORIC, le pointeur interne de l'ordinateur se déplace le long de l'instruction jusqu'à ce que toutes les données soient épuisées. Pour que le pointeur revienne au début de l'instruction et que les informations qu'elle contient soient lues à nouveau, il faut utiliser **RESTORE** si l'on ne veut pas recevoir un message d'erreur **"OUT OF DATA"**. Le BASIC ORIC ne permet pas de reprendre la lecture à un numéro de ligne particulier.


```
1 REM......RESTORE.......
5 HIRES:PAPER 1:INK 0
10 C=1
20 FOR A=1 TO 5
25 IF C=0 THEN WAIT 20:SHOOT
30 READ V
40 CURSET V,40,3
50 FOR B=1 TO 18 STEP 3
60 CIRCLE B,C
70 NEXT B
80 NEXT A
90 IF C=0 THEN END
100 RESTORE
110 C=0
120 GOTO 20
130 DATA 28,73,118,163,208
```

Mots-clés associés : DATA, READ

RETURN

Code BASIC : 156
écriture : RETURN

Cette instruction doit figurer à la fin de tout sous-programme. Elle indique à l'ORIC qu'il a atteint la fin d'un sous-programme et qu'il doit revenir à la ligne qui suit l'instruction où se trouve le **GOSUB** correspondant.

On ne peut modifier la destination de **RETURN** qu'en utilisant l'instruction **POP**.

```
1 REM......RETURN
10 CLS
20 PRINT"TAPEZ LE RAYON DU CERCLE"
30 INPUT R
40 C=2*PI*R:Z=C
50 GOSUB 200
60 CLS
70 PRINT"LA LONGUEUR DU CERCLE EST"Z
```


Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Numérisation http://wsc.n3.net

```
80 A=PI*(R^2):Z=A
90 GOSUB 200
100 PRINT"L'AIRE DU DISQUE EST"Z
110 GOTO 300
200 'S.P. arrondit a 2 decimales
210 Z=INT(100*(Z+.005))
220 Z=Z/100
230 RETURN
240 '.....fin du sous-Programme.....
300 END
```

Mots-clés associés : GOSUB, POP

RIGHT$

Code BASIC : 245
écriture : RIGHT$(a$,i)

Comme **MID$** et **LEFT$**, cette instruction extrait d'une chaîne certains caractères. Sa présentation est la suivante :

RIGHT$(a$,i)

a$ étant une variable chaîne définie au préalable et i étant égal au nombre de caractères consécutifs que l'on désire extraire de la partie droite de la chaîne. Par exemple :

```
10 REM...... RIGHT$ .........
20 A$="ORIC RIGHT$"
30 FOR I=1 TO LEN(A$)
40 B$=RIGHT$(A$,I)
50 PRINT SPC(LEN(A$)-LEN(B$));B$
60 NEXT I
70 END
```

extraira de la chaîne **"ORIC RIGHT$"** le nombre de caractères spécifié, en allant de la droite vers la gauche. Si i est supérieur au nombre total de caractères de la chaîne en question, l'ORIC restituera l'ensemble de la chaîne.

Mots-clés associés : LEFT$, MID$

RND

Code BASIC : 223
écriture : RND(i)

Les jeux sur ordinateur seraient bien monotones sans la fonction **RND**, puisque c'est elle qui fait intervenir le générateur de nombres aléatoires intégré à l'ORIC. Il s'agit là en fait d'une fonction pseudo-aléatoire : en effet, elle produit une séquence de nombres compris entre Ø (qu'elle peut égaler) et 1 (auquel elle ne parvient jamais tout à fait).

RND fonctionne de différentes façons suivant la valeur du paramètre indiqué entre parenthèses. Son usage normal est **RND(1)**, qui produit un nombre aléatoire compris entre Ø et 1. Telle quelle, cette présentation n'a pas grand intérêt, comme vous le verrez peut-être en tapant à plusieurs reprise **PRINT RND(1)** en mode commande. Les nombres aléatoires que nous voulons générer doivent couvrir une fourchette plus large, ce qui nous permettra d'utiliser **RND** pour simuler un jeu de dés, ou une coordonnée x comprise entre 3 et 27, et ainsi de suite.

Pour ce faire, on multiplie le résultat de **RND(1)** par le facteur approprié, ce qui produit, par exemple, un nombre compris entre Ø et 5.999999 si l'on a multiplié par 6. En ajoutant 1 et en convertissant le résultat en nombre entier, on obtient le résultat d'un coup de dé. Le programme ci-dessous montre deux façons équivalentes d'arrondir à la valeur entière inférieure. La première utilise **INT**, et la deuxième affecte simplement le résultat de **RND(1)*6+1** à la variable entière N%, qui prend automatiquement la valeur entière inférieure.

```
1 REM ....RND. ET VALEUR ARRONDIE
10 LET A=RND(1)*6+1
20 N%=A
30 R=INT(A)
40 PRINTN%;R;A
45 WAIT 50
46 IF KEY$<>"" THEN END
50 GOTO 10
```

Le résultat de **RND(Ø)** est plus prévisible. On obtient la valeur du dernier nombre aléatoire généré.

15/08/2000 Page 101 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

```
1 REM ....RND,MAIS PAS HASARD....
10 LET A=RND(0)*6+1
20 N%=A
30 R=INT(A)
40 PRINTN%;R;A
45 WAIT 50
46 IF KEY$<>"" THEN END
50 GOTO 10
```

Il est parfois utile de générer la même séquence de nombres aléatoires. On introduit la séquence de nombres en un point spécifique en donnant au paramètre de **RND** une valeur donnée négative. Le programme ci-dessous montre qu'en donnant comme valeur de départ au générateur **RND(– 4)**, on obtient la même séquence de nombres.

```
1 REM .RND..GERME DE HASARD......
2 CLS
5 REPEAT
10 GERME=RND(-4)
15 FOR I=1 TO 20
20 LET A%=RND(1)*1000
30 PRINT A%,
40 WAIT 30
50 NEXT:PRINT
60 PING
80 UNTIL KEY$<>""
```

Mots-clés associés : aucun

RUN

Code BASIC : 152
écriture : RUNln
RUN

Utilisé seul, en mode commande, **RUN** déclenche l'exécution du programme BASIC qui se trouve dans la mémoire de l'ORIC. **RUN** ln, où ln est un numéro de ligne, fait commencer l'exécution du programme à la ligne spécifiée. Si cette ligne ne figure pas dans le programme, l'ORIC envoie un message d'erreur **(UNDEFINED STATEMENT ERROR)**. **RUN** peut également être utilisé en mode programme, comme le montre l'exemple ci-dessous :

```
1 REM......RUN........
5 FORL=1TO40 :POKE 47999+L,32:NEXT
10 A$="ENFONCEZ UNE TOUCHE POUR ARRETER":GOSUB 70
20 CLS:PAPER 1
30 X=RND(1)*32+1:Y=RND(1)*20+1
40 PLOT X,Y,"ORIC":WAIT 100
50 V$=KEY$
60 IF V$ THEN END ELSE RUN
70 Z=LEN(A$):FOR L=1 TO Z
75 REM....AFFICHE SUR LIGNE SUP....
80 POKE 47999+L,ASC(MID$(A$,L,1))
90 NEXT L
100 RETURN
```

Mots-clés associés : CONT, END, STOP

SCRN

Code BASIC : 242
écriture : SCRN(x,y)

L'instruction **SCRN** fournit le code **ASCII** du caractère situé à la position d'écran définie par les coordonnées x (numéro de colonne Ø-38) et y (numéro de ligne Ø-26). SCRN ne fonctionne que dans les modes **LORES** et **TEXT**. x et y peuvent être des expressions numériques, mais leurs valeurs doivent être comprises dans les limites indiquées, sans quoi on obtient un message d'erreur "**ILLEGAL QUANTITY**".

Les exemples simples ci-dessous illustrent l'utilisation de **SCRN**. Le premier utilise **PRINT@** et le deuxième **PLOT**. Le signe # est affiché sur l'écran à l'emplacement 5,5 par ces deux méthodes et la ligne 2 utilise **SCRN** pour afficher d'abord (par **PRINT**) le code **ASCII** de #, après quoi la même expression **SCRN** est employée avec **CHR$** pour afficher le caractère qui se trouve effectivement dans l'emplacement mémoire de l'écran correspondant à la colonne x, ligne y.

```
1 REM.......SCRN...........
5 CLS
10 PRINT@ 5,5;"#"
20 PRINT SCRN(5,5);CHR$(SCRN(5,5))
```


Numérisation http://wsc.n3.net
William Saint-Cricq / wsaintcr@wanadoo.fr Page 102 sur 163 15/08/2000

Numérisation http://wsc.n3.net

```
1 REM.......SCRN...........
5 CLS
10 A$="#"
15 PLOT 5,5,A$
20 PRINT SCRN(5,5);CHR$(SCRN(5,5))
```

SCRN fonctionne également avec des caractères re-définis en renvoyant le code ASCII du caractère qui a été re-défini. Dans le programme ci-dessous, un astérisque joue le rôle d'un missile lancé après mise à feu vers le haut de l'écran, guidé vers la gauche et vers la droite avec les touches de commande du curseur appropriées, jusqu'à ce qu'il atteigne une des cibles (formées par les caractères re-définis !) disposées de gauche à droite de l'écran (quand **SCRN**(x,y) = 33, 33 étant le code de !).

```
 10 REM......SCRN............
 15 PLOT 5,5,A$
 20 CLS:FOR I=1 TO 8
 30 READ A:POKE 46343+I,A
 40 NEXT:POKE #24E,1:POKE #24F,1
 50 PRINT@2,0;"TIR:barre d'esp.
DEPL..fleches";
 60 FOR I=1 TO 10:PRINT@ INT(RND(1)*37)
+2,1;"!";
 70 NEXT:PRINT CHR$(17);CHR$(6)
 80 REPEAT
 90 REPEAT:UNTIL KEY$=" " OR KEY$=CHR$(
13):IF KEY$=CHR$(13) THEN 200
 100 X=20:Y=26
 110 PRINT@ X,Y;"*";:SHOOT
 120 REPEAT:OX=X
 130 IF KEY$=CHR$(8) AND X>2 THEN X=X-1
 135 WAIT 1
 140 IF KEY$=CHR$(9) AND X<39 THEN X=X+
1
 150 Y=Y-1
 160 PRINT@OX,Y+1;" ";
 170 IF SCRN(X,Y)=33 THEN PRINT@X,Y;" "
:EXPLODE:Y=1:GOTO 190
 180 PRINT@ X,Y;"*"
 190 UNTIL Y=1:PRINT@X,Y;" "
 200 UNTIL KEY$=CHR$(13)
 210 POKE #24E,32:POKE #24F,4:PRINT CHR
$(17);CHR$(6)
 400 DATA 30,33,45,63,45,33,30,0
```

Mots-clés associés : ASC, PLOT, PRINT

15/08/2000

SGN

Code BASIC : 214
écriture : SGN(n)

SGN est une fonction numérique qui donne le signe d'un nombre en le représentant par la valeur – 1 s'il est négatif, Ø si le nombre est égal à Ø et 1 s'il est positif.

```
 10 REM .......SGN...........
 15 CLS
 20 INPUT"DONNEZ UN NOMBRE";N
 30 PRINT"VOTRE NOMBRE EST"N
 40 ON SGN(N)+2 GOSUB 100,200,300
 50 PRINT:PRINT"APPUYEZ SUR UNE TOUCHE
POUR UN AUTRE  ESSAI"
 60 GETT$
 70 GOTO 15
 100 PRINT"C'EST UN NOMBRE NEGATIF":RET
URN
 200 PRINT"C'EST LE NOMBRE ZERO":RETURN

 300 PRINT"C'EST UN NOMBRE POSITIF":RET
```

Le programme donné en exemple utilise **SGN** pour tester le nombre entré en **INPUT**, et utilise la valeur obtenue plus 2 avec l'instruction **ON...GOSUB** de la ligne **4Ø** qui envoie au sous-programme approprié destiné à afficher le signe du nombre.

Mots-clés associés : ABS

SHOOT

Code BASIC : 163
écriture : SHOOT

SHOOT et une instruction sonore préprogrammée qui produit le bruit d'un coup de fusil. Pour utiliser **SHOOT** dans un programme, il suffit d'une ligne comme :

1Ø SHOOT

Page 103 sur 163

William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

Pour utiliser des instructions **SHOOT** multiples, comme dans le cas des autres instructions sonores préprogrammées (sauf **ZAP**), il faut se servir de **WAIT** pour donner au bruit le temps d'être produit et éviter que les bruits successifs ne se recouvrent. Essayez, par exemple :

```
10 FOR A=1 TO 6
20 SHOOT
30 NEXT
```

Le son produit n'est pas différent de celui que donnerait une seule instruction **SHOOT**, parce que la boucle s'achève avant la fin du premier bruit. Ajoutez la ligne :

```
10 FOR A=1 TO 6
20 SHOOT
25 WAIT25
30 NEXT
```

et un six-coups tirera une salve où chaque coup sera bien distinct.

Mots-clés associés : EXPLODE, PING, ZAP

SIN

Code BASIC : 227
Présentation : SIN(n)

SIN est une fonction trigonométrique qui donne le sinus de l'angle dont la valeur est indiquée par n, n étant exprimé en radians. La fonction calcule la base du rapport trigonométrique, qui donne pour un triangle rectangle, comme on le voit sur la figure ci-dessous, le sinus de l'angle **A (SIN(A))** comme la longueur du côté opposé divisée par la longueur de l'hypoténuse, soit **BC/AB**.

L'expression numérique évaluée par la fonction **SIN** doit être exprimée en radians et non en degrés. Comme la mesure angulaire va de 0 à 360 degrés, la mesure en radians va de 0 à 2***PI** radians. La conversion des degrés en radians et vice versa se fait de façon simple en utilisant la fonction intrinsèque **PI** de l'ORIC :

x degrés = x***PI**/180 radians
x radians = x/**PI***180 degrés

Le programme ci-dessous affiche les valeurs du sinus des angles de 0 à 360, de dix en dix degrés. La ligne 30 convertit les degrés en radians ; la ligne 40 affiche la valeur de l'angle et le résultat de l'application de la fonction **SIN** à la valeur en radians.

```
10 REM.......SIN..........
20 FOR ANGLE=0 TO 360 STEP 10
30 LET RADIANS=ANGLE*PI/180
40 PRINTANGLE,SIN(RADIANS)
50 NEXT
```

Si vous étudiez attentivement le résultat, vous constaterez que les valeurs varient entre 0 et 1 sans atteindre ni l'une ni l'autre de ces limites, en raison des imprécisions dues aux multiples calculs effectués. Remplacez la ligne 40 suivant l'exemple ci-dessous, qui arrondit le résultat à la quatrième décimale, pour avoir un tableau plus précis des valeurs de **SIN** :

```
10 REM....Fonction SIN......
20 FOR ANGLE=0 TO 360 STEP 10
30 LET RADIANS=ANGLE*PI/180
40 PRINT ANGLE,(INT(SIN(RADIANS)*1E4+0
.5)/1E4)
50 NEXT
```

Le programme suivant utilise **TAB** associé à **SIN** pour calculer la position d'affichage d'une astérisque qui produit une sinusoïde à mesure que l'écran défile. Comme la valeur donnée par **SIN** varie entre 0 et 1, l'expression 18 **SIN(T)** à la ligne 20 donne une valeur comprise entre −18 et +18, qui se combine à la position donnée par **TAB** pour varier le positionnement de gauche à droite de l'écran.

```
10 FOR T=0 TO 8*PI STEP PI/8
20 PRINT TAB(20+18*SIN(T));"*"
30 NEXT
```

Mots-clés associés : ATN, COS, PI, TAN

15/08/2000 Page 104 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

SOUND

Code BASIC : 167
écriture : SOUND c, h, v
c = canal 0-6
h = hauteur 0-65535
v = volume 0-15

L'instruction **SOUND** produit un son défini au préalable grâce au micro-processeur spécialisé de l'ORIC. c définit le canal mis en activité parmi les canaux 1, 2 et 3 (canaux timbre) et 4, 5 et 6 (canaux bruit). On peut utiliser **SOUND** pour mettre en action des canaux sans instructions **PLAY** ; dans ce cas, c'est le canal timbre 1 qui est utilisé. h commande la hauteur du son produit ; il en détermine la fréquence. L'intervalle utile se situe entre 0 et 2000, mais le programme ci-dessous vous permettra d'en juger par vous-même. Le volume va de 0, qui met en action le paramètre d'enveloppe d'une instruction **PLAY** et produit un volume nul sans instruction **PLAY**, à 15 (fort !) en passant par 1 (faible).

```
10 FOR P=0 TO 32767
20 SOUND 1,P,2
30 NEXT
```

Les programmes ci-dessous donnent une idée de la maniabilité de l'instruction **SOUND**. La structure de base du programme est la même dans chaque cas, mais des sons très différents sont produits. Le premier programme comporte une instruction **PLAY** qui met en action le canal bruit 1, suivie par les instructions **SOUND** qui utilisent **WAIT** pour fixer la longueur. c est fixé à 4 (= canal bruit 1), la hauteur varie suivant la valeur de la variable boucle P, et deux volumes différents sont utilisés.

```
1 REM......SOUND.........
5 FOR P=1 TO 3
10 PLAY 0,1,1,250
20 SOUND 4,8+P,6
30 WAIT 5
40 SOUND 4,3+P,3
50 WAIT 20
60 PLAY 0,0,0,0
70 NEXTP
80 GOTO5
```


Le dernier paramètre de l'instruction **PLAY** n'entre pas en action tant que le volume de l'instruction **SOUND** est différent de 0. Remplacez la ligne 20 par **SOUND** 4,8 + P,0 et la ligne 40 par **SOUND** 4,3 + P,0 et vous verrez, ou plutôt entendrez, ce qui se passe quand la valeur d'enveloppe entre en action. (Vous pouvez en faire autant pour les deux exemples suivants). Dans le programme qui suit, seule la valeur du canal son est changée ; elle est fixée à 1, c'est-à-dire un canal timbre.

```
1 REM ....AUTRE EXEMPLE POUR SOUND
5 FOR P=1 TO 3
10 PLAY 0,1,1,250
20 SOUND 1,8+P,6
30 WAIT 5
40 SOUND 1,3+P,3
50 WAIT 20
60 PLAY 0,0,0,0
70 NEXT
80 GOTO 5
```

Dans le dernier exemple, on a simplement supprimé la ligne 10, et on utilise donc **SOUND** sans instruction **PLAY** ; cependant, **PLAY** 0,0,0,0 est utilisé pour arrêter le son.

```
1 REM ....3 EME EXEMPLE POUR SOUND
5 FOR P=1 TO 3
20 SOUND 1,8+P,6
30 WAIT 5
40 SOUND 1,3+P,3
50 WAIT 20
55 'effacez l.60 pour un autre son
60 PLAY 0,0,0,0
70 NEXT
```

Mots-clés associés : MUSIC, PLAY, WAIT

Numérisation http://wsc.n3.net
William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Numérisation http://wsc.n3.net

SPC

Code BASIC : 197
écriture : SPC(n)

SPC est un opérateur de l'instruction **PRINT** qui place n espaces sur l'écran. Si n n'est pas un nombre entier, la valeur est tronquée. On peut utiliser des expressions, et **SPC** est utile pour structurer un affichage ou une impression, puisqu'on peut mettre des chaînes de caractères entre les parenthèses. On utilise **SPC** avec l'instruction **PRINT** pour placer des espaces avant ou entre les différents éléments à afficher.

Le programme ci-dessus utilise une valeur de boucle pour définir le nombre d'espace à placer entre les astérisques ; de surcroît, il vous indique le nombre d'espaces qu'il a placé. L'instruction **SPC** remplace utilement l'instruction **TAB** ; elle est d'un usage très souple, comme le montre l'exemple suivant. La valeur produite à l'intérieur de la boucle **FOR...NEXT** est convertie en chaîne à la ligne 30 au moyen de **STR$**, et **LEN** est utilisé avec cette chaîne pour calculer un nombre donné d'espaces à la ligne 40. On obtient ainsi un affichage où tous les nombres sont alignés.

```
1 REM..CALCUL DE LA VALEUR DE SPC..
10 FOR F=1 TO 10
20 LET N%=F^2*39
30 LET N$=STR$(N%)
40 PRINT SPC(20-LEN(N$));N%
50 NEXT F
```

Mots-clés associés : PRINT, LPRINT, TAB

SQR

Code BASIC : 222
écriture : SQR(n)

SQR est une fonction qui calcule la racine carrée de la valeur représentée par l'expression numérique n ; celle-ci doit être positive (>0), sans quoi on a un message d'erreur **ILLEGAL QUANTITY ERROR**.

```
10 REM.......SQR.........
20 FOR N=30 TO 20 STEP -1
30 PRINTN,SQR(N)
40 NEXT N
```

Ce programme produit un affichage des nombres de 30 à 20 et de leurs racines carrées.

Mots-clés associés : Aucun

STOP

Code BASIC : 179
écriture : STOP

L'instruction **STOP** interrompt l'exécution d'un programme et provoque l'affichage du message **BREAK IN ln**, ln étant le numéro de la ligne où figure l'instruction **STOP**. Pour faire repartir le programme, on utilise la commande **CONT**, suivie de **RETURN**. **STOP** est similaire à l'instruction **END**, mais **END** met fin au programme, dont on ne peut alors recommencer l'exécution, comme le montre l'exemple ci-dessous.

```
10 REM......STOP.............
20 FOR X=1 TO 3
30 PRINT "X="X
40 PRINT"PROGRAMME ARRETE:TAPEZ 'CONT'
      POUR REPARTIR"
50 STOP
60 NEXT X
```

Mots-clés associés : END, WAIT

William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Numérisation http://wsc.n3.net

STORE

Code BASIC : 13Ø
écriture :
STORE v,"nom fichier" (,S)
STORE v$,"nom fichier" (,S)
STORE v%,"nom fichier" (,S)

On utilise **STORE** pour sauvegarder le contenu d'un tableau sur cassette magnétique. Le tableau doit auparavant avoir été dimensionné, soit au moyen de l'instruction **DIM**, soit en utilisant un élément du tableau pour déclarer implicitement un tableau à 11 éléments, sinon on aura une erreur par manque de données **(OUT OF DATA ERROR)**. Il peut s'agir d'un tableau de nombres en virgule flottante, par exemple A(2Ø), d'un tableau de nombres entiers par exemple A%(2Ø), ou d'un tableau de chaînes : A$(2Ø). La variable du tableau doit être spécifiée par A, A$ ou A% pour que l'identification du tableau soit correcte. Les tableaux à plusieurs dimensions sont admis. Le nom du fichier est laissé à votre libre choix, à condition de ne pas dépasser 16 caractères.

On utilise la même procédure que pour **CSAVE** ; de même qu'avec **CSAVE**, le débit implicite (rapide) de 24ØØ bauds peut être remplacé par le débit lent de 3ØØ bauds en ajoutant à la fin de l'instruction un S précédé d'une virgule (,S). Le message **SAVING FILENAME** ("filename" étant le nom du tableau ou fichier) apparaît sur la ligne d'état, suivi d'une lettre qui spécifie le type du tableau : R correspond aux tableaux contenant des nombres réels en virgule flottante, I aux tableaux de nombres entiers, et S aux tableaux de chaînes de caractères. Le programme suivant vous permettra de voir à l'œuvre les instructions **STORE** et **RECALL**. Le tableau A$(2Ø) est dimensionné et garni de chaînes qui servent à la démonstration. Puis il est sauvegardé sur cassette et les variables sont remises à zéro. Ensuite, le programme rappelle le tableau et affiche des valeurs échantillons.

```
 1 REM.....STORE RECALL.......
 10 DIMA$(20)
 20 FOR F=0 TO 230 LET A$(F)="NOMBRE "+
STR$(F)
 40 NEXT
 50 PRINT"LANCEZ L'ENREGISTREMENT,PUIS
APPUYEZ  SUR UNE TOUCHE"
 55 GETA$
 60 STORE A$,"TABLEAU",S
 70 CLEAR
 80 DIM A$(20)
```



```
90 PRINT"REMBOBINEZ LA CASSETTE,METTEZ
EN LEC- TURE,APPUYEZ SUR UNE TOUCHE"
95 GET A$
100 RECALL A$,"TABLEAU",S
110 PRINTA$(9)
120 PRINTA$(10)
130 END
```

Mots-clés associés : CLOAD, CSAV, DIM, RECALL

STR$

Code BASIC : 234
écriture : STR$(n)

Cette fonction de chaîne transforme une valeur numérique en chaîne alphanumérique. Cette instruction est donc l'inverse de **VAL**, qui donne la valeur numérique d'une chaîne. On peut utiliser des nombres exponentiels et hexadécimaux, qui seront convertis en notation standard (celle qui apparaîtrait normalement à l'écran) avant d'être transformés en chaîne. Le premier caractère de la chaîne comporte le signe si le nombre est négatif, mais reste blanc si le nombre est positif.

```
 1 REM...... STR$................
 10 LET N=12.34
 20 CLS:PRINT STR$(N):WAIT 15
 30 PLOT 10,1,STR$(N):WAIT 45
 40 PLOT 0,1,STR$(N):WAIT 15
 50 PLOT 10,8,STR$(N):WAIT 15
 60 LET X=-23.5
 70 PRINT
 80 PRINT STR$(X):WAIT 15
 90 PLOT 9,2,STR$(X)
 100 PRINT"L'HEXADECIMAL #A3 S'ECRIT"ST
R$(#A3)
 110 PRINT"LE NOMBRE 1.345E-4 S'ECRIT"S
TR$(1.345E-4)
 120 PRINT"LE NOMBRE 1.23E2  S'ECRIT"ST
R$(1.23E2)
```

Ce programme utilise **PLOT** pour positionner les caractères, mais on aurait tout aussi bien pu utiliser **PRINT** .

15/08/2000 Page 107 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

STR$ permet également de placer des nombres sur l'écran **HIRES**, alors qu'on ne peut normalement y placer que des caractères de chaîne au moyen de l'instruction **CHAR**. Le programme ci-dessous donne un exemple de cet usage : on obtient avec **STR$** la forme chaîne d'un nombre, puis on emploie l'instruction **ASC** pour trouver tour à tour le code de chaque caractère, qui est placé sur l'écran. On utilise la même technique pour éliminer le premier caractère si le nombre n'est pas négatif.

```
10 REM......STR$...........
20 HIRES
30 FOR A=1 TO 10 STEP .5
40 A$=STR$(A)
50 GOSUB 100
60 NEXT
70 END
100 'AFFICHAGE D'UNE CHAINE EN HIRES
110 FOR B=1 TO LEN (A$)
120 CURSET B*6+A*6,A*16,0
130 CHAR ASC(MID$(A$,B)),0,1
140 NEXT
150 RETURN
```

Mots-clés associés : VAL

TAB

Code BASIC : 194
écriture : PRINT TAB(n)

On utilise **TAB** dans une instruction **PRINT** pour placer à une colonne donnée les éléments à afficher. La valeur de n définit la colonne vers laquelle la position d'affichage se déplace. Les éléments suivants seront affichés immédiatement après si rien n'est placé après l'instruction **TAB(n)** ou si elle est suivie d'un point-virgule ; si on utilise une virgule, ils s'inscriront dans le champ d'affichage suivant. Les nombres sont affichés précédés d'un espace, sauf s'ils sont négatifs. Les colonnes sont numérotées de Ø à 39 de gauche à droite de l'écran ; le programme suivant montre comment agissent les colonnes protégées.

```
10 FOR F=0 TO 15
30 PRINT TAB(F);F
40 NEXT F
50 ' tapez CTRL ] et essayez encore
```


Une instruction **PRINT** peut comporter plusieurs **TAB**, comme on le voit dans l'exemple ci-dessous.

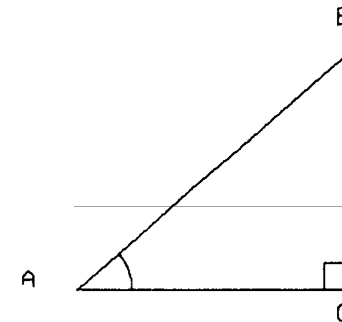
```
10 PRINT TAB(10)10;TAB(20);20
20 PRINT TAB(10);10;TAB(20);-20
30 PRINT TAB(10)"X"TAB(20),"Y"
```

Mots-clés associés : LPRINT, PRINT, POS, SPC

TAN

Code BASIC : 228
écriture : TAN(n)

TAN est une fonction trigonométrique qui donne la tangente de l'angle évalué par l'expression numérique n. Le résultat donné par **TAN** est équivalent à **SIN(n)/COS(n)**. Cela correspond dans la figure ci-dessous au rapport entre côté opposé et côté adjacent du triangle rectangle. La valeur de n doit être exprimée en radians. Reportez-vous à **SIN** pour la conversion des degrés en radians et vice versa.



Le premier exemple ci-dessous affiche simplement une table des valeurs de **TAN** pour des mesures d'angle allant de Ø à 36Ø degrés (2*PI radians). Le second programme calcule la distance de B et C (ce pourrait être par exemple la largeur d'une rivière) connaissant la distance AC (mesurée le long de la berge) et l'angle formé en A.

```
10 REM...... TAN.........
20 DEF FNR(DEG)=DEG*PI/180
30 FOR D=0 TO 360
40 P=D/20:P%=D/20
50 PRINT D,TAN(FNR(D))
60 IF P=P% THEN WAIT 100
70 NEXT
```


William Saint-Cricq / wsaintcr@wanadoo.fr Page 108 sur 163 15/08/2000

```
 10 REM ......TAN...............
 20 CLS:PRINT"ANGLE EN A ";
 30 READ A:PRINT A
 40 PRINT"LONGUEUR AC";:READ AC:PRINT A
C
 60 PRINT:PRINT" PUISQUE TG(A) EST DEFI
NIE PAR BC/AC"
 70 PRINT"BC= AC*TG(A),DONC BC="TAN(A)*
AC
 100 END
 110 ' l'angle est en radians...
 120 DATA 0.73,20
```

Mots-clés associés : ATN, COS, SIN, PI

TEXT

Code BASIC : 161
écriture : TEXT

L'instruction **TEXT** met l'ORIC en mode texte standard ; ce mode est celui que l'on obtient au moment de la mise en marche, et il donne un écran de 40 colonnes sur 27 lignes, avec affichage du jeu de caractères standard (et des caractères définis par l'utilisateur). Les instructions **LORES** et **HIRES** définissent d'autres modes écran qui subsistent tant qu'on n'en sort pas en utilisant respectivement **CLS** et **TEXT**.

L'écran **LORES** défile vers le haut en laissant un écran **TEXT** ; il faut cependant répéter plusieurs fois l'instruction **PRINT**.

Mots-clés associés : HIRES, LORES

TROFF

Code BASIC : 133
écriture : TROFF

TROFF, abréviation des mots anglais **TRACE OFF**, désactive la fonction d'affichage du parcours (ou "trace") qui affiche à l'écran les numéros des lignes d'un programme au fur et à mesure qu'elles sont exécutées par l'interpréteur BASIC. Voir **TRON**, qui met en activité cette fonction.

Mots-clés associés : TRON

TRON

Code BASIC : 132
écriture : TRON

TRON (c'est-à-dire **TRACE ON**) permet de repérer les erreurs ou "bugs" dans les programmes en affichant entre crochets [] le numéro de chaque ligne du programme chaque fois que l'interpréteur BASIC la rencontre et l'exécute. L'insertion temporaire de **TRON** et **TROFF** dans un programme permet au programmeur de visualiser l'ordre dans lequel les lignes sont exécutées, ainsi que les résultats normalement affichés à l'écran de l'exécution de ces lignes. Le programme ci-dessous présente un problème de boucle sans fin. En affichant les numéros de ligne, l'instruction **TRON** nous permet de voir leur ordre d'exécution.

```
10 REM.....TRON......
20 CLS
30 TRON
40 FOR A=1 TO 10
50 PRINT A
60 IF A=6 THEN A=1
70 NEXT
80 END
100 END
110 ' l'angle est en radians...
120 DATA 0.73,20
```

En insérant un **TROFF** en ligne 65, on obtiendrait l'affichage des numéros de ligne pour le premier cycle de déroulement de la boucle **FOR...NEXT**.

Mots-clés associés : TROFF

TRUE

Code BASIC : 239
écriture : TRUE

TRUE est une constante système intrinsèque à l'ORIC qui restitue la valeur – 1 utilisée pour représenter le résultat de l'évaluation d'une expression conditionnelle comme vraie ("true" en anglais).

Numérisation http://wsc.n3.net
15/08/2000 Page 109 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

La valeur correspondant à "faux" ("false") est 0, et c'est la valeur détenue par la variable **FALSE.** L'association de ces deux variables permet de rendre un programme plus clair, en y adjoignant le plus souvent des indicateurs ou "drapeaux", auxquels on affecte habituellement une de ces deux valeurs et qui représentent des conditions faciles à tester. Il faut utiliser **TRUE** avec prudence si l'opérateur **NOT** doit être utilisé : en effet, l'ORIC considère comme vraie toute valeur d'une expression numérique distincte de zéro, mais alors que **NOT TRUE = FALSE** et **NOT FALSE = TRUE,** le test conditionnel **IF AB THEN...** sera évalué comme vrai **(TRUE)** si la variable AB est différente de zéro ; par contre, si AB est égal à, mettons, 34, **NOT AB** donnera −23. Pour l'ORIC, cette valeur n'a rien à voir avec **FALSE !**

Le programme ci-dessous donne un exemple d'utilisation de **TRUE** et **FALSE** pour vérifier une boucle **REPEAT...UNTIL.**

```
 10 REM.......TRUE...........
 20 FLAG=FALSE
 30 PRINT"TAPEZ UN MOT DE DEUX LETTRES"
 40 REPEAT
 50 UNTIL FLAG=TRUE
 60 PRINT"VOUS POURRIEZ LIRE LES INSTRU
CTIONS ..."
 70 NEXT
 80 END
 100 END
 110 ' l'angle est en radians...
 120 DATA 0.73,20
```

Mots-clés associés : FALSE

UNTIL

Code BASIC : 140
écriture : UNTILc

UNTIL fait partie de la structure de boucle **REPEAT...UNTIL.** Quand le programme arrive à l'expression **UNTIL**, l'expression conditionnelle fait l'objet d'une évaluation. Si elle est évaluée comme vraie **(TRUE)**, le programme sort de la boucle et exécute l'instruction suivante. Si la condition est fausse **(FALSE)**, l'exécution reprend à l'instruction suivant le **REPEAT** qui lance la boucle.

Si aucun **REPEAT** correspondant ne peut être trouvé, on obtient un message d'erreur :
?BAD UNTIL ERROR.

Mots-clés associés : FALSE, REPEAT, TRUE

USR

Code BASIC : 217
écriture : DEF USR = adr
USR(i)

Cette fonction permet d'avoir accès, dans le cours d'un programme en BASIC, aux programmes en code machine. La première présentation indiquée ci-dessus définit par adr l'adresse de départ du programme en code machine.

Dans la deuxième présentation, le programme est appelé au moyen d'**USR(i)** et place la valeur de i dans l'accumulateur en virgule flottante. Une fois que le programme en code machine est achevé, **USR** renvoie au programme principal ; le résultat doit alors être soit affiché **(PRINT USR(0))** soit affecté à une variable **(A = USR(0)).** S'il n'y a pas de valeur à transmettre à un programme en code machine, il faut utiliser **CALL** pour appeler ce programme. Reportez-vous au chapitre 10 où vous trouverez une initiation à la programmation en code machine.

Mots-clés associés : CALL, DEF

VAL

Code BASIC : 235
écriture : VAL(a$)

Cette fonction de chaîne restitue la valeur numérique des caractères de la chaîne donnée entre parenthèses. Le premier caractère de la chaîne à évaluer doit être un blanc, un signe moins, un # ou un nombre, faute de quoi on obtient zéro. A la suite de ces caractères, ou du premier nombre, la chaîne est évaluée jusqu'au premier

William Saint-Cricq / wsaintcr@wanadoo.fr Page 110 sur 163 15/08/2000

Numérisation http://wsc.n3.net

caractère non-numérique, en équivalent décimal si un nombre hexadécimal (commençant par #) se trouve dans la chaîne. La notation exponentielle est également traitée ; c'est la forme sous laquelle le nombre est affiché à l'écran qui est retenue, et non la forme sous laquelle il était inscrit dans la chaîne. A titre d'exemple, essayez d'entrer 1.23E2 dans le programme ci-dessous, parmi d'autres formulations numériques. Une fois que l'ORIC a trouvé dans la chaîne des caractères qu'il peut interpréter comme un nombre, il ne tiendra pas compte des caractères non-numériques qu'il trouvera par la suite.

```
 10 REM.......VAL...............
 20 CLS
 30 REPEAT
 40 INPUT"UNE CHAINE QUELCONQUE,SVP";A$
 50 PRINT"ELLE COMMENCE PAR LE NOMBRE"V
AL(A$)
 70 PRINT
 80 UNTIL A$="STOP"
 90 END
```

Mots-clés associés : ASC, STR$

WAIT

Code BASIC : 181
écriture : WAIT n

En utilisant cette instruction, on obtient un temps d'attente de n centièmes de seconde avant que l'exécution du programme se poursuive. Il est indispensable d'utiliser **WAIT** avec les instructions sonores de l'ORIC, pour contrôler la durée des résultats des instructions **PLAY**, **SOUND**, et **MUSIC**. On peut aussi l'utiliser pour introduire des pauses dans les programmes ou pour ralentir les affichages à l'écran, mais notez qu'une entrée faite au clavier ne peut interrompre la période d'attente. Pour obtenir un temps d'attente avant que l'utilisateur appuie sur une touche ou rentre des données, il faut utiliser **GET** et **INPUT**.

Mots-clés associés : MUSIC, SOUND, PLAY

ZAP

Code BASIC : 165
écriture : ZAP

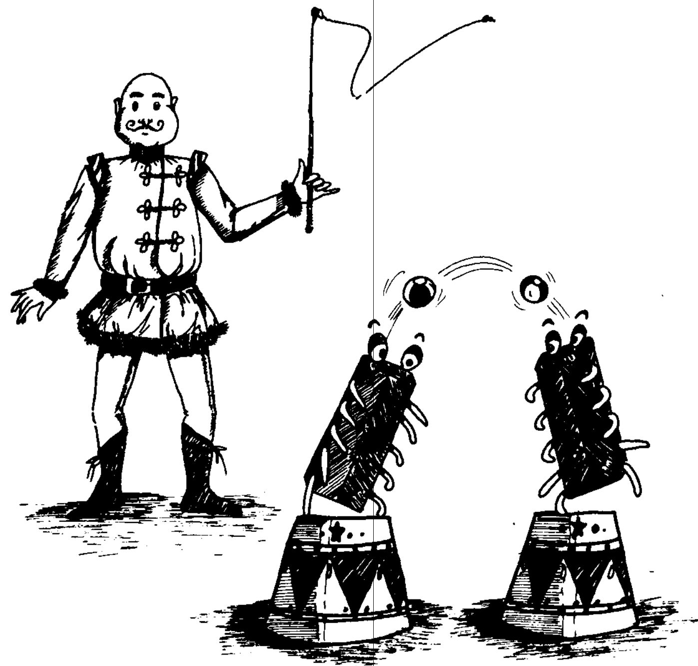
ZAP fait partie des sons préprogrammés de l'ORIC ; on l'utilise dans les jeux pour obtenir un bruit évoquant la détonation d'une arme de science-fiction. A la différence des autres membres de sa famille, **ZAP** peut s'utiliser sans temps d'attente ; on peut produire le son **ZAP** de façon répétée sans faire appel à l'instruction **WAIT** nécessaire avec **PING, SHOOT** et **EXPLODE.**

```
10 REM......ZAP...........
20 FOR F=1 TO 6
30 ZAP
40 NEXT
```

Mots-clés associés : EXPLODE, SHOOT, PING

William Saint-Cricq / wsaintcr@wanadoo.fr Page 111 sur 163 15/08/2000

Numérisation http://wsc.n3.net



Chapitre 10
Initiation au code machine

Vous vous êtes déjà essayé à la rédaction de programmes en BASIC Oric, et vous vous demandez peut-être si vous ne pourriez pas accélérer un peu le rythme, pour que vos Envahisseurs se déplacent sur l'écran plus vite et plus régulièrement, et pour que vos programmes, d'un manière général, s'exécutent plus rapidement. Oui, c'est possible, mais voyons d'abord pourquoi l'animation paraît parfois lente ou saccadée, et pourquoi des temps d'attente évidents interrompent parfois l'exécution de programmes complexes.

Le BASIC est un langage *interprété* (bien que parfois, il puisse aussi être *compilé*, si l'on dispose du programme et du système d'exploitation adéquats) : cela veut dire que les instructions que vous codez (le programme) sont mémorisées au fur et à mesure que vous les tapez. Au moment de l'exécution, l'interpréteur BASIC doit donc examiner chaque instruction, "comprendre" quel est le but de l'instruction, et appeler le programme en code machine qui convient pour mettre en œuvre l'instruction. Bien que les programmes en code machine soient en eux-mêmes rapides et efficaces, il est parfois nécessaire de faire appel à plusieurs de ces programmes pour mettre en œuvre une seule instruction. Et surtout, chaque instruction doit être *analysée* : l'interpréteur y recherche les opérateurs tels que "+", "—", **"IF"**, etc., et réorganise l'instruction pour lui donner une forme adaptée à l'exécution linéaire dans l'ordre spécifique exigé par l'Oric (conforme à l'ordre des priorités) ; puis il la transmet aux programmes en code machine. Ce processus d'interprétation se produit pour chaque instruction, chaque fois que l'instruction se présente, et c'est ce qui explique la lenteur des opérations. Si nous programmons directement en code machine et que nous mettons au point nos propres programmes en vue de chaque fonction spécifique, nous obtenons un code incomparablement plus rapide, et il se peut même que nous devions introduire nous mêmes des temps d'attente pour ralentir les opérations ! Vous avez déjà essayé d'abattre un Envahisseur qui met un dixième de seconde pour traverser l'écran ?

Cependant, la programmation en code machine présente aussi quelques désavantages. C'est un langage difficile à apprendre et les erreurs y sont plus fréquentes et plus difficiles à éliminer : on n'y est pas aidé par des messages utiles du genre **"SYNTAX ERROR"** ! Mais l'utilisation du code machine est intéressante et gratifiante ; loin de n'être à la portée que des programmeurs professionnels, le code machine est tout à fait accessible à l'amateur attentif et sérieux.

William Saint-Cricq / wsaintcr@wanadoo.fr Page 112 sur 163 15/08/2000

Numérisation http://wsc.n3.net

Pour utiliser le code machine de façon efficace et correcte, il faut d'abord acquérir quelques bases concernant le microprocesseur 65Ø2 ; mais avant d'aborder ce sujet, nous devons en savoir un peu plus sur les systèmes numériques, c'est-à-dire sur les façons de représenter et de traiter les données numériques. Il faut aussi savoir comment sont représentés les caractères (ceux de l'alphabet, par exemple). Nous allons donc aborder ces questions avant de passer aux codes d'instructions proprement dits. Ne sautez pas les paragraphes suivants, ne les lisez pas en diagonale : leur lecture peut vous épargner de longues heures passées à rechercher et à éliminer les erreurs !

Les systèmes de numération : le système décimal

La Nature nous a donné dix doigts sur lesquels compter, et c'est pourquoi notre arithmétique se fonde principalement sur un système décimal. Les Arabes nous ont donné le zéro et la virgule, et cela nous a permis de mettre au point des méthodes de multiplication et de division qui nous facilitent bien la vie. Voyons d'abord ce qu'il en est du système décimal.

Quand nous comptons, nous commençons généralement à un, bien que dans un compte à rebours on aille à reculons jusqu'à zéro. 5... 4... 3... 2... 1... ZÉRO — PARTEZ ! Le zéro est important : en fait, on devrait commencer par là lorsqu'on compte en allant vers les nombres supérieurs. Lorsqu'on atteint le nombre neuf, on revient à zéro, mais en lui ajoutant un chiffre de tête : "un" lors de la première série, "deux" lors de la deuxième, etc. On a : Ø, 1, 2, 3, 4 ...9, 1Ø. Quand on atteint 99, on ajoute un deuxième chiffre de tête et on revient de nouveau à zéro : 1ØØ.

En fait, on ajoute un au nombre précédent jusqu'à ce que le dernier chiffre soit 9. Puis on remet à zéro le dernier chiffre et on retient un, que l'on ajoute à l'avant-dernier chiffre. Si ce chiffre est déjà 9, on le remet à zéro et on retient un que l'on rajoute au chiffre précédent, et ainsi de suite. En fin de compte, additionner à un autre nombre un nombre supérieur à un revient à lui ajouter un à plusieurs reprises. Par exemple, on peut écrire 9+5 sous la forme 9+1+1+1+1+1. Si nous ne le faisons pas d'habitude, c'est que nous avons mémorisé les sommes de toutes les combinaisons possibles des chiffres de Ø à 9 et que l'addition est devenue un procédé qui s'effectue de façon automatique.

Pourquoi prenons-nous la peine de revenir à des évidences ? C'est que la patience est une vertu et que les automates n'ont pas de capacité créatrice ; il faut donc que nous nous libérions de nos présupposés et de nos automatismes. Le système de numération des ordinateurs n'est pas le système décimal. En informatique, 1K représente 1Ø24 unités, et non 1ØØØ. Pourquoi, alors qu'un kilogramme (ou kg) représente 1ØØØ grammes ? Vous allez tout comprendre, car nous allons passer au système *binaire*.

Le système binaire

Le système binaire n'utilise que deux chiffres : zéro et un (Ø et 1). En informatique, on emploie toujours le zéro barré : Ø, pour le différencier de la lettre O, et nous vous conseillons de prendre cette habitude. Dans le système décimal, nous l'avons vu, on retient un chaque fois qu'on ajoute un à neuf, et ce neuf est alors remis à zéro. Dans le système binaire, on retient un dès qu'on ajoute un à un, et le chiffre le plus à droite revient à zéro. On a donc 1+1=1Ø. C'est la première règle de ce système. La deuxième est encore plus facile : Ø+1 (ou 1+Ø)=1. La troisième est évidente : Ø+Ø=Ø.

Exercez-vous avec l'opération suivante :
1Ø1Ø1+Ø11Ø1=?

Si vous n'avez pas trouvé la réponse : 1ØØØ1Ø, relisez attentivement le paragraphe précédent.

Comparons maintenant les chiffres décimaux avec leurs équivalents binaires.

Décimal	Binaire	Hexadécimal
Ø	ØØØØ	Ø
1	ØØØ1	1
2	ØØ1Ø	2
3	ØØ11	3
4	Ø1ØØ	4
5	Ø1Ø1	5
6	Ø11Ø	6
7	Ø111	7
8	1ØØØ	8
9	1ØØ1	9
1Ø	1Ø1Ø	A
11	1Ø11	B
12	11ØØ	C
13	11Ø1	D
14	111Ø	E
15	1111	F

Comme vous êtes un lecteur astucieux, vous avez tout de suite remarqué que les nombres binaires se présentent comme une séquence de quatre chiffres binaires et que le tableau comporte une troisième colonne intitulée "Hexadécimaux". Peut-être avez-vous remarqué que cette troisième colonne était identique à celle des nombres décimaux jusqu'à la valeur décimale 1Ø, après quoi elle devenait alphabétique. Il est en effet assez lassant d'écrire de longues chaînes de "un" et "zéro", et des erreurs peuvent en découler, aussi représente-t-on par un seul chiffre chaque groupe de quatre chiffres binaires. Puisqu'il n'existe pas de nombres à un seul chiffres supérieur à 9, on emprunte à l'alphabet ses six premiers caractères : A, B, C, D, E et F, pour représenter les nombres 1Ø à 15.

15/08/2000
Page 113 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

Cela rend la vie bien plus facile : par exemple, le nombre décimal 1Ø24 a pour équivalent binaire Ø1ØØØØØØØØØØ, et pour équivalent hexadécimal 4ØØ. On obtient cette valeur en groupant les chiffres binaires quatre par quatre à partir de la droite et en convertissant chaque groupe en bon équivalent hexadécimal. On a donc :

```
Ø1ØØ  ØØØØ  ØØØØ  binaire
  4     Ø     Ø   hexa
```

C'est très ingénieux, direz-vous peut-être, mais quel rapport cela a-t-il avec la programmation en code machine ? Patientez encore un peu, et vous verrez que le rapport est étroit. Retenez d'abord deux termes que vous devez connaître. Par convention, on appelle *bits* les chiffres binaires (*bi*nary dig*its*). On peut donc dire qu'un chiffre hexadécimal représente quatre bits. Ensuite, l'unité de base de la mémoire de l'ordinateur est l'*octet*, constitué de 8 bits. On peut donc exprimer la valeur d'un octet sous forme de deux chiffres hexadécimaux. Par exemple :

17 (en décimal) = ØØØ1ØØØ1 (en binaire)
= 11 (en hexadécimal)

La valeur binaire la plus élevée que peut prendre un octet s'exprime ainsi :

1111 1111
= F F en hexadécimal
= 255 en décimal

Un octet peut donc prendre 256 valeurs : de Ø à 255 en décimal. Quand on veut se référer aux différents bits d'un octet, on les numérote de Ø à 7 et de droite à gauche.

Si l'on n'a pas une connaissance approfondie des systèmes numériques, la conversion des nombres binaires ou hexadécimaux en décimal est assez fastidieuse pour les grands nombres. Le principe de base en est le suivant : le système décimal est en base 1Ø, et la valeur de chaque chiffre en allant de droite à gauche est égale au produit de ce chiffre par 1Ø élevé à une puissance croissante. La puissance est le nombre de fois qu'un nombre est multiplié par lui-même ; toutes les puissances zéro sont égales à 1, sauf zéro puissance zéro qui est égal à zéro.

Le nombre décimal 123, par exemple, peut s'écrire des façons suivantes :

$$(1 \times 1Ø^2) + (2 \times 1Ø^1) + (3 \times 1Ø^Ø)$$

ou

$$(1 \times 1Ø \times 1Ø) + (2 \times 1Ø) + (3 \times 1)$$

ou

$$(1 \times 1ØØ) + (2 \times 1Ø) + (3 \times 1)$$

ou

$$1ØØ + 2Ø + 3 = 123$$

Les nombres écrits en petit au-dessus des 1Ø indiquent les puissances.

En binaire, on applique le même principe, mais en base 2. On peut donc écrire le nombre décimal 15 en binaire des façons suivantes :

$$\begin{aligned} 1111 &= (1 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^Ø) \\ &= (1 \times 2 \times 2 \times 2) + (1 \times 2 \times 2) + (1 \times 2) + (1 \times 1) \\ &= (1 \times 8) + (1 \times 4) + (1 \times 2) + (1 \times 1) \\ &= 8 + 4 + 2 + 1 \\ &= 15 \text{ en décimal} \end{aligned}$$

L'hexadécimal utilise la base 16 : il y a seize chiffres dans ce système, de Ø à 15. Le nombre hexadécimal 21 peut donc s'écrire des façons suivantes :

$$\begin{aligned} 21(\text{hexa}) &= (2 \times 16^1) + (1 \times 16^Ø) \\ &= (2 \times 16) + (1 \times 1) \\ &= 32 + 1 \\ &= 33 \text{ en décimal} \end{aligned}$$

Les exemples ci-dessus illustrent la conversion en décimal du binaire et de l'hexadécimal. Pour convertir en binaire ou en hexadécimal un nombre décimal, on le divise à plusieurs reprises par la base appropriée, puis on enchaîne les restes de ces opérations en commençant par celui de la dernière opération et en allant à rebours :

```
15(décimal)15/2 = 7, reste 1
            7/2 = 3, reste 1
            3/2 = 1, reste 1       1111 (binaire)
            1/2 = Ø, reste 1
11 (décimal)11/2 = 5, reste 1
             5/2 = 2, reste 1
             2/2 = 1, reste Ø      1Ø11 (binaire)
             1/2 = Ø, reste 1
```

De même, pour convertir un nombre décimal en hexadécimal, on divise par 16 (décimal) :

```
43 (décimal)  = 43/16 = 2, reste 11 = B(hexa)
                 2/16 = Ø, reste 2 = 2(hexa)      = 2B
1Ø7 (décimal) = 1Ø7/16 = 6, reste 11 = B(hexa)
                  6/16 = Ø, reste 6 = 6(hexa)     = 6B
```

Les nombres négatifs

Jusqu'à présent, nous avons envisagé uniquement des opérations arithmétiques débouchant sur un résultat positif. Nous devons envisager également des résultats négatifs. Comment sont-ils représentés et identifiés ?

15/08/2000
Page 114 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

Dans le système binaire, on indique par conversion qu'un nombre est négatif en fixant à 1 le bit de poids fort (celui qui est le plus à gauche). Mais cela n'est pas suffisant : il faut avoir recours à une autre convention pour qu'on puisse additionner des nombres positifs et négatifs et obtenir un résultat correct.

Additionnons par exemple −4 et +5 en utilisant le bit de poids fort du premier nombre pour indiquer qu'il est de signe négatif :

−4 = 10000100 (binaire)
+5 = 00000101 (binaire)

L'addition binaire donne 10001001

Ce résultat est visiblement incorrect. Pour obtenir un résultat correct, on doit soumettre la valeur absolue des nombres négatifs à une opération appelée *complémentation à deux*, qui consiste à permuter la valeur de chaque bit, c'est-à-dire à changer les 0 en 1 et vice versa, et à ajouter 1. Essayons avec 4 :

4(décimal) =	00000100
inversion des 0 et des 1 :	11111011
addition de 1 :	11111100
addition de +5 :	00000101
résultat :	00000001

Le résultat comporte neuf bits : 100000001. Le bit le plus à gauche "tombe" de l'extrémité : c'est une retenue dont on n'a pas à tenir compte. Les bits restants donnent comme résultat +1 ; c'est bien la somme de +5 et −4. Pour interpréter un nombre négatif, on soustrait 1 et on permute les bits.

Par exemple :

```
  11111111
 -       1

  11111110
```

Permutation = 00000001 = 1 donc 11111111 = −1 (décimal)
Prenons maintenant comme exemple 127 + 1 :

127 = 01111111 — le bit de signe est 0 (positif)
+1 = 00000001

10000000 — le bit de signe est 1 (négatif)

Ce "dépassement" du bit 6 sur le bit 7 qui semble changer le signe d'un résultat est un cas d'erreur qui doit être testé et auquel on doit remédier en cours de calcul. Dans la pratique, ce n'est que le bit de poids fort du nombre qui doit être soumis à un test d'erreur ; dans un nombre de 16 bits, il s'agit du bit 15. Il est donc indispensable de déterminer la taille du plus grand nombre à traiter quand on décide du nombre d'octets à affecter au stockage des nombres en mémoire. On n'a pas à tenir compte du dépassement sur les octets de poids faible dans un nombre de plusieurs octets, seuls les reports doivent être testés.

Le décimal codé binaire ou DCB

Jusqu'à présent, nous avons travaillé en binaire. On peut aussi représenter les nombres décimaux sous forme binaire et effectuer des opérations arithmétiques avec les nombres ainsi codés.

Puisque le chiffre décimal le plus grand est 9 et qu'on peut le représenter au moyen de 4 bits — 1001 — on peut "ranger" dans un octet deux chiffres décimaux. Par exemple 99 = 10011001. Ce procédé de représentation des nombres décimaux porte le nom de *décimal codé binaire*, ou DCB en abrégé.

Quand on effectue des opérations arithmétiques sur des nombres en DCB, il faut vérifier la présence d'un "dépassement" à 9. C'est-à-dire d'une génération de combinateurs binaires supérieurs à 9, et modifier en conséquence le chiffre concerné, en reportant 1 sur le chiffre suivant.

On peut calculer les nombres en complément à dix en soustrayant la valeur positive d'un nombre d'une chaîne de 9 correspondant à la valeur maximale des nombres à représenter et en ajoutant 1. Par exemple, si le nombre le plus grand sur lequel on compte effectuer des calculs est 999, le complément à dix de 2 (c'est-à-dire −2) s'obtient comme suit :

```
  999
−   2
 ----
  997
+   1
 ----
  998
```

La somme de 763 − 2, par exemple, sera donc :

```
                                    763
1761 = 761 + 1 (retenue négligeable) +998
```

Heureusement, il n'est pas nécessaire d'avoir recours à ce procédé lorsqu'on écrit en code machine 6502, puisque cette puce-là a des aptitudes à l'arithmétique décimale ! Mais les notions exposées ci-dessus permettent d'avoir une vision plus complète de la question. Nous espérons donc avoir débroussaillé le domaine complexe de la numération informatique, et nous pouvons maintenant passer à l'adressage.

15/08/2000 Page 115 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

L'adressage

Quand on programme en BASIC, on associe un numéro de ligne à chaque instruction ou groupe d'instructions. Si le programme comporte des **GOTO** et **GOSUB**, on peut considérer ces numéros de ligne comme des *adresses* auxquelles ces instructions renvoient en modifiant la gestion normale de l'ordre d'exécution du programme. Si notre programme suivait un ordre purement séquentiel, on n'aurait besoin ni de **GOTO** ni de **GOSUB**, et on n'aurait pas à numéroter les lignes. Dans certains langages évolués autres que le BASIC, les numéros de ligne servent uniquement d'indicateurs en cas de branchement.

Quand on utilise le code machine, l'adressage ne se fait pas au moyen de numéros d'instruction, et on a besoin d'un pointeur qui renvoie les équivalents en code machine de **GOTO** et de **GOSUB** aux parties appropriées du programme. De plus, on doit décider si les variables et les constantes doivent être mémorisées, alors que cela se fait automatiquement quand on programme en BASIC. Quand nous étudierons les instructions en code machine, nous constaterons qu'il est souvent nécessaire de leur donner une adresse (renvoyant à des données ou à l'instruction suivante) ; cette adresse doit être fournie sous une forme exécutable par la machine, c'est-à-dire en binaire, puisque c'est le seul langage que comprenne l'ordinateur. (Pour plus de commodité, on représentera de préférence le binaire par de l'hexadécimal).

La puce 65Ø2 comporte seize lignes d'adressage, c'est-à-dire qu'une adresse-mémoire compte seize bits (bien qu'il existe un cas particulier où les adresses font huits bits). L'adresse mémoire la plus élevée est donc l'adresse 1111 1111 1111 1111, ou FFFF (hexa), ou 65535 (décimal), que l'on note en général 64K. Si l'on n'utilisait pas la notation hexadécimale (en quatre chiffres hexa) il faudrait inclure des séries de seize bits dans toutes les instructions comportant une adresse mémoire ! Le temps de codage et d'entrée du moindre programme serait alors excessivement long, et de nombreuses erreurs risqueraient de s'y glisser. Voilà pourquoi, cher programmeur, si vous avez effleuré d'un œil rapide les paragraphes ci-dessus, nous vous conseillons de revenir au début et des les lire attentivement. Faites donc aussi quelques exercices avec des exemples de votre choix, et vérifiez vos réponses en utilisant les tableaux donnés en annexe à ce manuel. Ce ne sera pas du temps perdu, car vous deviendrez bientôt un virtuose de la conversion d'un système de numérotation à un autre. Même si vous ne tenez pas à pénétrer toutes les subtilités du code machine, la connaissance des systèmes binaire et hexadécimal constitue un bon passeport pour l'informatique.

Les instructions en code machine

Ces instructions comportent essentiellement deux éléments : le code opération et l'opérande.

Le code opération, dans le cas du 65Ø2 et de la plupart des microprocesseurs, tient en un octet. La valeur de cet octet en binaire indique l'opération à effectuer (addition ou soustraction, par exemple). Par commodité, chaque code opération porte son nom, ou *mnémonique*, qu'on est censé se rappeler plus facilement que le code hexa correspondant. Vous trouverez à l'Annexe 8 une liste des codes opérations et de leurs mnémoniques.

La deuxième partie de l'instruction ou opérande varie suivant la nature de l'instruction ; elle peut comporter un ou deux octets ou figurer implicitement dans le code opération, sans être formulée en tant qu'opérande. Dans ce dernier cas, le code opération définit a lui tout seul l'ensemble de l'opération. Quand les opérandes sont matérialisées, elles contiennent soit une adresse (un ou deux octets) soit une donnée (un octet).

Code op. Donnée (1 octet)
ou Adresse (2 octets)
Adresse (2 octets)

Les codes opération en 1 octet donnent 256 possibilités d'instructions (valeurs allant de Ø à 255). En fait, pour programmer de façon efficace, on a besoin de beaucoup moins d'instructions, et le 65Ø2 utilise les valeurs qui restent disponibles pour modifier le jeu d'instruction de base de façon à y inclure différents modes d'adressage. A chaque type d'instruction peuvent donc correspondre différents codes opération dont chacun utilise un mode d'adressage distinct. Mais avant d'étudier ces modes nous devons nous familiariser avec les notions de *registre*, de *page*, d'*indexation* et d'*adressage indirect*.

Il convient ici de mentionner brièvement un principe des opérations en code machine. Le fonctionnement du 65Ø2 est rythmé par une horloge qui synchronise les cycles opératoires de l'unité centrale de traitement (UCT). Chaque cycle correspond au déroulement d'une phase de l'opération exécutée. Le premier cycle appelle le code opération, et les cycles suivants recherchent le deuxième et le troisième octet, correspondant à l'opérande. Le cycle suivant effectue l'opération, sauf si le mode d'adressage utilisé est l'adressage indirect, auquel cas les cycles sont plus nombreux. Le nombre de cycles varie donc suivant les opérations.

William Saint-Cricq / wsaintcr@wanadoo.fr Page 116 sur 163 15/08/2000

Les registres

Un registre est un emplacement mémoire du microprocesseur 65Ø2 dans lequel on range les données nécessaires à l'exécution d'une instruction. Comme les registres sont inclus dans la puce elle-même, ils ne sont directement adressables que par l'intermédiaire des instructions qui agissent sur eux. Le 65Ø2 comporte six registres, dont un de seize bits : le compteur d'instruction ou PC (P comme programme). Les autres (registres A,X,Y,S et P) étant des registres de huit bits. Les registres peuvent également servir de mémoires temporaires (où les données sont stockées entre deux programmes) ou de compteurs. Cependant, chacun des registres énumérés ci-dessus a une fonction spécifique qui sera exposée un peu plus loin. Voyons d'abord les autres notions.

Les pages

Comme nous l'avons dit plus haut, le 65Ø2 a seize lignes d'adresse qui lui donnent un accès direct à 2^{16}, soit 65536, emplacements en mémoire centrale. L'adressage se fait au moyen d'adresses de deux octets, dont la valeur est comprise entre ØØØØ et FFFF hexa. Mais les instructions d'un octet, d'une exécution plus rapide, sont également admises, et cinq des six registres sont longs de huit bits (un octet). De ce fait, certaines instructions ne donnent accès qu'à 256 octets de mémoire, en tête de la mémoire vive (on RAM). Il existe un mode d'adressage spécial qui permet aux instructions d'adressage courtes d'accéder beaucoup plus rapidement aux 256 premiers octets de mémoire. On appelle ce mode *adressage en page zéro,* ce qui nous amène à la notion de pages de mémoire.

En raison de l'architecture interne du 65Ø2, on peut considérer que les 64K de RAM sont constitués de pages de 256 octets chacune, la page zéro et la page un ayant des fonctions spéciales. Précisons que les instructions qui entraînent le dépassement d'une limite de page prennent un cycle de plus à exécuter.

Nous parlerons maintenant des limitations de longueur des registres, ce qui nous permettra d'aborder la notion d'indexation.

L'indexation

L'indexation permet de modifier les adresses de façon dynamique : l'adresse de l'instruction est modifiée par ce moyen au moment de l'exécution. Le recours à l'indexation est nécessaire pour accéder à des articles consécutifs d'un tableau ou d'une matrice auxquels on ne pourrait accéder autrement qu'un par un, en utilisant une instruction pour chaque article du tableau. L'utilisation des indices dans les fichiers en BASIC est un équivalent de cette méthode, dans un langage évolué.

On a mentionné ci-dessus les registres X et Y. Il serait plus exact de les appeler registres d'index X et Y, car ils sont principalement destinés à l'indexation. En bref, certains des codes opération spécifient que l'adresse de la donnée sur laquelle doit porter l'opération doit être calculée par le microprocesseur en ajoutant à l'opérande qui spécifie l'adresse le contenu des registres X ou Y. Il suffit donc, pour accéder à plusieurs articles d'un tableau, de modifier le contenu du registre d'index approprié. L'indexation constitue donc un mode d'adressage.

L'adressage indirect

C'est une fonction extrêmement puissante qui, au lieu de calculer ou de modifier l'adresse recherchée, sélectionne parmi plusieurs adresses possibles l'opérande donnant l'adresse de l'instruction. Le principe de ce mode d'adressage est le suivant : l'adresse qui suit le code opération renvoie à un emplacement mémoire qui contient l'adresse recherchée. Cette dernière adresse donne accès à la donnée sur laquelle porte l'instruction.

Les registres du 65Ø2

Comme un grand nombre d'instructions utilisent les registres énumérés ci-dessus, nous commencerons par examiner ces registres.

— *Le registre A ou accumulateur*

Le registre A, également appelé accumulateur, sert à accumuler les résultats des opérations arithmétiques. Les opérations arithmétiques utilisant l'accumulateur se font très rapidement dans la mesure où elles sont effectuées dans le registre interne. Mais les données sur lesquelles sont effectuées ces opérations doivent être chargées dans l'accumulateur à partir de la mémoire. Inversement, une fois l'opération effectuée, les données traitées sont à nouveau chargées en mémoire.

— *Les registres X et Y ou registres d'index*

On utilise principalement ces registres pour la modification dynamique d'adresse. Quand ils servent d'index ils doivent d'abord être chargés ou initialisés. Certaines instructions permettent de modifier le contenu de ces registres, et d'autres leur affectent la fonction d'index.

— *Le registre S ou pointeur de pile*

Le registre S permet de repérer le prochain emplacement disponible dans une zone de mémoire spécialisée appelée *pile.* On peut définir la pile comme une liste séquentielle d'articles, qui sert aux appels et aux sorties de sous-programmes. On étudiera cette notion de

Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr Page 117 sur 163 15/08/2000

façon plus approfondie en liaison avec les instructions de gestion des sous-programmes. Le registre S donne toujours accès à la page 1 de la mémoire vive. De ce fait, le programmeur ne doit pas utiliser la page 1. Certaines instructions spécifiques utilisent la pile pour modifier le contenu de la page 1.

— Le registre P ou registre d'état du processeur

Ce registre est en fait un ensemble de huit drapeaux d'un bit, qui indiquent l'état de l'UCT une fois que l'instruction a été exécutée. Les drapeaux sont initialisés et réinitialisés automatiquement par l'exécution de l'instruction. On peut tester la valeur de ces différents drapeaux, et il existe une gamme d'instructions qui permettent de le faire.

Les drapeaux du registre P sont les suivants :
Bit 7 : drapeau N, mis à 1 si le résultat de l'opération en négatif.
Bit 6 : drapeau V, indicateur de dépassement, mis à 1 si un report se produit du bit 6 de l'accumulateur au bit 7.
Bit 5 : inutilisé.
Bit 4 : drapeau B, mis à 1 quand on exécute la commande BRK (Break).
Bit 3 : drapeau D, mis à 1 quand le processeur fonctionne en mode décimal ; mis à Ø quand il fonctionne en mode binaire.
Bit 2 : drapeau I ou signe d'interruption, mis à 1 en cas d'interruption, ou en cas d'instruction invalidant les interruptions ultérieures. Remis à Ø par une instruction spécifique.
Bit 1 : drapeau Z, mis à 1 quand le résultat d'une opération arithmétique ou d'un transfert de données est égal à zéro.
Bit Ø : drapeau C. Indique une retenue en arithmétique ou la présence d'un bit permuté ou sorti d'une adresse ou d'un registre.

Certains bits de ce registre peuvent être mis à un ou à zéro directement par le programmeur.

— Le registre PC ou compteur d'instruction

C'est le seul registre de seize bits ; il contient l'adresse de l'instruction qui suit celle qui est en cours d'exécution. Il n'est pas directement accessible par le programmeur, mais il peut le charger dans la pile et l'examiner dans la zone de la pile. L'UCT (unité centrale de traitement) utilise ce registre pour appeler l'instruction qui va devoir être exécutée ; le registre est automatiquement incrémenté à chaque nouvel appel d'un octet donnant l'adresse d'une instruction. Il est également modifié par les instructions de branchement, dès lors que ces instructions changent l'ordre d'exécution du programme.

Les modes d'adressage du 65Ø2

Après avoir étudié ces quelques notions liées à l'adressage, nous allons passer aux modes d'adressage admis par le 65Ø2.

— L'adressage implicite

On parle d'adressage implicite quand l'adresse est spécifiée par le code opération. Avec le 65Ø2, on considère que les instructions qui agissent directement sur des registres particuliers ont des adresses implicites (INX et INY, par exemple, sont respectivement des instructions d'incrémentation des registres X et Y) ; il s'agit donc d'instructions à un seul octet.

— L'adressage direct

On appelle adresse directe celle qui suit le code opération. Les instructions qui fonctionnent en mode direct comportent à la suite du code opération un octet unique de données, appelé littéral. Ce type d'instruction tient sur deux octets.

— L'adressage en page zéro

On appelle adresse en page zéro une adresse spécifiant un emplacement de mémoire situé en page zéro, c'est-à-dire dans les 256 premiers octets de la mémoire vive. Une instruction en page zéro comprendra donc deux octets.

— L'adressage indexé

Ce mode d'adressage utilise les registres d'index X et Y pour modifier l'adresse qui suit le code opération. Cette adresse peut comporter un octet (en page zéro) ou deux octets (à n'importe quel emplacement-mémoire). Les instructions comprendront donc deux ou trois octets.

— L'adressage absolu

Les adresses absolues sont des adresses de deux octets non-modifiées par l'indexation. Les instructions qui utilisent ce mode d'adressage comportent donc trois octets.

— L'adressage indirect

Dans l'adressage indirect pur, on utilise une adresse de deux octets qui suit le code opération et permet d'accéder à une adresse-donnée de deux octets située ailleurs dans la mémoire. Les deux adresses ne sont pas sujettes à modification par l'indexation.

Dans le jeu d'instructions du 65Ø2, une seule instruction utilise cette forme d'adressage : l'instruction JMP (instruction de saut ou de branchement) qui tient donc sur *trois* octets.

Numérisation http://wsc.n3.net
15/08/2000 Page 118 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

— *L'adressage indirect indexé*

Le 65Ø2 fait appel à deux formes d'adressage indirect indexé, qui utilisent toutes les deux les registres d'index et qui ne donnent accès qu'à la page zéro de la mémoire.

Le premier mode utilise le registre d'index X ; nous l'appellerons par la suite "adressage indirect X". On l'utilise *uniquement* pour l'adressage de tableaux en page zéro. Il serait plus exact de l'appeler "adressage indexé indirect", puisqu'on ajoute le registre d'index X à l'adresse en page zéro pour trouver l'emplacement qui contient le premier octet de l'adresse indirecte. On obtient ainsi l'adresse de deux octets utilisée pour accéder à la mémoire.

Le deuxième mode utilise le registre d'index Y ; nous l'appelerons "adressage indirect Y". Ce mode donne accès à des éléments spécifiques de tableaux qui se trouvent à un endroit quelconque de la mémoire centrale. On calcule l'adresse finale (contenant la donnée) en ajoutant le contenu de Y à l'adresse de seize bits indiquée par l'adresse en page zéro qui suit le code opération. Si l'adresse de seize bits trouvée en page zéro renvoie au début d'un tableau rangé dans la mémoire centrale, on peut alors pointer le registre d'index Y sur une entrée quelconque située dans une limite de 256 octets après le début du tableau (rappelez-vous que la dimension du registre d'index Y est d'un octet !). Puisque le code opération est toujours suivi d'une adresse en page zéro, les instructions en adressage indirect indexé tiennent sur deux octets.

— *L'adressage relatif*

L'adressage relatif se fonde sur la valeur détenue par le compteur d'instructions ; ce mode d'adressage est utilisé par un groupe d'instructions appelées *branchements conditionnels.* Ces instructions tiennent sur deux octets et indiquent un test à faire sur les drapeaux du registre P. L'adresse qui suit le code opération est une adresse d'un octet qui spécifie l'adresse relative de l'instruction à exécuter si la condition testée est remplie. Comme cette adresse n'a qu'un octet, le nombre d'adresses possibles est limité à 256. Comme la plupart des boucles sont assez courtes, l'octet d'adresse peut contenir des adresses de branchement *positives* et *négatives* (progressives et régressives). Pour ce faire, on utilise le bit fort de l'adresse comme bit de signe, ce qui permet un branchement progressif allant jusqu'à +127 octets et un branchement régressif allant jusqu'à −128 octets (on utilise le procédé de calcul en complément à 2). Comme les instructions de branchement tiennent sur deux octets, la limite réelle du branchement est de −126 (−128+2) et de +129 (+127+2), relativement à l'adresse de l'instruction de branchement.

Classement des instructions selon le mode d'adressage

Nous avons déjà vu qu'il existe plusieurs modes d'adressage pour les instructions du 65Ø2, et plusieurs codes opération pour chaque type d'instruction. Nous allons maintenant répertorier ces modes en donnant à chacun un nom abrégé ; de la sorte, au moment d'utiliser des instructions, il vous suffira de jeter un coup d'œil au tableau figurant à la fin de ce volume (Annexe 8) pour savoir immédiatement quels sont les modes d'adressage (et les codes opération) disponibles pour chaque instruction.

Mode	Longueur	Définition du mode
I	1	Implicite (à opérande implicite)
IM	2	Direct (le littéral suit le code opération)
ZP	2	Page zéro (adresse d'un octet)
ZX	2	Page zéro, indexé en X
ZY	2	Page zéro, indexé en Y
IX	2	Indirect X
IY	2	Indirect Y
RA	2	Relatif (pour l'instruction "branch" seulement)
AB	3	Absolu non-indexé
AX	3	Absolu, indexé en X
AY	3	Absolu, indexé en Y
IN	3	Indirect non-indexé (pour l'instruction **JMP** seulement)

Le tableau suivant indique le format des différents modes d'instruction. Chaque case représente un octet.

Écriture des instructions

I (implicite)	**Code op.**		
IM (direct)	**Code op.**	**Littéral**	
ZP (page zéro)	**Code op.**	**Adr. en ZP**	
ZX (ZP indexé en X)	**Code op.**	**Adr. en ZP**	
ZY (ZP indexé en Y)	**Code op.**	**Adr. en ZP**	
IX (ZP indirect X)	**Code op.**	**Adr. en ZP**	
IZ (ZP indirect Y)	**Code op.**	**Adr. en ZP**	
RA (relatif)	**Code op.**	**Adr. relat.**	
AB (absolu)	**Code op.**	**Adr. de**	**2 octets**
AX (absolu indexé en X)	**Code op.**	**Adr. de**	**2 octets**
AY (absolu indexé en Y)	**Code op.**	**Adr. de**	**2 octets**
IN (indirect pur)	**Code op.**	**Adr. ind. de**	**2 octets**

15/08/2000 Page 119 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

On peut également classer les instructions suivant les conséquences qu'entraîne leur exécution sur les registres et les emplacements mémoire. On peut enfin les classer suivant leur fonction.

Les pages suivantes sont consacrées aux instructions. Nous les grouperons d'abord par fonction, sous les rubriques "manipulation", "test", "arithmétique", "logique" et "commande". Les grands groupes d'instructions seront subdivisés en sous-groupes selon le même principe. On trouvera alors facilement, en se référant à l'annexe 8, le code-opération correspondant à chaque mnémonique d'instruction, classée suivant le mode d'adressage. Voyons maintenant le jeu d'instructions du 65Ø2.

Instructions de manipulation des données

Ces instructions servent à déplacer les données de la mémoire vers un des registres ou vice-versa, ou d'un registre à l'autre, ou à l'intérieur d'un même registre. Le 65Ø2 ne dispose pas d'instructions capables de déplacer directement les données d'un ensemble d'emplacements mémoire vers un autre, aussi faut-il toujours faire transiter les données par un registre. Les registres qui permettent d'effectuer cette opération ne contiennent qu'un octet à la fois, et de ce fait, on ne peut déplacer les données qu'octet par octet. Comme le BASIC nous a habitués aux variables numériques et aux chaînes de caractères, cela peut paraître laborieux et lent. Mais chaque déplacement ne prend que quelques microsecondes, et le programmeur peut avoir recours aux boucles pour éliminer un codage répétitif.

a) Instructions de chargement ("load")

Les instructions suivantes chargent le registre approprié avec un octet de données pris à l'emplacement mémoire spécifié par l'adresse suivant le code opération.

LDA Load A (accumulateur)
LDX Load X (registre d'index X)
LDY Load Y (registre d'index Y)

b) Instructions de mise en mémoire ("store")

Les instructions suivantes placent le contenu du registre à l'emplacement mémoire donné par l'adresse. C'est l'inverse de l'opération de chargement.

STA Store A (accumulateur)
STX Store X (registre d'index X)
STY Store Y (registre d'index Y)

Pour faire passer des données d'un emplacement mémoire à un autre (de 1ØØØ) (hexa) à 1ØØ1 (hexa), par exemple), on utilisera la suite d'instructions :

LDA $1ØØØ
STA $1ØØ1

Le signe $, en assembleur, signale une adresse hexa. Nous adopterons cette convention à partir de maintenant, ainsi que quelques autres. Notez que dans le cas ci-dessus, on peut également utiliser les registres d'index (quand ils ne servent pas d'index, bien entendu).

c) Instructions de transfert d'un registre à un autre

On utilise ces instructions pour faire passer directement les données d'un registre à un autre, sans utiliser d'emplacement mémoire comme lieu de rangement intermédiaire. Comme l'opération se fait entièrement entre registres, et que les mnémoniques (et les codes opération) suffisent à la spécifier, aucune adresse n'est nécessaire, les instructions tiennent sur un octet, et leur exécution est rapide.

TAX Transfert de A à X
TAY Transfert de A à Y
TXA Transfert de X à A
TYA Transfert de Y à A
TSX Transfert de S à X
TXS Transfert de X à S

Il n'existe pas d'instructions permettant un transfert de Y à S ou de X à Y.

d) Instructions de décalage

Ces quatre instructions manipulent les données à l'intérieur de l'accumulateur ou d'un octet unique de mémoire. Si l'on représente l'un ou l'autre de ces emplacements comme une chaîne de huit bits :

7	6	5	4	3	2	1	Ø

on décale cette chaîne d'un bit vers la gauche ou vers la droite de façon à faire tomber le bit de l'extrémité (soit 7, soit Ø, selon l'instruction employée) dans le bit de retenue du registre P. La position ainsi libérée est alors garnie d'un zéro ou du contenu préalable du bit de retenue (là encore, selon l'instruction employée).

William Saint-Cricq / wsaintcr@wanadoo.fr Page 120 sur 163 15/08/2000

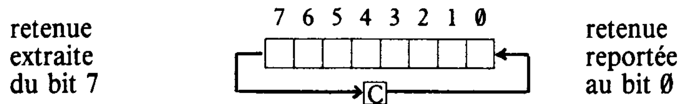
Numérisation http://wsc.n3.net

ASL Décalage arithmétique à gauche
LSR Décalage logique à droite
ROR Permutation circulaire à droite
ROL Permutation circulaire à gauche

L'accumulateur est le seul registre dont le contenu peut être décalé de façon immédiate. Notez que dans les deux types de décalage, les nombres négatifs ne sont pas pris en compte. L'instruction **LSR** (qui indique un nombre positif) fixe à Ø la valeur du bit 7, alors que l'instruction **ASL** lui donne la valeur précédente du bit 6 (soit **Ø**, soit 1, c'est-à-dire que le signe est positif ou négatif).

Les instructions de permutation, à la différence des instructions de décalage **ASL** et **LSR,** opèrent en fait des permutations sur neuf bits, puisque le contenu préalable du bit de retenue vient occuper **la** position de bit libérée.

La figure ci-dessous illustre ce processus dans le cas de l'instruction **ROL**.



Il existe quatre autres instructions qu'on peut considérer comme des instructions de manipulation, mais on les abordera avec les instructions de commande, parce qu'elles portent sur la manipulation de la pile.

Instructions de test

C'est à l'aide de ce jeu d'instructions qu'il est possible de programmer réellement, puisqu'elles permettent de faire appel à différents blocs de programme à la suite de décisions programmées. L'équivalent en BASIC est le **IF...THEN GOTO...** ; on peut donc les consédérer comme des branchements conditionnels. Certaines instructions font néanmoins exception à cette définition. Les instructions de comparaison testent, mais ne comportent pas de branchement ; elles positionnent dans le registre P les drapeaux utilisés par les instructions de branchement conditionnel.

Précisons dès maintenant que la plupart des instructions du 65Ø2 (mais pas toutes) agissent sur les drapeaux du registre P. Reportez-vous au paragraphe sur les drapeaux du registre P pour mieux comprendre ce qui suit. Le tableau des codes opération donné à l'Annexe 8 indique sur quel drapeau agit chaque instruction, et permet donc de choisir les instructions de branchement conditionnel appropriées, lors de la rédaction d'un programme.

a) branchements conditionnels

		Registre P
BCC	Branchement si la retenue = Ø	C
BCS	Branchement si la retenue = 1	C
BEQ	Branchement en cas d'égalité à Ø (drapeau Z = 1)	Z
BMI	Branchement en cas de signe moins (drapeau N = 1)	N
BNE	Branchement en cas de non égalité à zéro (drapeau Z = Ø)	Z
BPL	Branchement en cas de signe plus (drapeau N = Ø)	N
BVC	Branchement si le dépassement = Ø (drapeau V = Ø)	V
BVS	Branchement si le dépassement = 1 (drapeau V = 1)	V

Toutes ces instructions de branchement conditionnel utilisent uniquement des adresses relatives, c'est-à-dire des adresses d'un octet située entre + 129 et − 126 par rapport à l'octet de l'instruction en cours d'exécution.

b) Instructions de comparaison

CMP Comparaison de la donnée avec le contenu de l'accumulateur
CPX Comparaison de la donnée avec le contenu du registre d'index X
CPY Comparaison de la donnée avec le contenu du registre d'index Y

La valeur affectée par ces instructions aux drapeaux Z et C du registre P peut alors être testée au moyen d'instructions de branchement conditionnel :

Registre < donnée : C = Ø : Test par **BCC**
Registre = donnée : Z = 1 : Test par **BEQ**
Registre > = donnée : C = 1 : Test par **BCS**
Registre > donnée : Z = 1 et C = 1 : Test par **BEQ** suivie de **BCS**

William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Numérisation http://wsc.n3.net

c) Comparaison de bits

BIT Comparaison et positionnement des bits d'état

L'instruction **BIT** permet de comparer la donnée en mémoire et le contenu de l'accumulateur, et met à 1 le drapeau Z s'ils sont égaux. Les bits 7 et 6 de la donnée en mémoire sont transférés au registre d'état (drapeaux N et V respectivement). L'accumulateur reste inchangé.

Instructions arithmétiques

Le 65Ø2 ne met en œuvre que deux types d'instructions arithmétiques. Il s'agit des opérations d'addition ou de soustraction. Pour la multiplication et la division, il faut faire appel à des sous-programmes, sur lesquels peuvent influer, bien sûr, des additions ou des soustractions répétées.

a) Instructions arithmétiques

ADC	Addition avec retenue
INC	Incrémentation de la mémoire (addition de 1)
INX	Incrémentation du registre X (addition de 1)
INY	Incrémentation du registre Y (addition de 1)
SBC	Soustraction avec retenue
DEC	Décrémentation de la mémoire (soustraction de 1)
DEX	Décrémentation du registre X (soustraction de 1)
DEY	Décrémentation du registre Y (soustraction de 1)

Il faut expliquer de façon un peu plus détaillée les instructions **ADC** et **SBC**, dont le fonctionnement fait appel à la valeur en cours du drapeau C (drapeau de retenue). **ADC** ajoute au contenu de l'accumulateur la donnée qui se trouve à l'adresse spécifiée, plus la valeur de la retenue. **SBC** soustrait de l'accumulateur la donnée moins l'inverse du drapeau C (1 si la retenue = Ø, Ø si la retenue = 1). Quand on commence une série d'opérations arithmétiques liées entre elles, le drapeau C doit être mis à zéro avant le premier **ADC** et mis à un avant la première soustraction. On y parvient au moyen des instructions suivantes :

CLC Mise à zéro du drapeau C
SEC Mise à un du drapeau C

Les autres instructions sont relativement simples, et ne mettent pas en jeu le drapeau C. L'incrémentation augmente de 1 la valeur du registre ou de l'emplacement mémoire, la décrémentation la diminue de 1. Remarquez l'absence d'instructions directes d'incrémentation/décrémentation du contenu de l'accumulateur.

b) Mode décimal

Il existe deux autres instructions en rapport avec le calcul arithmétique, mais qui n'effectuent pas d'opérations arithmétiques :

SED Mise en mode décimal
CLD Sortie du mode décimal (mise en mode binaire)

Décimal ? Mais oui, le 65Ø2 est capable de calculer en décimal, comme vous l'avez peut-être compris d'après la description des drapeaux du registre P. Les calculs en décimal se font sur des données en décimal codé binaire, c'est-à-dire que chaque octet comporte *deux* nombres (entre Ø et 9) en **DCB.** Quand on passe du décimal au binaire, il est essentiel de se mettre en mode décimal avant de faire des calculs en décimal, et d'en sortir avant d'effectuer des opérations en binaire. Essayez d'additionner deux nombres DCB en mode binaire pur et vérifiez le résultat (prenez deux nombres supérieurs à 5).

Instructions logiques

Il existe trois instructions logiques qui agissent au niveau du bit. On les utilise pour tester des configurations de bits ; elles positionnent un drapeau qui indique le résultat du test. Ce résultat est soit VRAI soit FAUX ; pour mieux comprendre comment ces instructions fonctionnent, il faut se reporter à un tableau appelé table de vérité. Les trois instructions sont les suivantes :

AND	ET LOGIQUE donnée/accumulateur
ORA	OU INCLUSIF donnée/accumulateur
EOR	OU EXCLUSIF donnée/accumulateur

AND

Examinons d'abord **AND** au moyen de la table de vérité suivante. **AND** et représenté par le symbole ∧.

A	B	A∧B
Ø	Ø	Ø
Ø	1	Ø
1	Ø	Ø
1	1	1

15/08/2000 Page 122 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

A est un bit de l'accumulateur et B est le bit de donnée correspondant. A∧B se lit "A ET B" ; c'est le résultat de l'opération logique. Un résultat égal à zéro est FAUX ; un résultat égal à 1 est VRAI.

En consultant la table, on constate que la seule combinaison de valeurs dont le résultat est VRAI est la dernière, où les deux bits sont à 1. Toutes les autres combinaisons donnent Ø, c'est-à-dire FAUX.

Les bits de l'accumulateur sont modifiés en conséquence et prennent la valeur résultant de A∧B. En général, cette instruction sert à mettre à Ø certains bits d'un octet.

Supposons que le bit Ø de l'accumulateur ait été programmé pour jouer le rôle d'un drapeau (ou symbole de renvoi), et que nous voulions mettre le bit à Ø sans agir sur les autres bits. Nous y parviendrons en procédant à l'opération logique ET (**AND**) entre l'accumulateur et le nombre binaire 1111111Ø (FE hexa).

Bits :	7	6	5	4	3	2	1	Ø	
	1	1	Ø	1	Ø	Ø	Ø	1	Accumulateur avant l'opération
				AND					
	1	1	1	1	1	1	1	Ø	Donnée-masque
Résultat :	1	1	Ø	1	Ø	Ø	Ø	Ø	Accumulateur après l'opération

Confrontez le résultat bit par bit avec la table de vérité. Vous constaterez que *seul* le bit Ø a été modifié. La "donnée-masque" est une donnée qui sert uniquement à modifier le contenu de l'accumulateur.

ORA

L'instruction ORA sert à mettre les bits à 1. La table de vérité suivante correspond à cette instruction :

A	B	A∨B
Ø	Ø	Ø
Ø	1	1
1	Ø	1
1	1	1

Dans ce cas, si l'un des deux bits A et B ou les deux sont égaux à 1, le résultat est égal à 1. Reprenons l'exemple précédent : nous voulons remettre le bit Ø à 1.

Bits :	7	6	5	4	3	2	1	Ø	
	1	1	Ø	1	Ø	Ø	Ø	Ø	Accumulateur avant l'opération
				ORA					
	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	Donnée-masque
Résultat	1	1	Ø	1	Ø	Ø	Ø	1	Accumulateur après l'opération

Le symbole représente le OU inclusif. On peut définir l'opération ci-dessus comme "accumulateur∨masque".

EOR

L'instruction EOR (OU exclusif) donne un résultat VRAI (égal à 1) si un et un seul des deux bits en cause est égal à 1 : il faut donc qu'un bit soit à 1 et l'autre à Ø. Voici la table de vérité correspondante :

A	B	A∨B
Ø	Ø	Ø
Ø	1	1
1	Ø	1
1	1	Ø

15/08/2000

Page 123 sur 163

William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

∨ est le symbole qui représente le OU exclusif. Cette instruction permet de mettre les bit à 1 s'ils sont à 0, et à 0 s'ils sont à 1 : elle change l'état d'un bit, qu'on peut alors utiliser comme une "bascule", par exemple.

	7	6	5	4	3	2	1	0	
Bits	1	0	1	0	1	0	1	0	Accumulateur avant l'opération
	EOR								
	1	1	1	1	1	1	1	1	Donnée-masque
Résultat	0	1	0	1	0	1	0	1	Accumulateur après l'opération

Dans cet exemple, une seule opération a permis de modifier plusieurs bits. On peut aussi y parvenir avec **AND** et **ORA.** Quel serait l'effet d'**EOR** sur l'accumulateur ci-dessus avec un masque de 00000000 (00 hexa) ?

Instructions de commande

Toutes ces instructions, à l'exception de **NOP, CLI** et **SEI,** servent à modifier le déroulement d'un programme, en agissant sur l'ordre d'exécution.

- **NOP** Pas d'opération
- **JMP** Saut ou branchement inconditionnel
- **JSR** Branchement sur un sous-programme
- **RTS** Retour au programme principal après exécution d'un sous-programme
- **PHA** Place A sur la pile
- **PHP** Place P sur la pile
- **PLA** Retire A de la pile
- **PLP** Retire P de la pile
- **BRK** Break (interruption)
- **RTI** Sortie de l'interruption
- **SEI** Mise à 1 du drapeau d'invalidation de l'interruption
- **CLI** Mise à 0 du drapeau d'invalidation de l'interruption

a) **NOP**

Cette instruction d'un octet sert à retarder l'exécution d'un programme (le retard introduit est de deux cycles) ou à éliminer des instructions, si l'on désire comprimer une partie du programme. Bref, c'est une instruction qui ne fait rien, et qui s'avère extrêmement utile. Lorsqu'on l'utilise pour éliminer des instructions, elle doit recouvrir la totalité des octets d'instruction ; il est donc parfois nécessaire d'avoir recours à deux ou trois instructions **NOP**.

b) **JMP**

C'est l'équivalent en langage-machine du **GOTO BASIC.** Le déroulement du programme reprend directement à l'adresse spécifiée par l'instruction.

c) **JSR** et **RTS**

Cest instructions comptent parmi les plus importantes du 6502. **JSR** provoque un branchement sur un sous-programme situé ailleurs dans le cours du programme, et **RTS** permet de revenir à l'instruction qui suit le branchement. Elles sont équivalents aux instructions **GOSUB** et **RETURN** du **BASIC.** Pour comprendre leur fonctionnement, il faut connaître celui de la pile.

Une pile, comme son nom l'indique, est une structure composée d'objets empilés les uns sur les autres. En l'occurrence, les objets sont des adresses, prélevées dans le registre PC (compteur d'instructions). A quoi la pile sert-elle, et comment l'utilise-t-on ? On le comprendra mieux à partir d'un exemple. On trouvera à la page suivante un exemple d'ordre d'exécution d'un programme en code machine.

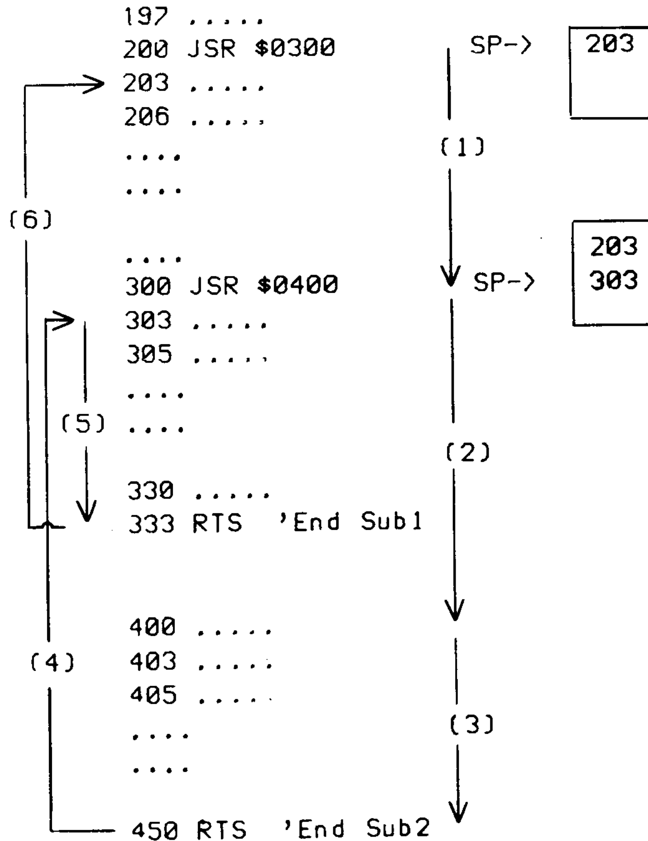
Prenons donc l'exemple suivant. Une instruction **JSR** à l'emplacement $0200 appelle un sous-programme situé à l'emplacement $0300. Au moment de l'appel, le contenu du registre PC est $0203 ; c'est l'adresse de l'instruction qui suit le branchement par **JSR.** Le 6502 range cette adresse dans la pile et "saute" jusqu'à l'emplacement $0300. A cet emplacement, un autre **JSR** renvoie à un sous-programme en $0400. Il faut à nouveau conserver le contenu du registre PC ($0303) en le rangeant dans la pile à la suite de la première adresse.

Le sous-programme $0400 est complètement exécuté ; il se termine par une instruction **RTS.** La dernière adresse à avoir été placée dans la pile ($0303) est, sous l'effet de **RTS**, chargée dans le compteur d'instructions, et le programme se branche sur le contenu du registre PC, c'est-à-dire sur l'instruction qui suit le deuxième branchement par **JSR.** De même, une fois que le sous-programme $0300 s'est achevé sur un **RTS,** le PC reçoit l'adresse $0203 et l'exécution reprend à partir de cette adresse.

William Saint-Cricq / wsaintcr@wanadoo.fr Page 124 sur 163 15/08/2000

Numérisation http://wsc.n3.net

Il ne reste qu'un point d'interrogation : comment est-ce la bonne adresse qui est choisie dans la pile ? Eh bien, c'est grâce au registre S, ou pointeur de pile, qui pointe toujours l'emplacement de la pile qui va être disponible, et qui est mis à jour automatiquement par les instructions **JSR** et **RTS, JSR** déplaçant le pointeur vers l'avant et **RTS** le faisant remonter.



Précisons enfin que la pile du 6502 est résidente en page 1 de la mémoire : de 100 hexa à 1FF hexa. Comme le pointeur de pile est un registre de 8 bits, le 1 de tête entre en jeu dans les deux adresses, c'est-à-dire que le 6502 suppose toujours la présence d'un bit supplémentaire en tête de l'adresse. Par ailleurs, la pile du 6502 est à l'envers, c'est-à-dire que le premier emplacement de pile disponible est en bas de la pile. Chaque instruction **JSR** décrémente de 2 le pointeur de pile (les adresses emmagasinées étant des adresses absolues de 16 bits en provenance du registre PC).

On a dans l'exemple ci-dessus deux **JSR** imbriqués, le second **JSR** étant émis à partir du sous-programme appelé par le premier **JSR.** Comme la pile contient 256 octets (une page), et que le PC en contient deux, 128 niveaux d'imbrication sont possibles. Il arrive très rarement, dans la pratique, qu'un tel degré d'imbrication soit atteint ; le programme peut donc utiliser, si besoin est, les emplacements de la page 1 situés en-dessous de l'adresse fixée comme borne inférieure.

Instructions de placement et de prélèvement sur la pile ("Push" et "Pull")

Ces quatre instructions placent et prélèvent sur la pile les bits provenant de l'accumulateur et du registre P (bits d'état). Il est parfois nécessaire de préserver les drapeaux d'état du registre P pour pouvoir les récupérer tels qu'ils étaient avant un appel de sous-programme. L'instruction **PHP** place sur la pile le contenu du registre P et met à jour le pointeur de pile. **PLP** récupère le contenu du registre P à l'adresse de pile indiquée par le pointeur de pile, après quoi elle remet à jour le pointeur de pile.

On peut de même préserver et récupérer le contenu de l'accumulateur au moyen des instructions **PHA** et **PLA.** Rappelez-vous que l'instruction **RTS** prend au sommet de la pile les deux derniers octets à y avoir été placés, et veillez à l'ordre des instructions "Push" et "Pull", pour éviter que **RTS** n'obtienne une information erronée en essayant de se brancher sur le contenu préservé dans la pile de l'accumulateur ou du registre P.

Interruptions

Un examen complet des Interruptions dépasserait les limites de ce chapitre, et nous nous contenterons donc d'effleurer ce sujet. Le lecteur qui désirerait étudier ce domaine de façon plus approfondie peut se reporter aux nombreux textes de référence consacrés au 6502.

En résumé, le 6502 dispose de deux types d'interruptions : l'une est liée au matériel, l'autre au logiciel. Quand une interruption survient, le contenu du registre PC et celui du registre P sont placés dans la pile, et un branchement se fait vers une adresse indirecte située en haut de la mémoire. Ces adresses sont appelées vecteurs, et on parle alors d'interruption vectorisée. Le sous-programme trouvé à l'adresse finale prend en charge l'interruption, puis il revient à l'endroit où l'exécution normale du programme s'est interrompue au moyen d'une instruction **RTI** qui récupère dans la pile les registres P et PC et met à jour le pointeur de pile.

15/08/2000
Page 125 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr

L'instruction **BRK** est une instruction d'interruption programmée, qui provoque un branchement vers l'adresse indirecte située aux emplacement FFFE, FFFF. Ces adresses se trouvent normalement dans la ROM de l'ORIC (mémoire morte), et ne sont donc pas directement modifiables. Mais nous reparlerons de **BRK** un peu plus loin.

Les deux dernières instructions sont **SEI** et **CLT.** Ces instructions permettent de mettre à 1 ou à Ø le drapeau I du registre P (drapeau d'invalidation des interruptions). Quand il est mis à 1, toutes les interruptions sont invalidées, c'est-à-dire qu'il n'en est pas tenu compte, à l'exception d'un type d'interruption matérielle : la **NMI,** ou interruption non-invalidable ("non-maskable interrupt"). Il est nécessaire d'invalider les interruptions quand une procédure doit être menée à bien dans un laps de temps limité, généralement très bref, parce qu'il s'agit d'une procédure à durée critique.

Les conventions et les règles de la programmation en code machine

Nous avons maintenant fait le tour du jeu d'instructions du 65Ø2, et nous sommes prêts à nous lancer dans la programmation. Nous y serons aidés par un certain nombre de conventions.

Nous avons déjà parlé du $, qui signale les données en hexadécimal. Le #, lui, signale un littéral, c'est-à-dire un opérande en adressage direct ; quant au signe %, il indique qu'une donnée est en binaire.

La programmation en code machine comporte deux étapes : le niveau symbolique et le niveau code. Par exemple :

LDA # $ØØ
(niveau symbolique) : charge le littéral ØØ hexa dans l'accumulateur

A9ØØ
(niveau code) : **LDA** directe = A9 hexa

D'ordinaire, on met au point le programme au niveau symbolique, puis on le traduit en code. C'est ce qu'on appelle l'assemblage ; si le programmeur effectue lui-même cette traduction, on parle d'assemblage à la main.

REMARQUE IMPORTANTE

Jusqu'à présent, nous avons présenté toutes les adresses hexa de deux octets sous la forme $hhll, hh étant l'octet de poids fort ("high order") et ll l'octet de poids faible ("low order"). Cette présentation convient parfaitement au niveau symbolique. Mais pour être comprises par le 65Ø2, les adresses de deux octets doivent être écrites dans l'ordre inverse, et c'est ce qu'il faut faire en code machine. Une adresse sera donc codée sous la forme $llhh. Par exemple :

LDA $1Ø3E (charge l'accumulateur avec le contenu de l'adresse absolue 1Ø3E)
devient, une fois codé :
AD 3E1Ø (**LDA** code op. absolu = AD)

Il faut s'en souvenir lorsqu'on passe de la forme symbolique à la forme codée. Commençons maintenant à programmer, en prenant comme exemple des programmes en code machine d'utilité courante.

Quelques exemples de code machine

La plupart des programmes font appel au calcul arithmétique, et nous proposons donc d'abord des séries d'instructions à caractère arithmétique. Nous commencerons par l'addition de deux nombres binaires d'un octet, situés aux adresses 5ØØ hexa et 5Ø1 hexa, le résultat devant être stocké en 5Ø2 hexa, en supposant que la somme tient en un octet.

CLC	Mise à Ø du drapeau de retenue
LDA $5ØØ	Charge le premier nombre dans l'accumulateur
ADC $5Ø1	Ajoute le second nombre à l'accumulateur
STA $5Ø2	Met le résultat en mémoire

Facile, n'est-ce pas ? Remarquez l'emploi de **CLC** avant **ADC.**

Essayons encore, cette fois avec deux nombres binaires de trois octets, situés aux emplacements $5Ø1-$5Ø3 et $5Ø4-$5Ø6, le résultat de l'opération étant mis en mémoire en $5Ø7-$4Ø9, et en supposant, comme dans l'exemple précédent, que le résultat tient en trois octets.

Nous pourrions répéter les trois dernières instructions pour chaque octet à ajouter, mais ce ne serait guère efficace. Nous aurons donc recours à l'indexation. Nous supposerons aussi, dans le cas présent, que les nombres sont mis en mémoire en commençant par l'octet de poids fort (adresse inférieure), comme on le fait conventionnellement. On pourrait les mettre en mémoire dans l'autre sens, en commençant par l'octet de poids faible, à la manière des adresses

Numérisation http://wsc.n3.net
15/08/2000 Page 126 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

en deux octets : cela dépend du programmeur. Voilà le programme en question :

Adresse	Instruction	
6ØØ	**CLC**	Met à Ø le drapeau de retenue
6Ø1	**LDX #$Ø3**	Charge Ø3 (littéral) dans X
6Ø3	**LDA $5ØØ,X**	Charge le contenu de l'adresse $5Ø3 + X
6Ø9	**STA $5Ø6,X**	Met en mémoire le résultat (adresse $5Ø6 + X)
6ØC	**DEX**	Décrémente X de 1
6ØD	**BNE* – 12**	Retourne à $6Ø3 si X < >Ø

Les deux programmes ci-dessus ont été écrits sous une forme symbolique : il faudra les traduire en code machine avant de les utiliser. Le # indique que les caractères suivants constituent un opérande littéral, qui sera mis en mémoire en même temps que le reste de l'instruction, sous forme d'adresse immédiate. "X" indique que l'adresse précédente doit être indexée au moyen du registre X : l'adresse de donnée finale est égale à la somme de la dernière adresse et du contenu du registre X. Dans la dernière instruction, "*" se rapporte au contenu du compteur d'instruction (registre PC) une fois que l'instruction a été extraite et décodée, mais avant son exécution. Voyons maintenant le même programme en code machine.

Adresse	Mode	
6ØØ	18	Implicite
6Ø1	A2Ø3	Direct
6Ø3	BDØØ5Ø	Absolu, X
6Ø6	7DØ35Ø	Absolu, X
6Ø9	9DØ65Ø	Absolu, X
6ØC	CA	Implicite
6ØD	DØF4	Relatif

Pour comprendre l'enchaînement logique de ce processus, on peut envisager le problème sous forme de schéma, en utilisant la méthode de l'organigramme. Il est recommandé d'élaborer un organigramme avant d'entreprendre un travail de programmation en code, car cela permet de visualiser les impératifs logiques, et la tâche du programmeur s'en trouve simplifiée. Ce travail préalable fait vraiment gagner beaucoup de temps. Si cela vous intéresse, reportez-vous à un texte de caractère général sur la programmation.

Vous avez pu remarquer qu'on décrémentait l'index pour adresser les octets de données suivant un ordre ascendant. Le tableau suivant aide à comprendre pourquoi :

Adresse	Index X	Adresse finale
5ØØ	3	5Ø3
5ØØ	2	5Ø2
5ØØ	1	5Ø1

Supposons le nombre $12345 stocké aux adresses $5Ø1-$5Ø3 : l'octet de poids faible ($45) est pris en premier, puis l'octet du milieu ($23), puis l'octet de poids fort ($1), ce qui est l'ordre correct pour effectuer une addition.

Si les nombres étaient mis en mémoire de la même façon que les adresses (comparez les instructions de l'adresse $6Ø3 dans les deux exemples : sous forme symbolique et sous forme codée), il faudrait incrémenter de 1 le registre (le nombre $12345 serait mis en mémoire sous la forme $54321), et utiliser une instruction de comparaison pour tester une valeur d'index de 3. En procédant comme ci-dessus, on économise une instruction, et cette méthode est plus facile à suivre. Vous trouverez à l'Annexe 8 un tableau des instructions ou codes opération pour chaque mode d'adressage.

On peut rédiger de la même manière des programmes de soustraction, en prenant comme modèle les programmes d'addition. Mais pensez à utiliser **SEC** (met la retenue à 1) au lieu de **CLC.** On peut également utiliser la complémentation à 2 et effectuer la soustraction en ajoutant un nombre négatif.

On peut effectuer les multiplications et les divisions par addition répétitive. Mais il est plus efficace d'utiliser la méthode de l'addition décalée. Vous trouverez dans les ouvrages de base (Leventhal ou Zaks) les programmes appropriés.

Puisque le présent ouvrage est un guide de l'ORIC, nous allons passer à des programmes conçus spécifiquement pour cette machine. La manipulation de l'écran constitue un bon point de départ : nous allons donc élaborer un programme destiné à déplacer un caractère sur l'écran, en utilisant la commande par clavier. Deux procédés sont possibles : on peut, soit accéder directement à la zone écran, soit utiliser les programmes propres de l'ORIC en les appelant à partir de notre programme en code machine. Si l'on utilise l'accès par clavier, il sera plus simple d'avoir recours à la deuxième méthode.

Le programme ci-dessous utilise les touches fléchées de l'ORIC pour déplacer un caractère autour de l'écran dans la direction de la flèche. Nous limiterons le déplacement du caractère aux lignes 2 à 25 et aux colonnes 2 à 39, en évitant les deux premières lignes et les colonnes d'attribut.

William Saint-Cricq / wsaintcr@wanadoo.fr Page 127 sur 163 15/08/2000

L'écran **TEXT** comporte 27 lignes de 4Ø colonnes, mais les colonnes de gauche sont réservées aux couleurs et on ne les utilise généralement pas dans les modes **TEXT** ou **LORES**.

L'écran occupe les emplacements mémoire $BB8Ø à $BFEØ (sur les machines de 48K) ou $3B8Ø à $3FEØ (sur les machines de 16K). Le programme ci-dessous est rédigé pour une machine de 48K. Pour l'adapter à une machine de 16K, il suffit de soustraire 8ØØØ hexa de toutes les adresses écran.

```
100 HIMEM32767    '...$7FFF
101 '
110 FOR X=0 TO 113
120 :    READ CODE
130 :    POKE 32768+X,CODE
140 NEXT X
150 '
160 CLS:PRINT"DEPL:FLECHES....ARRET:BA
RRE D'ESP."
170 POKE 0,#D0:POKE 1,#BB 'AFFICHAGE
180 POKE 2,#02:POKE 3,#02 'COL/RANG
190 POKE #24E,1:POKE#24F,1'KBD
200 '
210 FOR X=1 TO 10
220 :    CALL#8000
230 NEXT X
240 IF KEY$<>" " THEN 210
250 POKE #24E,32:POKE #24F,4
260 STOP
1000 'adresses op          ;SYMB. ADR.
1010 DATA #D8              'CLD        8000
1020 DATA #A9,#20          'LDA$20        1
1030 DATA #A4,#02          'LDY COL       3
1040 DATA #91,#00          'STA($00),Y    5
1050 DATA #20,#3B,#02      'JSR GTORKB    7
1060 DATA #10,#51          'BPL DISPLY    A
1070 DATA #A4,#02          'LDY COL
1080 DATA #C9,#08          'CMP $08
1090 DATA #90,#4B          'BCC DISPLY
1100 DATA #F0,#0D          'BEQ LEFT
1110 DATA #C9,#0B          'CMP #$11
1120 DATA #F0,#19          'BEQ UP
1130 DATA #B0,#43          'BCS DISPLY
1140 DATA #C9,#09          'CMP #$09
1150 DATA #F0,#0B          'BEQ RIGHT
1160 DATA #4C,#41,#80      'JMP DOWN
1170 DATA #C0,#02          'LEFT CPY#$02
1180 DATA #F0,#38          'BEQ DISPLY
1190 DATA #88              'DEY
1200 DATA #4C,#5D,#80      'JMP DISPLY
1210 DATA #C0,#27  'RIGHTCPYY#$27(39)
1220 DATA #F0,#30          'BEQ DISPLY
1230 DATA #C8              'INY
1240 DATA #4C,#5D,#80      'JMP DISPLY
1250 DATA #A6,#03          'LDX ROW
1260 DATA #E0,#02          'CPX #$02
1270 DATA #F0,#26          'BEQ DISPLY
1280 DATA #C6,#03          'DEC ROW
1290 DATA #EA,#EA          'NOP,NOP
1300 DATA #20,#64,#80      'JSR SUB
1310 DATA #4C,#5D,#80      'JMP DISPLY
1320 DATA #A6,#03          'DOWN LDX ROW
1330 DATA #E0,#1B          'CPY #$1B
1340 DATA #F0,#18          'BEQ DISPLY
1350 DATA #E6,#03          'INC ROW
1360 DATA #A9,#28          'LDA #$28 (40)
1370 DATA #20,#51,#80      'JSR ADD
1380 DATA #4C,#5D,#80      'JMP DISPLY
1390 DATA #18              'ADD CLC
1400 DATA #65,#00          'ADC $00
1410 DATA #85,#00          'STA $00
1420 DATA #A9,#00          'LDA #$00
1430 DATA #65,#01          'ADC $01
1440 DATA #85,#01          'STA $01
1450 DATA #60              'RTS
1460 DATA #A9,#58    'DISLPLYDA#$58 "X"
1470 DATA #91,#00          'STA($00),Y
1480 DATA #84,#02          'STY COL
1490 DATA #60              'RTS
1500 DATA #38              'SUB SEC
1520 DATA #A5,#00          'LDA $00
1530 DATA #E9,#28          'SBC #$28
1540 DATA #85,#00          'STA $00
1550 DATA #A5,#01          'LDA $01
1560 DATA #E9,#00          'SBC #$00
1570 DATA #85,#01          'STA $01
1580 DATA #60              'RTS
```

Explication du programme

La ligne 1ØØ définit la limite supérieure de mémoire pour un programme en BASIC. Dans cet exemple, nous avons réservé environ 6K aux instructions en code machine ; en fait, on n'utilise que 114 octets, et ce nombre a été choisi pour que le programme commence à 8ØØØ hexa. Dans la pratique, quand on insère dans des programmes en BASIC des blocs de programme en code machine, il vaut mieux soustraire de la position HIMEM en cours la longueur des programmes en code machine, ce que l'on fait au moyen d'un **PRINT DEEK** (A6), en utilisant ensuite HIMEM comme ci-dessus

Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr Page 128 sur 163 15/08/2000

pour fixer la limite du BASIC.

Les lignes 110 à 140 constituent une boucle qui extrait le code machine des instructions **DATA** et le charge aux adresses spécifiées. Cette suite d'instructions est appelée **LOADER,** ou programme de chargement ; c'est la façon classique d'utiliser le BASIC pour charger des programmes en code machine.

La ligne 160 efface l'écran.

La ligne 170 positionne la première adresse d'écran ($BBD0 dans l'exemple choisi) à des emplacements en page 0 connus (0, +1 hexa). Remarquez que l'adresse est présentée "à l'envers".

La ligne 180 positionne les limites inférieures de colonne et de ligne aux emplacements en page 0 2 et 3.

La ligne 190 fixe aux vitesses les plus élevées les cadences de temporisation et de répétition du clavier. Il s'agit de constantes système du système d'exploitation de l'ORIC ; on en trouvera d'autres à l'annexe 8.

Les lignes 210 à 230 constituent une boucle destinée à appeler dix fois le sous-programme en code machine. Remarquez qu'en BASIC, # indique un nombre en hexadécimal, alors qu'en assembleur (code symbolique), il signale un littéral. La ligne 240 teste la présence d'un caractère "espace", utilisé dans ce programme pour revenir au programme principal. Quand cette condition se réalise, la ligne 250 redonne leurs valeurs originelles de 32 et de 4 aux cadences de temporisation et de répétition du clavier, et interrompt l'exécution du programme.

Nous arrivons maintenant au programme en code machine lui-même. Pour plus de clarté, on a consacré une ligne d'instruction à chaque **DATA,** et le code symbolique (en langage assembleur) figure à chaque ligne sous forme de commentaire.

Les lignes 1010 à 1160 testent les caractères frappés au clavier en recherchant les codes de commande du curseur (touches fléchées), et se branchent, lorsque le test est rempli, sur les sous-programmes appropriés (dénommés **LEFT, RIGHT, UP** and **DOWN**, c'est-à-dire "à gauche", "à droite", "en haut" et "en bas") ; la frappe d'un autre caractère provoque une sortie sur le sous-programme d'affichage ("**DISPLAY**") qui intervient en fin de programme. Les premières instructions définissent le mode de numération **(CLD)** et effacent de l'écran le caractère qui s'y affichait auparavant ("X"). La ligne 1040 utilise l'adressage indirect Y pour recouvrir le "X" avec un blanc en indexant le pointeur d'écran en 00401 hexa avec la position de colonne en 02 hexa. La ligne 1050 appelle le programme-système clavier (pour plus de détails, se reporter à l'Annexe 9) et lui fait lire le clavier et ranger dans le registre A le caractère frappé.

La ligne 1010 recharge la position de colonne dans le registre Y en vue d'une utilisation ultérieure.

Les lignes 1170 à 1240 constituent les sous-programmes **LEFT** et **RIGHT** ; elles décrémentent ou incrémentent de 1 la position de colonne, après un test portant sur les valeurs de colonne minimum et maximum.

Les lignes 1250 à 1360 constituent les sous-programmes **UP** et **DOWN**. Elles modifient l'adresse de ligne et testent les valeurs minimum et maximum. Si la valeur envisagée se situe entre ces limites, le sous-programme appelle au moyen d'une instruction **JSR** les sous-programmes de soustraction et d'addition (qui vont respectivement de la ligne 1370 à la ligne 1430 et de la ligne 1480 à la ligne 1550 et ces sous-programmes modifient l'adresse d'écran aux emplacements 00+01 en lui appliquant une valeur de 40 décimal (caractères/ligne). Lors de la sortie de ces sous-programmes, un branchement JMP se fait vers le sous-programme **DISPLAY**, le dernier du programme.

Les lignes 1440 à 1470 constituent le sous-programme **DISPLAY**. Le registre A est chargé avec le code ASCII de "X" et l'affiche à l'écran au moyen d'une instruction **STA** indirect-Y, Y donnant la colonne et modifiant l'adresse d'écran (en 00+01 hexa). La valeur de Y est mise à jour puis placée à nouveu en 02 hexa, et la sortie du programme s'effectue par un **RTS** qui ramène au programme BASIC ayant appelé le sous-programme, en ligne 230.

Remarquez la ligne 280 : deux instructions **NOP** y sont utilisées pour recouvrir une autre instruction. Sans l'instruction **NOP,** il aurait fallu modifier par 2 toutes les adresses d'instruction données en-dessous de cette ligne, ce qui aurait pris beaucoup de temps. Notez l'utilisation de l'adressage relatif dans les instructions de branchement ; essayez de les décomposer en vous aidant des adresses symboliques.

Et maintenant, entrez le programme et utilisez les touches fléchées pour déplacer le "X" sur l'écran.

CALL

Dans le programme ci-dessus, l'instruction **BASIC CALL** permet d'appeler un sous-programme en code machine ; il suffit de donner dans la ligne d'instruction **CALL** l'adresse du programme appelé, en décimal ou en hexa (précédée du signe #). Il existe d'autres façons de passer au code machine, mais **CALL** a un grand avantage : cette instruction permet d'avoir facilement accès à plusieurs sous-programmes en code machine, en utilisant **CALL** plusieurs fois avec différentes adresses.

! (SHRIEK)

Cet opérateur permet de définir et d'utiliser de nouvelles instructions BASIC (écrites en code machine). Il faut d'abord utiliser une instruction **DOKE** pour placer l'adresse du programme en code machine aux emplacements $2F5 et $2F6. On exécute alors la commande en entrant ! suivi des paramètres requis par le programme en

Numérisation http://wsc.n3.net
William Saint-Cricq / wsaintcr@wanadoo.fr
Page 129 sur 163
15/08/2000

code machine. En rencontrant le ! , le système stockera ces paramètres dans le buffer d'entrée. Pour accéder aux paramètres, il est nécessaire d'accéder au buffer d'entrée, qui occupe les emplacements 35 hexa à 84 hexa. Les opérations impliquées dans cette procédure comportent un contrôle complet de la syntaxe et la validation des paramètres ; elles peuvent donc être assez complexes. Pour simplifier les tâches, on peut avoir recours à un programme-système situé en Page Zéro E2 hexa, qui range dans le registre A, caractère par caractère, le contenu du buffer d'entrée. L'instruction **JSR $ØØE2** donne accès à ce programme.

Une troisième méthode consiste à utiliser l'adresse du buffer d'entrée aux emplacements $E9 et $EA, indexée, par exemple, par le registre Y, pour obtenir tour à tour chaque caractère, en incrémentant Y à chaque tour au moyen d'une instruction **LDA** indirect-Y.

Si l'on a besoin de plusieurs instructions ! , il faut penser à faire précéder chacune d'elle de **DOKE** # 2F5, adresse du programme.

& (AMPERSAND; "et" commercial)

Le "et" commercial (&) permet de définir des fonctions supplémentaires utilisables dans un programme BASIC. La définition de la fonction doit être rédigée en code machine, et l'adresse de départ du sous-programme de définition doit être placée par **DOKE** à l'emplacement # 2FC avant tout appel de la fonction. Le & doit être suivi d'un argument entre parenthèses, dont la valeur est placée dans l'accumulateur en virgule flottante, afin de pouvoir être utilisée par le sous-programme. L'exemple suivant crée une fonction dont le but est de restituer la ligne où se trouve le curseur ("current cursor row") :

```
10 REM..DEFINIT & POUR DONNER
20 '    LE RANG COURANT DU CURSEUR
30 FOR I=0 TO 5
40 READ D%:POKE #400+I,D%
50 DOKE #2FC,#400 'adresse de depart
65 'charge Y avec la valeur de CURROW
70 DATA #AC,#68,#02
75 'entre le P9 Basic pour placer un
   octet dans l'accumulateur F.P
80 DATA #4C,#B6,#D4
```

Ce programme donne accès à la variable système correspondant à la ligne du curseur (**CURROW**), l'accumulateur en virgule flottante contenant la valeur restituée. &(Ø) restituera cette valeur à l'intérieur d'un programme BASIC. L'accumulateur en virgule flottante occupe six octects, de # DØ à # D5. Les nombres y sont rangés de la

façon suivante : le premier octet contient la valeur de l'exposant plus 128. Quatre octets contiennent la mantisse, l'octet de poids fort étant # D1, et le bit de poids fort représentant 2↑ – 1, tandis que le bit de poids faible de # D4 représente 2↑ – 32. # D5 est un octet de signe : Ø indique un nombre positif, # FF un nombre négatif. La valeur se présente donc sous la forme mantisse*2↑ exposant. La représentation de "zéro" est unique, l'exposant étant mis à zéro.

Le programme suivant corrigera le problème posé par certains lecteurs de cassettes, auquel on a fait allusion au chapitre 6. Le programme charge les emplacements # 281 à # 2BF en évitant les emplacements # 2A9 à # 2BØ. Quand il est chargé par cassette, il copie un autre programme aux emplacements # 221 à # 22A, et modifie le vecteur d'interruption rapide (en # 245 et # 246) de façon à le pointer sur ce programme. Ainsi, chaque fois qu'une interruption se produit, le drapeau d'erreur en # 2B1 est mis à zéro. Sauf si la recherche automatique ("auto-search") est inhibée, le fichier qui suit sur la cassette est alors chargé.

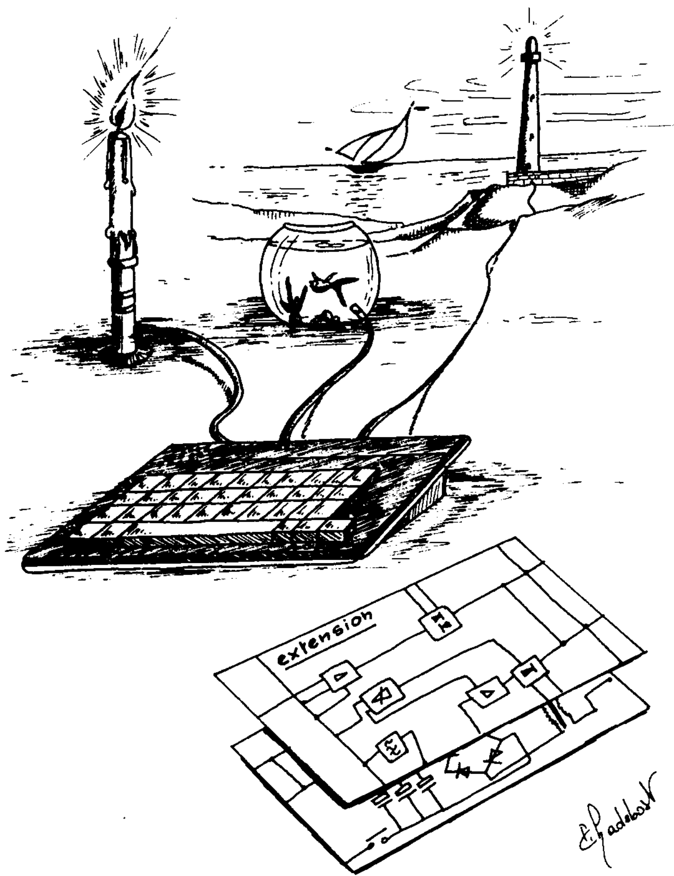
```
10 'Rectifie les erreurs de chargement
20 FOR D=0 TO 39
30 READ N%:POKE#28+D,N%
40 NEXT
50 FOR D=0 TO 13
60 READ N%:POKE#2B1+D,N%
70 NEXT
80 PRINT"METTEZ LE MAGNETOPHONE EN ENR
EGISTR.","PUIS APPUYEZ SUR UNE TOUCHE"
85 PRINT"LE NOM DU PROGRAMME EST '*'":
GET A$
90 CSAVE"*",A#281,E#2BF,AUTO 'ecrire
  de Plus ,S Pour le mode lent
100 DATA #08,#78,#AD,#F9,#FF
110 DATA #C9,#01,#D0,#28
120 DATA #AD,#B6,#E4,#C9,#A2
130 DATA #D0,#15,#A0,#09
140 DATA #B9,#B5,#02,#99,#21,#02
150 DATA #88,#10,#F7,#A9,#21
160 DATA #8D,#45,#02,#A9,#02
170 DATA #8D,#46,#02,#4C,#67,#E8
180 '
190 DATA #00,#4C,#B6,#E7,#48
200 DATA #A9,#00,#8D,#B1,#02,#68
210 DATA #4C,#22,#EE
220 'Inhiber l'auto-recherche du fichi
er suivant Par
230 'POKE #2B2,#60 et POKE #2A6,#60
```


Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Numérisation http://wsc.n3.net



Chapitre 11
Les entrées/sorties

L'ORIC dispose de possibilités d'entrée/sortie multiples et adaptables, qui permettent à un amateur passionné doté d'un minimum d'expérience de l'électronique de diriger et de contrôler des activités périphériques moyennant des techniques de raccordement relativement simples.

Le plan d'implantation de la mémoire fourni à l'Annexe 5 donne les informations nécessaires pour effectuer ces raccordements. Deux zones de la mémoire sont utilisables :
1. Le haut de la page 3, en descendant à partir de l'emplacement #Ø3FF.
2. Le bas de la page 3, entre #Ø3ØØ et #Ø3ØF, où se trouvent les 16 registres, ports de sortie, etc., du microprocesseur 5 6522 VIA de l'ORIC (Versatile Interface Adaptor : Adapteur d'Interfaces Polyvalent). 8 des 16 broches fournissent les huit bits de l'interface imprimante et sont également multiplexées au microprocesseur son. Sur les huit restantes, certaines sont raccordées à l'interface clavier. Ce microprocesseur est responsable de très nombreuses activités et doit donc être utilisé avec précaution.

On peut réaliser une configuration d'interface au niveau de la zone indiquée ci-dessus en 1. en utilisant le bus d'extension, qui donne accès aux broches de commande adéquates. Pour les emplacements situés dans la deuxième zone ci-dessus mentionnée, on peut utilier le port imprimante pour procéder aux raccordements nécessaires.

Tout emplacement mémoire situé dans la zone réservée comprise entre #BFEØ et #BFFF peut éventuellement servir d'entrée ou de sortie moyennant l'instruction **POKE** ou **PEEK** appropriée. Les périphériques raccordés à cet emplacement seront alors considérés comme une adresse mémoire sur laquelle l'écriture et la lecture sont possibles.

Il s'agit d'une zone de mémoire vive que l'on invalide en mettant la broche **MAP** en position basse **(MAP)**.

Soulignons que **MAP** invalidera la mémoire morte (**ROM** interne) si l'on y accède au même moment. Les unités de disquettes ORIC MICRODISC exploitent cette caractéristique, et l'on risque donc, si l'on essaie d'utiliser cette zone de mémoire, d'entrer en contradiction avec un périphérique de l'ORIC ; de plus, l'utilisation de cette zone pour les entrées/sorties est complexe. Elle ne convient donc pas à notre propos actuel. Nous nous sommes limités à la zone comprise entre #BFEØ et #BFFF, mais n'importe quelle zone de mémoire vive, à l'exception de #Ø3ØØ à #Ø3ØF, pourrait être invalidée au moyen de **MAP** et remplacée par une mémoire externe

William Saint-Cricq / wsaintcr@wanadoo.fr Page 131 sur 163 15/08/2000

ou tout autre dispositif pourvu d'une interface convenable.

Le haut de la page 3 de mémoire : #Ø3FF et plus bas

C'est certainement la zone où les raccordements d'interface sont les plus faciles. Elle utilise la broche I/O (E/S) comme signal bas de sortie ; si l'on utilise le haut de la page 3 (emplacements situés au-dessus de #Ø3ØF) elle utilise également le signal I/O CONTROL qui doit, comme dans le cas précédent, être mis en position basse (I/O CONTROL), pour invalider le microprocesseur interne 6522 VIA.

La broche I/O se met automatiquement en position basse chaque fois qu'on accède à une adresse comprise entre #Ø3ØØ et #Ø3FF au moyen d'une instruction **PEEK** ou **POKE**, mais les adresses #Ø3ØØ à #Ø3ØF sont utilisées par le 6522 VIA interne pour la transmission de données vers l'imprimante et le microprocesseur son ; si l'on invalide le 6522, au moment où on l'utilise sur cette gamme d'adresses au moyen de la broche d'entrée I/O CONTROL, l'accès à ces emplacements mémoire n'a plus de raison d'être. Si l'on emploie I/O CONTROL pour des emplacements compris entre #Ø31Ø et #Ø3FF, on doit donc avoir pour but de se raccorder à des dispositifs externes. Pour les E/S utilisateur, il faut utiliser ces adresses à partir de #Ø3FF en allant vers les adresses inférieures, puisque les périphériques ORIC sont conçus pour utiliser des adresses allant de #Ø3ØØ vers les adresses supérieures. Ainsi, tout risque de conflit est écarté le plus longtemps possible !

La broche de lecture-écriture **R/W (READ/WRITE)** reste disponible pour le passage de porte/décodage ; elle est en position basse pour **POKE** (écriture) et en position haute pour **PEEK** (lecture). On trouvera à l'Annexe 1Ø des exemples typiques de circuits.

Le bas de la page 3 de mémoire : de Ø3ØØ à Ø3ØF

A plus d'un titre, cette zone (interface imprimante) est la plus délicate à utiliser, mais c'est aussi la plus passionnante. Ce sont ces emplacements mémoire qui régissent les fonctions multiples et variés du 6522 VIA interne de l'ORIC.

Les emplacements mémoire compris entre #Ø3ØØ et #Ø3ØF constituent les 16 registres de commande du VIA. Il n'est pas nécessaire (heureusement !) de tous les énumérer, mais les lecteurs intéressés peuvent se reporter aux spécifications techniques fournies par les fabricants ou les distributeurs.

Il suffit de connaître les fonctions des 4 premières adresses et de la dernière : #Ø3ØØ, #3Ø1, #3Ø2, #3Ø3 et #Ø3Ø. Le microprocesseur 6522 dispose de deux jeux de 8 lignes de données, appelés port A et port B. Toute opération de lecture ou d'écriture portant sur ces

lignes de données passe par #Ø3ØØ pour le port B et par #Ø3Ø1 pour le port A, soit respectivement **ORB** et **ORA.** Mais on doit préciser initialement s'il s'agit d'une opération de lecture (où les lignes de données servent d'entrée) ou d'écriture (où elles servent de sortie). Pour ce faire, on utilise l'adresse #Ø3Ø2 pour le port B et l'adresse #Ø3Ø3 pour le port A, soit respectivement DDRB et DDRA. DDR est l'abréviation de Data Direction Register (registre de direction des données), la "direction" étant, soit l'entrée, soit la sortie. Les 7 lignes (BØ, B1, B2, B3, B5, B6, B7) du port B réalisent une configuration interne, tandis que B4 constitue la broche **STROBE** de l'interface imprimante.

L'ensemble des 8 lignes du port A (AØ, A1, A2, A3, A4, A5, A6, A7) sont tirées à l'extérieur et constituent les broches de données de l'interface imprimante (broches 3, 5, 7, 9, 11, 13, 15, 17).

Avant d'utiliser en sortie le port imprimante, il faut noter les éléments suivants. Généralement, le port A du 6522 est un port de sortie "verrouillant", c'est-à-dire qu'une fois arrivées au port A, les données s'arrêtent là. mais le 6522 de l'ORIC remplit de nombreuses fonctions, et d'autres transferts de données utilisent le port A, puisque ce port est multiplexé au microprocesseur son, qui est, quant à lui, raccordé à l'interface clavier. Il ne suffit donc pas d'envoyer un octet de données au port A avec **POKE** pour "verrouiller" cette donnée ; le verrouillage doit se faire de l'extérieur. La broche **STROBE** située sur la prise imprimante est en fait la ligne B4 du port B, que le VIA utilise pour dire à l'imprimante que les données sont prêtes. Cette broche transmet une impulsion négative (elle est normalement en position haute) et on peut l'utiliser comme impulsion de verrouillage pour basculer les données en direction d'un périphérique au moment approprié. On peut aussi utiliser la broche I/O du bus d'extension, qui se met en position basse lorsque le VIA est mis en œuvre, puisque le VIA est implanté en page 3 de la mémoire. Cette broche peut également fournir une impulsion de verrouillage, opérant, comme B4, un passage de position haute en position basse. On trouvera à l'Annexe 1Ø différents "verrous" utilisables.

Pour utiliser en sortie le port A du 6522 interne de l'ORIC, on peut employer un programme du type suivant.

On commence par l'instruction **LPRINT. LPRINT CHR$(i)** produira la séquence binaire correspondant à la valeur de i. Si i = l, la suite de bits sera :

```
Ø Ø Ø Ø Ø Ø Ø  1
D7 . . . . . . DØ
```

et la broche 3 de la prise de l'interface imprimante passera en position haute (5 V)

Numérisation http://wsc.n3.net
15/08/2000 Page 132 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

Si i = 4, la suite de bits sera :

Ø Ø Ø Ø Ø 1 Ø Ø
D7 . . . . . DØ

et la broche 7 de la prise de l'imprimante passera en position haute, et ainsi de suite.

Cette instruction correspond en fait à un sous-programme en code machine qui sort des données et émet un signal **STROBE** au moment approprié, quand les données sont acheminées sur les lignes.

Par contre, si l'on utilise l'instruction **POKE**, la donnée doit être placée au moyen de **POKE** à l'adresse #Ø3Ø1, comme on l'a dit ci-dessus, après quoi elle est transmise au port A à partir de cet emplacement. On aura donc #**POKE** #Ø3Ø1,i ,i étant compris entre Ø et 255.

L'impulsion de verrouillage pose un problème. Les lignes BØ à B7 sont commandées par **ORB**, c'est-à-dire #Ø3ØØ. B4 (impulsion **STROBE**) est normalement en position haute, donc, en sortant un Ø par B4, on obtiendrait le signal **STROBE** :

POKE #Ø3ØØ,i

où i = x x x Ø x x x x
D 7 . . . . D Ø

Par exemple, si i a pour valeur 175, B4 sera égal à Ø. On a alors la séquence :

POKE #Ø3Ø1 i (donnée émise en sortie)
POKE #Ø3ØØ, 175 (bascule)

Il semble cependant nécessaire de faire intervenir un laps de temps entre le **STROBE** et la bascule de sortie pour assurer un bon fonctionnement des opérations. Il existe une solution de rechange à la séquence ci-dessus : l'instruction **POKE** #Ø3Ø1 donne accès à la page 3, dès lors I/O passera en position basse au moment voulu, et peut donc servir d'impulsion de verrouillage.

S'agissant d'une entrée au niveau du port A de l'imprimante, il faut d'abord dire au 6522 que le port A doit maintenant recevoir et non transmettre. Ici, le verrouillage ne pose pas de problème, mais là encore, on peut utiliser soit la broche I/O soit la broche R/W pour mettre en service un microprocesseur buffer (à trois états) au cours d'une opération de lecture. Le port A du 6522 est programmé pour fonctionner en sortie à la mise en marche, et il faut donc le programmer pour le faire servir aux entrées. Pour ce faire, on peut utiliser **POKE** pour placer Ø à l'emplacement #Ø3Ø3, puisque c'est à cet emplacement qu'est implanté le registre de direction des données du port A (DDRA), comme on l'a vu plus haut. On a donc :

POKE #Ø3Ø3,Ø

qui met en position d'entrée les 8 bits du port A :

DDRA

Ø Ø Ø Ø Ø Ø Ø Ø
D 7 . . . . D Ø

alors que :

POKE #Ø3Ø3,3

met en position de sortie les bits DØ et D1 du port A, et en position d'entrée les bits D2 à D7 :

DDRA 3 : **D 7 . . . . D Ø**
Ø Ø Ø Ø Ø Ø 1 1
broches entrée / broches sortie

On a donc la série d'instructions suivantes :

```
1Ø  POKE #Ø3Ø3,Ø
15  REM met en position d'entrée les bits du port A
2Ø  PRINT PEEK (#Ø3Ø1)
25  REM lit le port A
3Ø  POKE #Ø3Ø,255.
35  REM ramène le 6522 comme périphérique de sortie
```

La ligne 3Ø est indispensable ; si les lignes 1Ø et 2Ø ne sont pas suivies de la ligne 3Ø, l'utilisateur constatera que le clavier est hors service.

Initialement, avant toute entrée de donnée, on obtiendra par **PRINT PEEK(**#Ø3Ø1**)** la valeur 255, puisque toutes les lignes de données du 6522 sont mises en position haute par des résistances "pull-up".

On trouvera aux Annexes 10 et 11 des informations sur les techniques d'adressage et de décodage des données et des exemples de raccordements de circuits d'interface.

Numérisation http://wsc.n3.net
William Saint-Cricq / wsaintcr@wanadoo.fr Page 133 sur 163 15/08/2000

ANNEXE 1

Table des codes ASCII

Code	Caractère	Codes CTRL
Ø	Nul	
1	Copie	CTRL-A
2		
3	Break	CTRL-C
4	Impression en double ligne	CTRL-D
5		
6	Déclic de touche	CTRL-F
7	Clochette (Ping)	CTRL-G
8	Curseur vers la gauche	CTRL-H
9	Curseur vers la droite	CTRL-I
1Ø	Curseur vers le bas	CTRL-J
11	Curseur vers le haut	CTRL-K
12	Effaçage écran	CTRL-L
13	**RETURN**	CTRL-M
14	Effaçage ligne	CTRL-N
15	Invalidation écran	CTRL-O
16		
17	Curseur	CTRL-Q
18		
19	Écran	CTRL-S
2Ø	Capitales (majuscules)	CTRL-T
21		
22		
23		
24	Annulation ligne	CTRL-X
25		
26		
27	**ESC** (changement de code)	
28		
29		
30		
31		
32	Espacement	

On obtient le deuxième jeu de caractères en **LORES 1.** Le mode **LORESØ** utilise le jeu standard.

CODE	CARACTÈRE STANDARD	CARACTÈRE ALTERNÉ
33	!	33
34	”	34
35	#	35
36	$	36
37	%	37
38	&	38
39	’	39
40	(	4Ø
41	)	41
42	*	42
43	+	43
44	,	44
45	–	45
46	.	46
47	/	47
48	Ø	48
49	1	49
50	2	5Ø
51	3	51
52	4	52
53	5	53
54	6	54
55	7	55
56	8	56
57	9	57
58	:	58
59	;	59
60	<	6Ø
61	=	61
62	>	62
63	?	63
64	@	64
65	A	65
66	B	66
67	C	67
68	D	68
69	E	69
70	F	7Ø
71	G	71
72	H	72
73	I	73
74	J	74
75	K	75
76	L	76
77	M	77
78	N	78
79	O	79
80	P	8Ø
81	Q	81
82	R	82
83	S	83
84	T	84
85	U	85
86	V	86
87	W	87
88	X	88
89	Y	89
90	Z	9Ø
91	[	91
92	\	92
93	]	93
94	↑	94
95	£	95
96	©	96
97	a	97
98	b	98
99	c	99
100	d	1ØØ
101	e	1Ø1
102	f	1Ø2
103	g	1Ø3
104	h	1Ø4
105	i	1Ø5
106	j	1Ø6
107	k	1Ø7
108	l	1Ø8

Numérisation http://wsc.n3.net
15/08/2000
Page 134 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

CODE	CARACTÈRE STANDARD	CARACTÈRE ALTERNÉ
1Ø9	m	1Ø9 ▮
11Ø	n	11Ø ▮
111	o	111 ▮
112	p	
113	q	
114	r	
115	s	
116	t	
117	u	
118	v	
119	w	
12Ø	x	
121	y	
122	z	
123	{	
124	|	
125	}	
126		
127	DEL	127 DEL

Lorsque vous émettez une instruction **PRINT** accompagnée d'un caractère dont le code ASCII est supérieur à 128,**PRINT** soustrait du caractère le bit le plus fort (128 compris) et affiche directement à l'écran le code restant. Les codes ainsi "amputés" ne sont pas considérés comme des bascules de commande, mais entrés directement en tant qu'attributs. Pour afficher ces codes, on doit utiliser **CHR$(i)**, ce qu'on ne peut faire que sur les écrans à basse résolution. Si l'on essaie de les utiliser sur l'écran **HIRES**, on obtiendra un message d'erreur **ILLEGAL QUANTITY.** Par exemple :

1ØØ PRINT CRH$(132);CHR$(145); "ATTRIBUT"

affichera la chaîne "**ATTRIBUT**" en lettres bleues sur fond rouge. Attention : les codes 138, 139, 142, 149, qui génèrent des caractères en hauteur double, supposent deux lignes de programme identiques pour produire l'effet désiré.

CODE	EFFET
128	premier plan noir (texte/graphiques)
129	premier plan rouge (texte/graphiques)
13Ø	premier plan vert (texte/graphiques)
131	premier plan jaune (texte/graphiques)
132	premier plan bleu (texte/graphiques)
133	premier plan magenta (texte/graphiques)
134	premier plan cyan (texte/graphiques)
135	premier plan blanc (texte/graphiques)
136	premier plan noir (texte/graphiques)
137	caractères graphiques
138	caractères en hauteur double (texte)
139	caractères en hauteur double (graphiques)
140	caractères clignotants (texte)
141	caractères clignotants (graphiques)
142	caractères clignotants en hauteur double (texte)
143	caractères clignotants en hauteur double (graphique)
144	fond noir
145	fond rouge
146	fond vert
147	fond jaune
148	fond bleu
149	fond magenta
15Ø	fond cyan
151	fond blanc

William Saint-Cricq / wsaintcr@wanadoo.fr Page 135 sur 163 15/08/2000

Numérisation http://wsc.n3.net

ANNEXE 2
Code "Escape" (changement de code)

Les codes "**ESCAPE**" suivants sont disponibles sur l'Oric. Ils insèrent des attributs à une position-caractère donnée de la zone mémoire d'affichage à l'écran. On peut les entrer directement à l'écran au moyen de la touche **ESC** suivie du caractère, ou les placer dans un programme avec l'instruction **PRINT CHR$(27)** suivie du caractère présenté sous forme de chaîne (soit entre guillemets, soit introduit par **CHR$).**

ESCAPE @	0	Encre noire
ESCAPE A	1	Encre rouge
ESCAPE B	2	Encre verte
ESCAPE C	3	Encre jaune
ESCAPE D	4	Encre bleue
ESCAPE E	5	Encre magenta
ESCAPE F	6	Encre cyan
ESCAPE G	7	Encre blanche
ESCAPE H	8	Texte standard
ESCAPE I	9	Texte "semi-graphique" ("alterné")
ESCAPE J	10	Hauteur double en standard
ESCAPE K	11	Hauteur double en "graphique"
ESCAPE L	12	Clignotement en standard
ESCAPE M	13	Clignotement en semi-graphique
ESCAPE N	14	Clignotement hauteur double standard
ESCAPE O	15	Clignotement hauteur double alterné
ESCAPE P	16	Papier noir
ESCAPE Q	17	Papier rouge
ESCAPE R	18	Papier vert
ESCAPE S	19	Papier jaune
ESCAPE T	20	Papier bleu
ESCAPE U	21	Papier magenta
ESCAPE V	22	Papier cyan
ESCAPE W	23	Papier blanc

Les codes "**ESCAPE**" suivants agissent sur la synchronisation de l'écran :

ESCAPE X	TEXT 60Hz
ESCAPE Y	TEXT 60Hz

ESCAPE Z	TEXT 50Hz
ESCAPE	TEXT 50Hz
ESCAPE }	GRAPHICS 60Hz
ESCAPE	GRAPHICS 60Hz
ESCAPE ~	GRAPHICS 50Hz
ESCAPE —	GRAPHICS 50Hz

Les codes ci-dessus permettent de coordonner l'affichage à l'écran avec la fréquence du courant alternatif qui alimente l'écran vidéo ou le téléviseur servant à l'affichage de l'Oric : de la sorte, les signaux émis par l'Oric assurent les fréquences de trame correctes. La fréquence du secteur est de 50Hz en Grande-Bretagne et en France, et de 60Hz aux États-Unis et en Europe.

William Saint-Cricq / wsaintcr@wanadoo.fr Page 136 sur 163 15/08/2000

Numérisation http://wsc.n3.net

ANNEXE 3
Messages d'erreurs

L'Oric émet des messages d'erreur chaque fois qu'un programme s'interrompt, soit à cause d'erreurs dans la syntaxe ou la structure du programme, soit à cause des limitations de la machine elle-même. L'Oric ne fait jamais d'erreurs, mais il ne peut pas tout faire. Le message d'erreur est suivi du numéro de la ligne de programme à laquelle se trouve l'erreur, s'il survient pendant l'exécution d'un programme. Il est ainsi bien plus facile de détecter les erreurs d'un programme, mais remarquons cependant que la cause de l'erreur peut se trouver antérieurement dans le programme. Par exemple, le message ?ILLEGAL QUANTITY ERROR IN 13Ø indique qu'une expression située à la ligne 13Ø n'a pu être traitée parce qu'une valeur incorrecte avait été affectée à un paramètre. Il se peut que la valeur ait été générée plus tôt dans le programme, mais l'erreur n'interrompt le programme qu'au moment où la valeur erronée est utilisée. Dans ce cas, on peut utiliser **TRON** et **TROFF** pour retracer l'exécution du programme (en y joignant, si besoin est, les instructions **PRINT** nécessaires pour afficher la valeur des variables) ; des instructions **STOP** assureront des pauses dans l'exécution du programme. On trouvera ci-dessous la liste des messages d'erreur de l'Oric, ainsi que leur explication.

?BAD SUBSCRIPT ERROR (Indice erroné) : le programme a cherché à se reporter à un élément de tableau inexistant. Dans le cas de tableau implicite, il s'agit d'un élément non compris entre Ø et 1Ø ; dans le cas d'un tableau dimensionné, l'élément est situé en-dehors des limites déclarées : par exemple, le tableau a été déclaré avec l'instruction DIM A(19,3), et on se réfère à un élément A(2Ø,3).

?BAD UNTIL ERROR (UNTIL erroné) : le programme a rencontré une instructions **UNTIL** sans que l'instruction **REPEAT** correspondante soit mise en mémoire au commencement de la boucle.

?CAN'T CONTINUE ERROR (continuation impossible) : lorsqu'on a interrompu un programme au moyen d'un **CTRL-C** ou d'une instruction **STOP**, on ne peut utiliser **CONT** que si le programme n'a pas été modifié. Si l'on cherche à utiliser **CONT** alors que des modifications sont intervenues, on obtient ce message d'erreur.

?DISP TYPE MISMATCH ERROR (discordance de mode écran) : on a utilisé des instructions applicables dans un seul mode écran avec un autre mode, par exemple **DRAW** ou **CHAR** dans le mode **TEXT** ou **LORES**, ou bien **PLOT** ou **PRINT** dans le mode **HIRES.**

?DIVISION BY ZERO ERROR (division par zéro) : une division par zéro, opération impossible, se trouve impliquée par une expression. Faites attention aux variables non définies, qui prennent une valeur de zéro.

?FORMULA TOO COMPLEX ERROR (formule trop complexe) : le nombre de valeurs intermédiaires que l'Oric doit mémoriser quand il interprète et qu'il évalue une expression peut excéder la capacité de mémoire disponible, et il en résulte cette erreur. Pour rendre l'expression évaluable, décomposez-la en parties plus petites.

?ILLEGAL DIRECT ERROR (commande illégal) : une instruction qui n'est admise qu'à l'intérieur d'une ligne de programme (**INPUT** ou **GET**, par exemple) a été utilisée en mode direct (mode commande).

?ILLEGAL QUANTITY ERROR : un paramètre figurant dans une expression se situe en-dehors des limites correctes. Reportez-vous aux définitions des mots-clés où ces limites sont données ; vérifiez les valeurs obtenues après calcul des expressions utilisées comme paramètres, ainsi que les **INT** utilisés dans le programme ; pensez aux opérations d'arrondissement effectuées automatiquement par l'Oric quand il transforme en nombres entiers les valeurs résultant d'un calcul. Cette erreur apparaît également quand une valeur entière supérieure à 32767 ou inférieure à – 32768 est affectée à une variable entière.

?NEXT WITHOUT FOR ERROR (**NEXT** sans **FOR**) : le programme a rencontré une instruction **NEXT** à laquelle ne correspondait aucune instruction **FOR...TO.** Soit le **FOR...TO** a été omis, soit la séquence de déroulement du programme est entrée dans une boucle.

?OUT OF DATA ERROR (manque de données) : une instruction **READ** renvoie à des **DATA** inexistantes ; il y a donc trop d'instructions **READ** et/ou trop peu de données de **DATA.**

William Saint-Cricq / wsaintcr@wanadoo.fr Page 137 sur 163 15/08/2000

?OUT OF MEMORY ERROR (manque de mémoire) : la mémoire de l'Oric est affectée à différentes fonctions : programme **BASIC**, variables, écran... Si l'ensemble des exigences de ces diverses zones est supérieur à la capacité de mémoire disponible, il en découle cette erreur. Remarquez que l'instruction **HIMEM** peut restreindre la zone qui reste disponible pour le programme **BASIC** et les variables. On peut utiliser **FRE** pour resserrer l'espace dévolu à la mise en mémoire des chaînes, qui se trouve plus grand que ne l'exige le volume des données lorsqu'un programme comporte beaucoup d'opérations effectuées sur des chaînes, puisque ces chaînes se remplacent mutuellement. Si plus de 24 sous-programmes, ou de boucles **REPEAT...UNTIL**, ou plus de 10 boucles **FOR...NEXT** sont imbriqués dans un programme, on dépasse les limites de l'espace affecté dans la pile à la mise en mémoire des numéros de ligne auxquels le programme doit revenir ; de ce fait, une zone particulière de la mémoire se trouve remplie, et on obtient le même message d'erreur.

?OVERFLOW ERROR (dépassement) : si l'Oric génère un nombre trop grand pour qu'il puisse le traiter au cours d'un calcul, il n'y a pas assez de place pour ce nombre dans la représentation en 5 octets utilisée par l'Oric, et cette erreur se produit. La plus grande valeur acceptée par l'Oric est approximativement égale à 1.7E38, et le plus petite à 2.93E-39.

?REDIM'D ARRAY ERROR (re-dimensionnement d'un tableau) : il est impossible de re-dimensionner au cours d'un programme un tableau déjà dimensionné, soit implicitement (11 éléments par dimension), soit au moyen d'une instruction **DIM.**

?REDO FROM START (recommencez au début) : on a entré une donnée non-numérique en réponse à une instruction **INPUT** demandant l'entrée de donnée numérique. Le programme revient à l'instruction **INPUT** pour permettre une nouvelle entrée de donnée.

?RETURN WITHOUT GOSUB ERROR (**RETURN** sans GOSUB) : le programme a rencontré une instruction **RETURN** sans avoir traité auparavant d'instruction **GOSUB** correspondante.

?STRING TOO LONG ERROR (chaîne trop longue) : une chaîne doit comprendre au maximum 255 caractères.

?SYNTAX ERROR (erreur de syntaxe) : la présentation de l'instruction en cours d'interprétation est incorrecte. Cela peut être dû soit à la ponctuation (incorrecte ou omise), soit à des mots-clés mal orthographiés.

?TYPE MISMATCH ERROR (discordance d'appellation) : cette erreur se produit quand on affecte une chaîne à une variable ou à une fonction numérique, et vice-versa.

?UNDEF'D STATEMENT ERROR (instruction non-définie) : le programme a cherché à se brancher sur un numéro de ligne inexistant, en réponse à une instruction **GOTO**, **GOSUB** ou **THEN.**

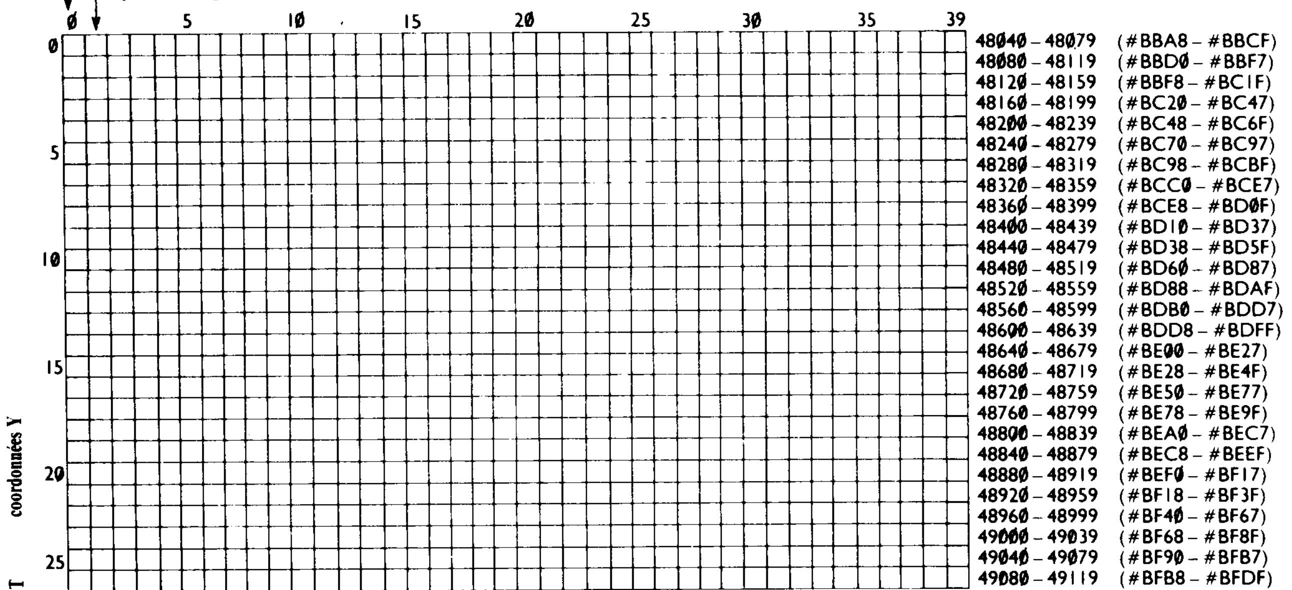
?UNDEF'D FUNCTION ERROR (fonction non-définie) : le programme a rencontré une instruction lui demandant d'évaluer une fonction définie par l'utilisateur alors que cette fonction n'avait pas été spécifiée au préalable par une instruction **DEF FN.**

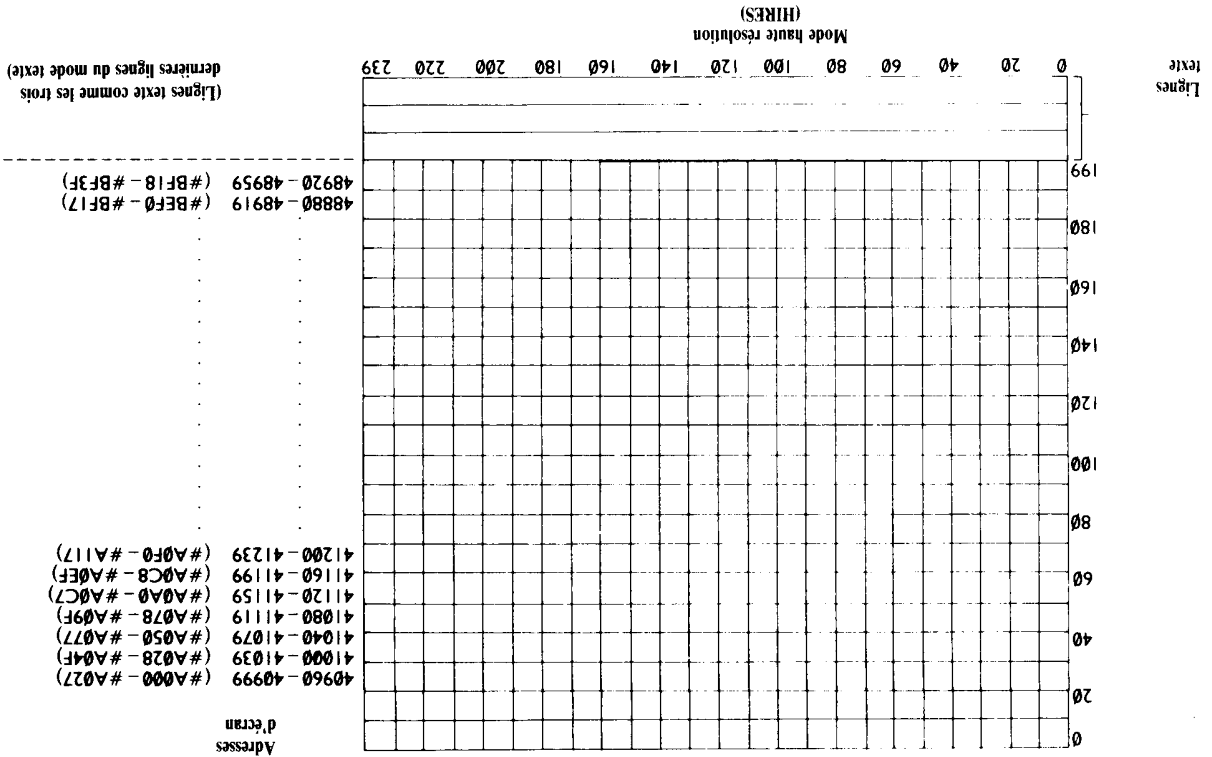
Numérisation http://wsc.n3.net

William Saint-Cricq / wsaintcr@wanadoo.fr Page 138 sur 163 15/08/2000

ANNEXE 4 Grilles d'écran

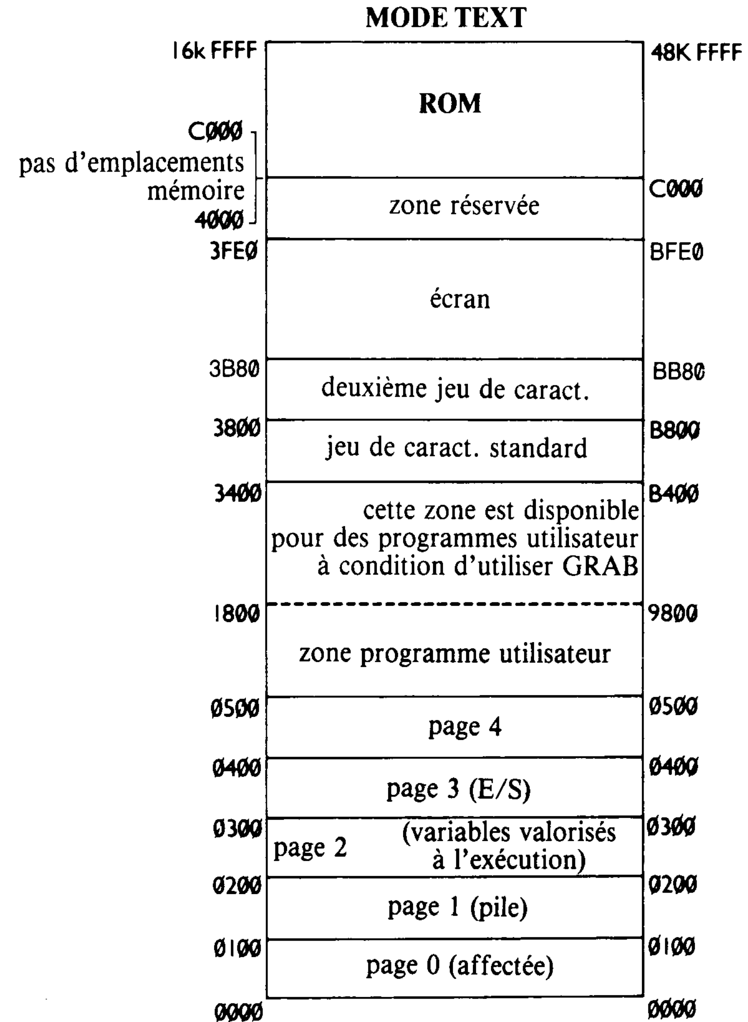
Colonne réservée (pour la couleur du fond), habituellement protégée dans les modes **TEXT** et **LORES.**
En mode **TEXT** cette colonne est normalement réservée à la couleur de premier plan : on peut l'utiliser en mode **LORES**



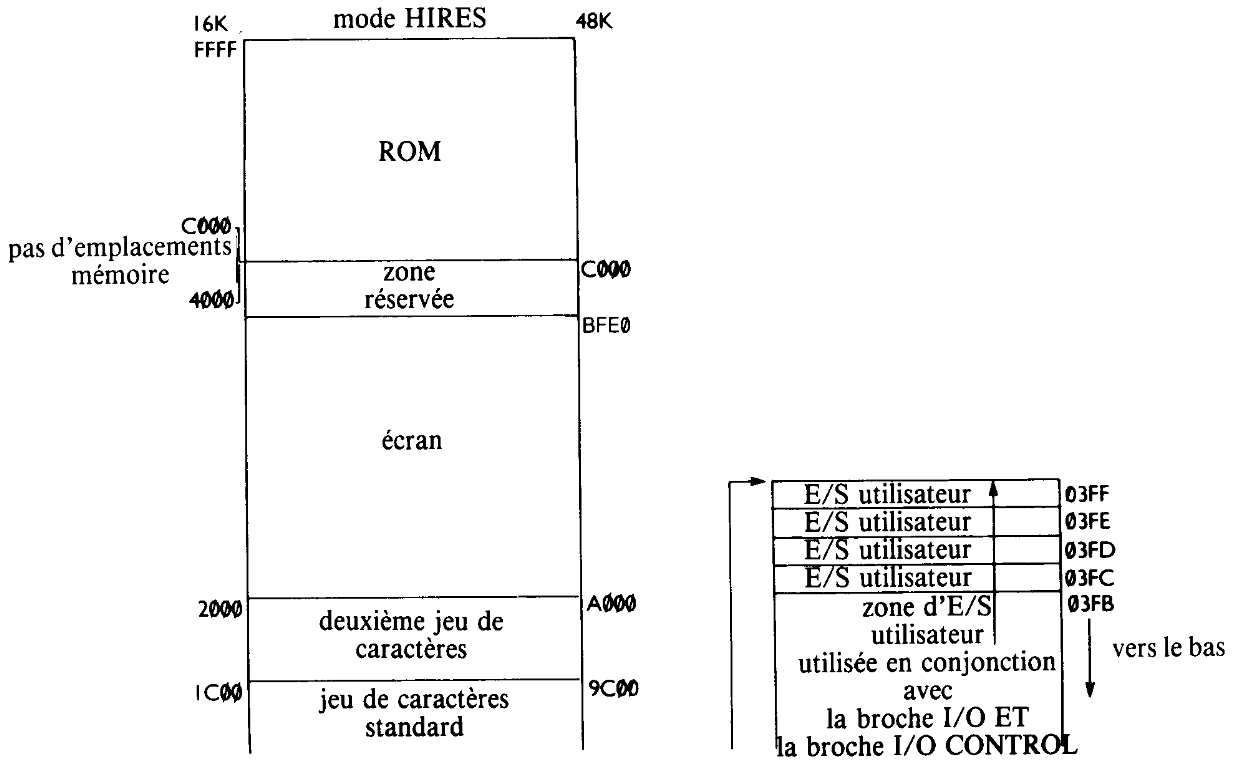


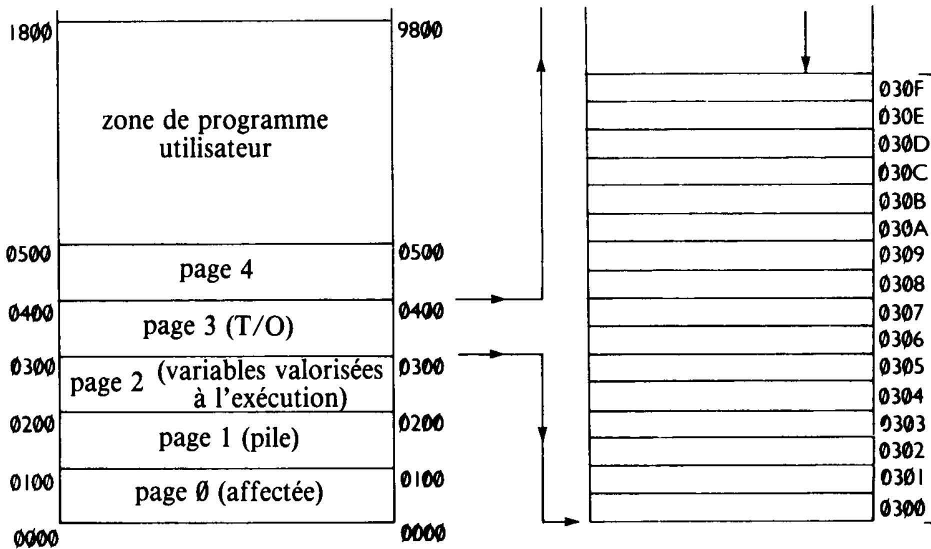
ANNEXE 5
Plan d'implantation de la mémoire

Ces plans d'implantation de la mémoire correspondent aux modes **TEXT** et **HIRES.** Toutes les adresses sont données en hexadécimal. On trouvera sur le plan **HIRES** les emplacements Entrée/Sortie disponibles pour l'utilisateur. Ce sont les mêmes en mode **TEXT.** Les adresses de l'Oric à 16K de **RAM** sont données à gauche, les adresses de l'Oric 48K à droite.



Numérisation http://wsc.n3.net
15/08/2000
Page 140 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr


mode HIRES
16K
48K
FFFF
ROM
C000
pas d'emplacements mémoire
4000
zone réservée
C000
BFE0
écran
2000
deuxième jeu de caractères
A000
1C00
jeu de caractères standard
9C00
E/S utilisateur
03FF
E/S utilisateur
03FE
E/S utilisateur
03FD
E/S utilisateur
03FC
zone d'E/S utilisateur utilisée en conjonction avec la broche I/O ET la broche I/O CONTROL
03FB
vers le bas


1800
9800
zone de programme utilisateur
0500
page 4
0500
0400
page 3 (T/O)
0400
0300
page 2 (variables valorisées à l'exécution)
0300
0200
page 1 (pile)
0200
0100
page Ø (affectée)
0100
0000
0000
030F
030E
030D
030C
030B
030A
0309
0308
0307
0306
0305
0304
0303
0302
0301
0300

Numérisation http://wsc.n3.net

ANNEXE 6
Table de conversion binaire/hexadécimal/décimal

On trouvera d'abord ici une table des équivalences entre les nombres binaires et hexadécimaux pour les valeurs décimales comprises entre Øet 255. La table qui suit permet la conversion hexa/décimal/hexa. Pour convertir un nombre hexa en décimal, il suffit d'une instruction **PRINT**#XXXX sur votre Oric ; on peut de même convertir un nombre décimal en hexa avec **HEX$**.

DEC.	HEXA.	BINAIRE			
			34	#22	00100010
			35	#23	00100011
1	#1	00000001	36	#24	00100100
2	#2	00000010	37	#25	00100101
3	#3	00000011	38	#26	00100110
4	#4	00000100	39	#27	00100111
5	#5	00000101	40	#28	00101000
6	#6	00000110	41	#29	00101001
7	#7	00000111	42	#2A	00101010
8	#8	00001000	43	#2B	00101011
9	#9	00001001	44	#2C	00101100
10	#A	00001010	45	#2D	00101101
11	#B	00001011	46	#2E	00101110
12	#C	00001100	47	#2F	00101111
13	#D	00001101	48	#30	00110000
14	#E	00001110	49	#31	00110001
15	#F	00001111	50	#32	00110010
16	#10	00010000	51	#33	00110011
17	#11	00010001	52	#34	00110100
18	#12	00010010	53	#35	00110101
19	#13	00010011	54	#36	00110110
20	#14	00010100	55	#37	00110111
21	#15	00010101	56	#38	00111000
22	#16	00010110	57	#39	00111001
23	#17	00010111	58	#3A	00111010
24	#18	00011000	59	#3B	00111011
25	#19	00011001	60	#3C	00111100
26	#1A	00011010	61	#3D	00111101
27	#1B	00011011	62	#3E	00111110
28	#1C	00011100	63	#3F	00111111
29	#1D	00011101	64	#40	01000000
30	#1E	00011110	65	#41	01000001
31	#1F	00011111	66	#42	01000010
32	#20	00100000	67	#43	01000011
33	#21	00100001	68	#44	01000100

69	#45	01000101	116	#74	01110100
70	#46	01000110	117	#75	01110101
71	#47	01000111	118	#76	01110110
72	#48	01001000	119	#77	01110111
73	#49	01001001	120	#78	01111000
74	#4A	01001010	121	#79	01111001
75	#4B	01001011	122	#7A	01111010
76	#4C	01001100	123	#7B	01111011
77	#4D	01001101	124	#7C	01111100
78	#4E	01001110	125	#7D	01111101
79	#4F	01001111	126	#7E	01111110
80	#50	01010000	127	#7F	01111111
81	#51	01010001	128	#80	10000000
82	#52	01010010	129	#81	10000001
83	#53	01010011	130	#82	10000010
84	#54	01010100	131	#83	10000011
85	#55	01010101	132	#84	10000100
86	#56	01010110	133	#85	10000101
87	#57	01010111	134	#86	10000110
88	#58	01011000	135	#87	10000111
89	#59	01011001	136	#88	10001000
90	#5A	01011010	137	#89	10001001
91	#5B	01011011	138	#8A	10001010
92	#5C	01011100	139	#8B	10001011
93	#5D	01011101	140	#8C	10001100
94	#5E	01011110	141	#8D	10001101
95	#5F	01011111	142	#8E	10001110
96	#60	01100000	143	#8F	10001111
97	#61	01100001	144	#90	10010000
98	#62	01100010	145	#91	10010001
99	#63	01100011	146	#92	10010010
100	#64	01100100	147	#93	10010011
101	#65	01100101	148	#94	10010100
102	#66	01100110	149	#95	10010101
103	#67	01100111	150	#96	10010110
104	#68	01101000	151	#97	10010111
105	#69	01101001	152	#98	10011000
106	#6A	01101010	153	#99	10011001
107	#6B	01101011	154	#9A	10011010
108	#6C	01101100	155	#9B	10011011
109	#6D	01101101	156	#9C	10011100
110	#6E	01101110	157	#9D	10011101
111	#6F	01101111	158	#9E	10011110
112	#70	01110000	159	#9F	10011111
113	#71	01110001	160	#A0	10100000
114	#72	01110010	161	#A1	10100001
115	#73	01110011	162	#A2	10100010

William Saint-Cricq / wsaintcr@wanadoo.fr Page 142 sur 163 15/08/2000

Numérisation http://wsc.n3.net

163	#A3	10100011
164	#A4	10100100
165	#A5	10100101
166	#A6	10100110
167	#A7	10100111
168	#A8	10101000
169	#A9	10101001
170	#AA	10101010
171	#AB	10101011
172	#AC	10101100
173	#AD	10101101
174	#AE	10101110
175	#AF	10101111
176	#B0	10110000
177	#B1	10110001
178	#B2	10110010
179	#B3	10110011
180	#B4	10110100
181	#B5	10110101
182	#B6	10110110
183	#B7	10110111
184	#B8	10111000
185	#B9	10111001
186	#BA	10111010
187	#BB	10111011
188	#BC	10111100
189	#BD	10111101
190	#BE	10111110
191	#BF	10111111
192	#C0	11000000
193	#C1	11000001
194	#C2	11000010
195	#C3	11000011
196	#C4	11000100
197	#C5	11000101
198	#C6	11000110
199	#C7	11000111
200	#C8	11001000
201	#C9	11001001
202	#CA	11001010
203	#CB	11001011
204	#CC	11001100
205	#CD	11001101
206	#CE	11001110
207	#CF	11001111
208	#D0	11010000
209	#D1	11010001
210	#D2	11010010
211	#D3	11010011
212	#D4	11010100
213	#D5	11010101
214	#D6	11010110
215	#D7	11010111
216	#D8	11011000
217	#D9	11011001
218	#DA	11011010
219	#DB	11011011
220	#DC	11011100
221	#DD	11011101
222	#DE	11011110
223	#DF	11011111
224	#E0	11100000
225	#E1	11100001
226	#E2	11100010
227	#E3	11100011
228	#E4	11100100
229	#E5	11100101
230	#E6	11100110
231	#E7	11100111
232	#E8	11101000
233	#E9	11101001
234	#EA	11101010
235	#EB	11101011
236	#EC	11101100
237	#ED	11101101
238	#EE	11101110
239	#EF	11101111
240	#F0	11110000
241	#F1	11110001
242	#F2	11110010
243	#F3	11110011
244	#F4	11110100
245	#F5	11110101
246	#F6	11110110
247	#F7	11110111
248	#F8	11111000
249	#F9	11111001
250	#FA	11111010
251	#FB	11111011
252	#FC	11111100
253	#FD	11111101
254	#FE	11111110
255	#FF	11111111

Table de conversion décimal/hexa/décimal

3		2		1		Ø	
HEXA	DEC	HEXA	DEC	HEXA	DEC	HEXA	DEC
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
1	4Ø96	1	256	1	16	1	1
2	8192	2	512	2	32	2	2
3	12288	3	768	3	48	3	3
4	16384	4	1Ø24	4	64	4	4
5	2Ø48Ø	5	128Ø	5	8Ø	5	5
6	24576	6	1536	6	96	6	6
7	28672	7	1792	7	112	7	7
8	32768	8	2Ø48	8	128	8	8
9	36864	9	23Ø4	9	144	9	9
A	4Ø96Ø	A	256Ø	A	16Ø	A	1Ø
B	45Ø56	B	2816	B	176	B	11
C	49152	C	3Ø76	C	192	C	12
D	53248	D	3328	D	208	D	13
E	56344	E	3584	E	224	E	14
F	6144Ø	F	384Ø	F	24Ø	F	15

Pour convertir un nombre décimal en hexadécimal à l'aide de cette table, il faut d'abord trouver dans la table le plus grand nombre inférieur au nombre à convertir (4912Ø, par exemple). Dans ce cas, ce sera 45Ø56, à la colonne 3 de la table. Le chiffre hexa correspondant est B : on a ainsi le premier chiffre hexadécimal. On retire ensuite au nombre original la valeur de ce nombre : 4912Ø – 45Ø56 = 4Ø64. On cherche le nombre inférieur à celui-ci qui en est le plus proche : c'est 384Ø, à la colonne 2. On obtient ainsi F, le chiffre hexa suivant. (Si le nombre recherché s'était trouvé à la colonne 1, le chiffre hexa suivant aurait eu la valeur zéro).

On répète l'opération : 4Ø64-384Ø donne un reste de 224, c'est-à-dire E en hexa, à la colonne 1. Il n'y pas de reste : le dernier chiffre est donc Ø. 4912Ø (décimal) = **BFEØ** (hexa).

L'opération inverse (conversion d'un nombre hexadécimal en décimal) est simple. On additionne simplement les valeurs décimales correspondant aux chiffres hexa. #BFEØ, par exemple, donne 45Ø56 + 384Ø + 224 + Ø = 4912Ø.

William Saint-Cricq / wsaintcr@wanadoo.fr Page 143 sur 163 15/08/2000

Numérisation http://wsc.n3.net

ANNEXE 7
Utilisation de l'imprimante ORIC MCP-40

Nous examinerons ici le fonctionnement de l'imprimante/table traçante Oric. Ce dispositif élaboré permet d'obtenir une production graphique de grande qualité, comparable aux sorties imprimées produites par des appareils valant plus de dix mille francs.

L'imprimante se prête à de multiples utilisations, et cela est dû en particulier au porte-plume à 4 couleurs, instrument d'avant-garde dont on peut régler la position avec une position avec une précision de l'ordre de 0,25 mm. L'imprimante, comme nous le verrons plus loin, peut jouer le rôle de table traçante destinée à une production graphique ; mais voyons d'abord son fonctionnement en tant qu'imprimante couleur.

On peut envoyer des caractères à l'imprimante au moyen des instructions **LPRINT** et **LLIST,** dont l'effet est presque identique à celui des instructions **PRINT** et **LIST.** Il existe, cependant, quelques codes de commande qui permettent d'avoir accès aux fonctions et aux couleurs "table traçante". Le tableau ci-dessous résume le rôle de ces quatre codes de commande :

Caractère	Effet
8	Espacement arrière
1Ø	Changement de ligne
11	Changement de ligne inversé
29	Rotation du porte-plume

Notez qu'en utilisant le caractère 29, le programmeur obtient une rotation d'un cran, ce qui ne lui permet pas de choisir telle ou telle couleur. Si vous utilisez cette méthode, vous devrez donc faire attention à la couleur qui se trouve utilisée à un moment donné.

Deux autres codes de commande peuvent servir à diriger le travail de l'imprimante : **CHR$(18)** commande à l'imprimante d'entrer en mode table traçante et **CHR$(17)** la fait revenir au mode texte.

Nous allons maintenant étudier l'utilisation de l'imprimante comme table traçante. Dans ce mode, la gamme de caractères de commande est beaucoup plus étendue. Une fois qu'on est entré en mode table traçante, tous les caractères alphabétiques ci-dessous sont considérés comme le début d'une séquence d'instructions. Les paramètres sont envoyés à l'imprimante par **LPRINT** à la suite du caractère de commande approprié ; il peut s'agir de chaînes alphanumériques ou bien du résultat de l'évaluation de variables ou de calculs. Certaines des instructions sont accompagnées de programmes où ces instructions sont utilisées pour produire des figures, à titre d'exemple.

Caractère	Paramètres	Effet
A		Sortie du mode table traçante
C	n	Passage de la plume au nombre n(Ø à 3). L'ordre des couleurs est généralement le suivant : Noir, Bleu, Vert, Rouge.
D	x,y	Dessin de la position en cours jusqu'à (x,y). On peut inclure plus d'un jeu de coordonnées.
H		Déplacement de la plume jusqu'à l'origine du tracé.
I		Retour de la plume à la position en cours
J	x,y	Dessin par rapport à la position en cours
L	n	Choix du type de ligne dessiné (voir diagramme). Le paramètre n est compris entre Ø et 15.

```
1Ø LPRINT CHR$(18)
2Ø FOR I=Ø TO 15
3Ø LPRINT "I"
4Ø LPRINT "L";I
5Ø LPRINT "P";"TYPE DE LIGNE:";I
6Ø LPRINT"D4ØØ,Ø"
7Ø LPRINT"MØ,-2Ø"
8Ø NEXT I
9Ø LPRINT CHR$(17)
1ØØ END
```

```
LINE TYPE: Ø
LINE TYPE: 1
LINE TYPE: 2
LINE TYPE: 3
LINE TYPE: 4
LINE TYPE: 5
LINE TYPE: 6
LINE TYPE: 7
LINE TYPE: 8
LINE TYPE: 9
LINE TYPE: 1Ø
LINE TYPE: 11
LINE TYPE: 12
LINE TYPE: 13
LINE TYPE: 14
LINE TYPE: 15
```


15/08/2000 Page 144 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

M	x,y	Déplacement de la plume vers x,y.
P	a$	Impression des caractères de a$ jusqu'au retour-chariot suivant ,
Q	n	Choix de l'orientation. La valeur de n (de Ø à 3) permet d'imprimer normalement, de hauteur bas, à l'envers, ou de bas en haut, comme dans le motif ci-dessous.

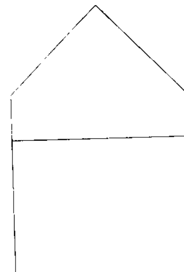
```
10 LPRINT CHR$(18)
20 LPRINT "M240,0"
30 FOR N=1 TO 3
40 LPRINT "Q";N
50 LPRINT "PBONJOUR A TOUS"
60 NEXT
70 LPRINT CHR$(17)
80 END
```

BONJOUR A TOUS
BONJOUR A TOUS
BONJOUR A TOUS
BONJOUR A TOUS

R	x,y	Déplacement de la plume par rapport à la position en cours.
S	n	Dimension des caractères à imprimer n est compris entre Ø et 63. Ø donne 8Ø caractères/ligne, 63 1 caractère/ligne.

```
10 LPRINT CHR$(18)
20 FOR N=0 TO 4
30 LPRINT "S";N
40 LPRINT "PHELLO"
50 NEXT
60 LPRINT"M0,-500"
70 LPRINT"S63"
80 LPRINT"PA"
90 LPRINT"S1"
100 LPRINT CHR$(17)
110 END
```

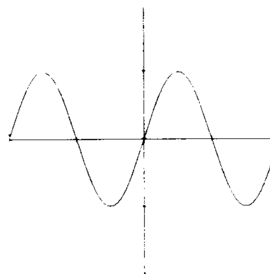

HELLOHELLOHELLOHELLOHELLO



X	a,d,n	Instruction de dessin d'axe. Les paramètres sont :
		a direction de l'axe (Ø pour y, 1 pour x)
		d distance entre les repères
		n nombre de repères le long de l'axe.

L'exemple ci-dessous illustre l'effet de ces paramètres.

```
10 LPRINT CHR$(18);"I"
20 LPRINT "M200,-200"
30 LPRINT "X0,100,4"
40 LPRINT "M0,0"
50 LPRINT "X1,100,4"
60 LPRINT "M0,0"
70 FOR X=0 TO 400
80 A=(X-200)*PI/100
90 Y= SIN(A)*100
100 LPRINT"D";X;",";Y
110 NEXT
```



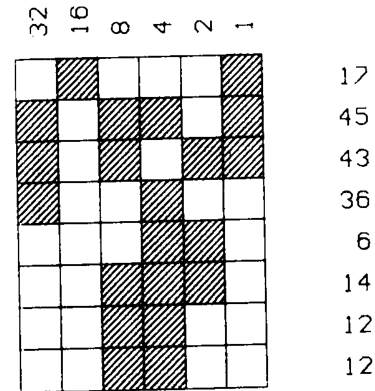
William Saint-Cricq / wsaintcr@wanadoo.fr Page 145 sur 163 15/08/2000

Numérisation http://wsc.n3.net

Nous examinerons maintenant un programme d'application simple permettant de dessiner le tableau des caractères définis par l'utilisateur. Cet exemple donne une idée des applications sérieuses que permettent les capacités de graphiques haute résolution de la MCP-E0 Oric. Vous avez pu remarquer que certaine des figures de ce livre ont été dessinés avec l'imprimante Oric, bien que sa largeur limitée l'empêche de produire des graphiques de grande dimension.

```
10 LPRINT CHR$(18)
20 LPRINT "M0,-480"
30 LPRINT"I"
40 FOR I=0 TO 8
50 LPRINT"M90,";I*30+30
60 LPRINT"D270,";I*30+30:IF I>6 THEN 90
70 LPRINT"M";I*30+90;",30"
80 LPRINT"D";I*30+90;",270"
90 NEXT
100 LPRINT"Q3"
110 FORI=5 TO 0 STEP -1
120 X=270-I*30-5:Y=275
130 LPRINT "M";X",";Y
140 LPRINT"P";2^I
150 NEXT:LPRINT"Q0"
160 FOR I=1 TO 8
170 READ P:Q=P
180 FOR J=5 TO 0 STEP -1
190 PP=INT(2^J+.1)
200 IF P>PP THEN P=P-PP:GOSUB 400
210 NEXT J
220 X=290:Y=270-I*30+5
230 LPRINT "M";X;",";Y
240 LPRINT"P";
250 IF Q<100 THEN LPRINT " ";
260 IF Q<10 THEN LPRINT " ";
270 LPRINT " ";Q
280 NEXT
290 LPRINT CHR$(17)
300 END
400 X=30*(8-J):Y=270-30*I
410 FOR A=0 TO 30 STEP 5
420 LPRINT "M";X",";Y+A
430 LPRINT "D";X+30-A;",";Y+30
440 NEXT
450 FOR A=5 TO 30 STEP 5
460 LPRINT "M";X+A;",";Y
470 LPRINT "D";X+30;",",Y+30-A
480 NEXT
490 RETURN
500 DATA 17,45,43,36,6,14,12,12
```


La figure ci-dessous est un exemple des sorties produites par ce programme, et montre comment il traduit les données décimales qui définissent le caractère en binaire, pour les transcrire ensuite sous forme graphique.



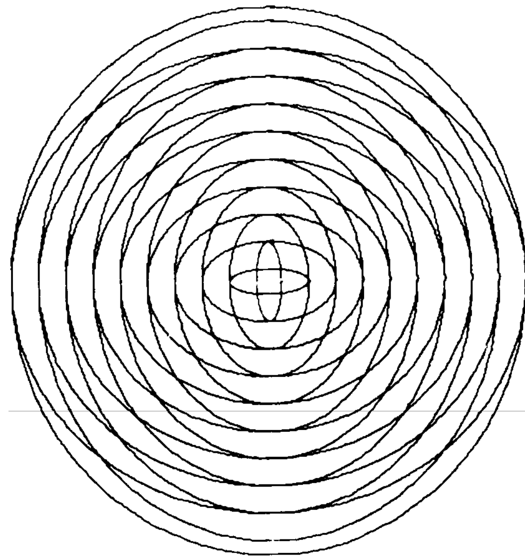
Nous espérons que cette présentation rapide du monde merveilleux de l'imprimante Oric vous aura communiqué notre enthousiasme pour cette petite machine d'une polyvalence remarquable. Nous ajouterons enfin que le manuel de la MCP-40 est très clair, et qu'en vous aidant de ces quelques exemples, vous ne devriez pas avoir de mal à l'utiliser efficacement avec votre Oric. L'imprimante dispose d'une interface Centronics standard, et on peut également, si on le désire, la raccorder à un autre ordinateur. Et maintenant, nous allons laisser le dernier mot à l'imprimante.

```
10 LPRINT CHR$(18);"I"
20 LPRINT "M240,-240"
30 LPRINT"I"
40 LPRINT "M0,0"
50 A=10:B=30
60 REPEAT
70 GOSUB 200
80 B=A:A=A+20
90 GOSUB 200
100 B=B+40
110 UNTIL B=210
120 B=200:GOSUB 200
130 LPRINT "M100,-250"
140 LPRINT"PSALUT!"
150 LPRINT CHR$(17)
160 END
```


William Saint-Cricq / wsaintcr@wanadoo.fr Page 146 sur 163 15/08/2000

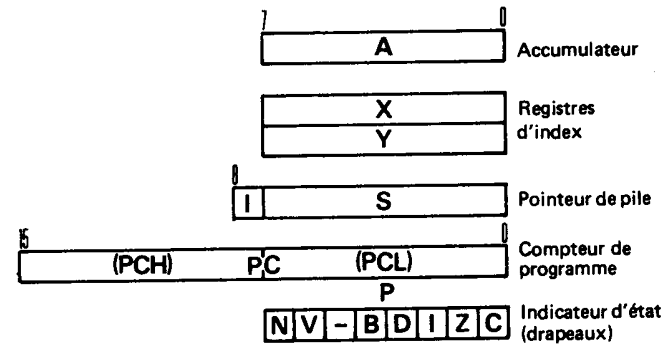
Numérisation http://wsc.n3.net

```
200 LPRINT"M";A;",0"
210 FOR M=0 TO 2*PI STEP PI/100
220 X=A*COS(M):Y=B*SIN(M)
230 LPRINT "D";X;",";Y
240 NEXT
250 LPRINT"D";A;",0"
260 RETURN
```



SALUT

ANNEXE 8
Registre du microprocesseur 65Ø2



N Signe moins (nombre négatif)
V Débordement
—
B Break (interruption)
D Décimal
I Inhibition d'interruption
Z Indicateur de zéro
C Indicateur de retenue

15/08/2000 Page 147 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

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.

William Saint-Cricq / wsaintcr@wanadoo.fr Page 148 sur 163 15/08/2000

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
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
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

Numérisation http://wsc.n3.net
Page 149 sur 163
William Saint-Cricq / wsaintcr@wanadoo.fr
15/08/2000

Numérisation http://wsc.n3.net

Mnémonique	Fonction	Langage d'assemblage	Mode d'adressage	Nombre d'octets	Code HEX	Drapeaux
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	
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

Mnémonique	Fonction	Langage d'assemblage	Mode d'adressage	Nombre d'octets	Code HEX	Drapeaux
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	
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

William Saint-Cricq / wsaintcr@wanadoo.fr Page 150 sur 163 15/08/2000

ANNEXE 9

LES ADRESSES ET LES ROUTINES DE LA ROM

Voici les routines de la ROM de l'ORIC ATMOS qui peuvent être appelées avec **CALL**. Elles sont également utilisables directement depuis une routine écrite en langage machine par l'utilisateur.

Pour la plupart de ces routines il sera suffisant de charger le registre approprié du 65Ø2 avant d'effectuer un "**JSR**" à la routine désirée. Toutefois, pour les routines graphiques ou sonores, des paramètres doivent être transférés. Il faut les placer dans la zone mémoire commençant à l'adresse #2EØ. Les paramètres sont des nombres codés en complément binaire à deux sur 16 bits. Nous désignerons l'adresse des paramètres par **PARAMS**. En fin de sous-routine **PARAMS+Ø** est monté à 1 si une erreur s'est produite. Le **CALL** mettra à Ø **PARAMS+Ø** avant l'exécution de la sous-routine.

Toutes les adresses sont indiquées en hexadécimal. Toutes les routines modifient les registres A, X et Y sauf indication contraire :

VDU — adresse : F77C

Affiche le caractère à l'écran et déplace le curseur à droite.

paramètre appelé : X = caractère à afficher
paramètre renvoyé : aucun
registres affectés : aucun

STOUT — adresse : F865

Affiche un message sur la ligne d'état (adresse 48ØØØ à 48Ø39).

paramètres appelés : A = adresse du message (début)
Y = adresse du message (fin)
X = position horizontale de départ

paramètre renvoyé : X = position prochaine du curseur.

Le message est terminé par un Ø.

GTORKB — adresse : EB78

Les caractères sont renvoyés à la vitesse de répétition déterminée par le contenu des mémoires 24E et 24F.

En 24E se trouve le délai qui précède la répétition de toutes les 30 ms.

En 24F se trouve le délai entre deux affichages. Pour obtenir l'affichage le plus rapide possible fixer 24E et 24F à 1.

paramètres appelés : aucun
paramètre renvoyé : A = code ASCII du caractère
regristres affectés : aucun

PRTCHR — adresse : F5C1

Envoi d'un caractère à l'imprimante.

paramètre appelé : A = code ASCII du caractère

registres affectés : aucun

OUTLED — adresse : E75A

En-tête (9 caractères en code ACSII n°16, SYN) envoyé à la cassette à la vitesse prévue.

paramètres appelés : aucun
paramètres renvoyés : aucun

Noter : pour toutes les routines concernant le magnétophone la vitesse est pilotée par le contenu de l'adresse 24D. Ø pour rapide (2400 bauds). Plus grand que Ø pour lent (300 bauds).

GETSYN — adresse : E735

Lit les octets parvenant du magnétophone jusqu'au moment de la synchronisation.

paramètres appelés : aucun
paramètres renvoyés : aucun
registres affectés : A et X

Numérisation http://wsc.n3.net
15/08/2000 Page 151 sur 163 William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

OUTBYT adresse : E65E

Émet un octet vers le magnétophone à la vitesse choisie.
paramètre appelé : A = caractère à émettre
paramètres renvoyés : aucun
registre affecté : A

RDBYTE adresse : E6C9

Lit un octet parvenant du magnétophone à la vitesse choisie.

paramètres appelés : aucun
paramètre renvoyé : A = caractère reçu
registre affecté : A

CURSET adresse : FØC8

paramètres appelés : **PARAMS + 1** : valeur de X
PARAMS + 3 : valeur de Y
PARAMS + 5 : valeur de FD

paramètres renvoyés : **PARAMS** mis à 1 si les valeurs passées ne conviennent pas.

CURMOV adresse : FØFD

paramètres appelés : **PARAMS + 1** : valeur de X
PARAMS + 3 : valeur de Y
PARAMS + 5 : valeur de FD
paramètres renvoyés : **PARAMS** mis à 1 si les valeurs passées ne conviennent pas.

DRAW adresse : F11Ø

paramètres appelés : **PARAMS + 1** : valeur de X
PARAMS + 3 : valeur de Y
PARAMS + 5 : valeur de FD

paramètres renvoyés : **PARAMS** mis à 1 si les valeurs passées ne conviennent pas.

CHAR adresse : F12D

paramètres appelés : **PARAMS + 1** : code ASCII du caractère
PARAMS + 3 : clavier
Ø standard
1 semi-graphique
PARAMS + 5 : valeur de FD
paramètres renvoyés : **PARAMS** mis à 1 si les valeurs passées ne conviennent pas.

CIRCLE adresse : F37F

paramètres appelés : **PARAMS + 1** : rayon
PARAMS + 3 : valeur de FD
paramètres renvoyés : **PARAMS** mis à 1 si les valeurs passées ne conviennent pas.

PATRN adresse : F11D

Pattern
paramètres appelés : **PARAMS + 1** : de Ø à 255
paramètres renvoyés : **PARAMS** mis à 1 si la valeur passé est négative ou supérieure à 255.
registre affecté : X

POINT adresse : F1C8

paramètres appelés : **PARAMS + 1** : valeur de X
PARAMS + 3 : valeur de Y
paramètres renvoyés : **PARAMS** mis à 1 si les valeurs passées ne conviennent pas à un point de l'écran graphique.
PARAMS + 1 mis à Ø = couleur du fond.
PARAMS + 1 mis à 1 = couleur de l'avant plan.

FILL adresse : F268

paramètres appelés : **PARAMS + 1** : nombre de lignes
PARAMS + 3 : nombre de cellules (de 6 pixels)
PARAMS + 5 : de Ø à 255
paramètres renvoyés : **PARAMS** mis à 1 si les valeurs passées ne conviennent pas.

William Saint-Cricq / wsaintcr@wanadoo.fr Page 152 sur 163 15/08/2000

Numérisation http://wsc.n3.net

PAPER **adresse : E2Ø4**

paramètre appelé : **PARAMS+1** : de Ø à 7 (couleur)
paramètres renvoyés : **PARAMS** mis à 1 si la valeur passée ne convient pas.

INK **adresse : F21Ø**

paramètre appelé : **PARAMS+1** : Ø à 7 (couleur)
paramètres renvoyés : **PARAMS** mis à 1 si la valeur passée ne convient pas.

PING accès direct **adresses FA9F**

SHOOT **FAB5**
EXPLD **FACB**
ZAP **FAE1**

KB BEEP accès direct **adresse : FB14**

Produit le "BIP" du clavier

CONTBP accès direct **adresse : FB2A**

Produit le "BIP" de la touche **CTRL RETURN, ESC.**

SOUND **adresse : FB4Ø**

paramètres appelés : **PARAMS+1** : canal de (1 à 6)
PARAMS+3 : période (de 1 à 65535)
PARAMS+5 : volume (de Ø à 15)
paramètres renvoyés : **PARAMS** mis à 1 si les valeurs passées ne conviennent pas.

MUSIC **adresse : FC18**

paramètres appelés : **PARAMS+1** : canal (de 1 à 3)
PARAMS+3 : octave (de Ø à 7)
PARAMS+5 : note (de Ø à 12)
PARAMS+7 : volume (de Ø à 15)
paramètres renvoyés : **PARAMS** mis à 1 si les valeurs passées ne conviennent pas.

PLAY **adresse : FBDØ**

paramètres appelés : **PARAMS+1** : canaux actifs (Ø à 7) (son)
PARAMS+3 : canaux actifs (Ø à 7) (bruit)
PARAMS+5 : enveloppe (1 à 7)
PARAMS+7 : durée enveloppe (Ø à 65535)
paramètres renvoyés : **PARAMS** mis à 1 si les valeurs passées ne conviennent pas.

W8912 **adresse : F59Ø**

Va lire en A l'adresse du 8912 où sera recopié le contenu de X. La routine vérifie en outre que la lecture du clavier n'est pas inhibée. Le registre OE ne doit pas être utilisé car c'est le port externe utilisé par le clavier.

paramètres appelés : A = numéro du registre de 8912
X = donnée à transférer
paramètres renvoyés : aucun

RAM DE L'ORIC ATMOS (page zéro/page deux)

PAGE Ø

LINWID	#31	longueur de la ligne pour le terminal (longueur de la ligne de l'imprimante en version 1.Ø) (4Ø par défaut).
TXTTAB	#9A-#9B	début du texte Basic
VARTAB	#9C-#9D	début des variables
ARYTAB	#9E-#9F	début des tableaux

William Saint-Cricq / wsaintcr@wanadoo.fr Page 153 sur 163 15/08/2000

Numérisation http://wsc.n3.net

STREND	# AØ- # A1	fin des variables, **LOMEM**
MEMSIZ	# A6- # A7	sommet de la mémoire disponible, **HIMEM**
CHRGET	# E2- # E7	code pour incrémenter **TXTPTR**
CHRGOT	# E8	instruction **LDA**
TXTPTR	# E9- # EA	pointeur indiquant le prochain caractère à interpréter.
SKPSPC	# EB- # EE	envoie à **CHRGET** en cas d'appui sur la barre d'espace.
QNUM	# EF- # F8	retenue si Ø - 9 et indicateur de zéro si **CHR$(Ø).**
CHRRTS	# F9	instruction de retour.

PAGE 2

KEYAD	# 2Ø8	adresse de la dernière touche enfoncée.
KBSTAT	# 2Ø9	# A4 = **SHIFT** de gauche, # A7 = **SHIFT** de droite : prioritaires # A2 = **CTRL** # A5 = **FUNCT**
CAPLCK	# 2ØC	# FF = **CAPS** # 7F = minuscules.
PAT	# 213	registre pour Pattern **(CIRCLE/DRAW)**
CURX	# 219	position horizontale du curseur en **HIRES**
CURY	# 21A	position verticale du curseur en **HIRES**
GRA	# 21F	1 = **HIRES**, Ø = **TEXT** ou **LORES.**
SXTNK	# 22Ø	1 = 16K sinon 48K
XVDU	# 238	saut à la routine **VDU**
XGETKY	# 23B	saut à la routine **GTORKB**
XPRTCH	# 23E	saut à la routine **PRTCHR**
XSTOUT	# 241	saut à la routine **STOUT**
INTFS	# 244	saut pour interruption
NMIJP	# 247	saut à la routine d'interruption non masquable **(NMI)**
INTSL	# 24A	retour après interruption. (normalement **RTI** mais possibilité de l'associer à un saut).
TSPEED	# 24D	vitesse : Ø rapide, différent de Ø : lent,
	# 24E	temporisation pour répétition automatique des touches du clavier.
KBRPT	# 24F	cadence de répétition des touches.
PWIDTH	# 256	nombre de colonnes sur l'imprimante (8Ø par défaut).
VWIDTH	# 257	nombre de colonnes à l'écran (4Ø par défaut).

15/08/2000

CURROW	# 268	position verticale du curseur (n° de la ligne).
MODE Ø	# 26A	les bits de cette octet définissent l'état de diverses fonctions.

BIT	FONCTION
7	libre
6	libre
5	1 : 40 colonnes; Ø : 38 colonnes
4	1 : le dernier caractère envoyé était **ESC** Ø : normal
3	1 : clavier muet Ø : clavier sonore
2	libre
1	1 : vidéo active Ø : pas d'image à l'écran
Ø	1 : curseur visible Ø : curseur invisible

BGND	# 26B	arrière plan (papier) : code couleur + 16
FGND	# 26C	avant plan (encre) : code couleur
CURON	# 27Ø	drapeau de présence du curseur
CURINV	# 271	drapeau de clignotement de curseur
TIMER 1	# 272- # 273	clavier
TIMER 2	# 274- # 275	curseur
TIMER 3	# 276- # 277	libre ou utilisé par **WAIT**. décrémente de 1 tous les 1/100s depuis 65535, se remet à 65535 après avoir atteint Ø. Modifié lorsque **WAIT** est utilisé.
VDUL 2	# 278- # 279	adresse de la deuxième ligne d'écran.
VDUL 1	# 27A- # 27B	adresse de la première ligne d'écran.
VDUCH	# 27C- # 27D	nombre de caractères à faire défiler (scrolling) normalement 26 lignes de 40.
NOROWS	# 27E	nombre de lignes affichées à l'écran.
ICHAR	# 2DF	code de la dernière touche actionnée.
PARAMS	# 2EØ...	mémoires tampon pour transmission de paramètres pour les routines graphiques et sonores.

Page 154 sur 163

William Saint-Cricq / wsaintcr@wanadoo.fr

Numérisation http://wsc.n3.net

ANNEXE 10

CIRCUITS D'ENTRÉES/SORTIES

Le fanatique d'électronique qui désire interfacer son ORIC pour contrôler le monde extérieur, n'a peut-être pas une grande connaissance de l'informatique. Le but recherché dans ces quelques pages est de lui donner les informations qui lui permettront d'utiliser quelques puces de silicium pour réaliser ses applications en s'inspirant des descriptions qui suivent. Il comprendra également comment piloter ces interfaces d'entrées/sorties (en anglais I/O : Input/Output) comme de simples adresses mémoire.

Adressage et décodage des mémoires

La figure 10.1 montre le brochage du microprocesseur 65Ø2. Ce microprocesseur est l'UC, l'unité centrale (en anglais CPO : Central Processing Unit), c'est en quelque sorte le "cerveau" de l'ORIC. Il comporte 8 lignes de données (DØ à D7) et 16 lignes d'adresses (AØ à A15).

Le tableau 10.2 montre qu'il y a 65536 manières d'arranger les 16 lignes d'adresses qui peuvent être chacune à l'état bas (Ø) ou à l'état haut (1). Ces états correspondent en fait à des tensions respectives de O.V. et de +5V.

Le 68Ø9 peut adresser 65536 mémoires différentes ou encore 64K.

Ces lignes d'adresses permettent de travailler soit avec des mémoires **ROM**, soit avec des mémoires **RAM**, soit encore avec d'autres circuits implantés dans l'espace mémoire (consultez la carte mémoire de l'Annexe 5). Les 8 lignes de données sont utilisées pour transférer des instuctions codées sur 8 bits ou des données brutes entre l'UC et les mémoires ou les périphériques.

A l'aide de la figure 10.3. on peut voir qu'il est possible d'arranger les 8 lignes de données selon 256 configurations différentes. Comment peut-on à présent utiliser ces lignes d'adresses afin de sélectionner certains emplacements mémoire ?

Au chapitre 11, nous avons vu que l'on pouvait utiliser soit certaines zones mémoires libres, soit la page 3. La zone libre se trouve entre les adresses #BFEØ et #BFFF.

A l'aide de la table de conversion de l'Annexe 6, on peut faire la traduction en décimal :

#	B	F	F	F
	45Ø56+	384Ø+	24Ø+	15=adresse 49151

Bien entendu, on peut utiliser l'ORIC pour faire ces calculs à l'aide d'une simple instruction **PRINT**#BFFF mais ce que nous importe plus, c'est la structure de 4 bits binaires correspondant à chaque chiffre hexadécimal.

De manière analogue, en consultant à nouveau la carte mémoire, nous apprenons que la page 3 se trouve entre les adresses #300 et #3FF (768 et 1Ø23 en décimal). Comment ces nombres sont-ils utilisés pour connaître l'état des lignes d'adresses ?

Prenons #BFFF par exemple : sur la figure 10.4, nous voyons la configuration binaire associée à chaque chiffre hexadécimal. Ainsi à B correspond 1Ø11 et à F 1111. L'état des lignes d'adresses sera donc :

HEXA	B	F	F	F
BIN	1 Ø11	1111	1111	111 1
ADRESSE	A15........			........AØ

Ainsi, chaque fois que l'on exécute l'instruction
POKE#BFFF, XXX
ou encore
POKE 4912Ø, XXX,
les lignes d'adresses prendront la configuraiton ci-dessus. Elles sont toutes à l'état haut (1) à l'exception de A14 qui est à l'état bas (Ø).

De même, pour donner un exemple en page 3, l'instruction **POKE**#3FF,XXX donnera la configuration suivante :

HEXA	#	Ø	3	F	F
BIN		ØØØØ	ØØ11	1111	111 1
ADRESSE		A15........			........AØ

Dans cet exemple, les lignes 15 à 1Ø sont à l'état bas (Ø).

Les états différents des lignes pour chaque adresse permettent de les décoder de manière à obtenir un signal de validation à l'état bas qui permettra d'activer un circuit déterminé.

Décodage et décodeurs

Il y a de nombreuses manières d'obtenir un signal de validation bas.

Nous allons décrire quelques exemples afin de mettre en lumière les principes généraux.

En général, les entrées et les sorties des circuits intégrés sont validées (ou activées) par un signal bas (Ø). Il est nécessaire que ce signal ne soit appliqué au circuit que lorsque l'adresse correcte est sur les lignes d'adresse.

Ce signal (CE : Chip Enable) peut servir à valider un circuit du type 74L5374. Ce circuit comme le montre la figure 10.6 comporte

William Saint-Cricq / wsaintcr@wanadoo.fr Page 155 sur 163 15/08/2000

Numérisation http://wsc.n3.net

six lignes de sorties "latch". (Un circuit "latch" conserve les données sur ses lignes de sorties tant que celles-ci ne sont pas explicitement modifiées. A ce titre, ce circuit permet de mémoriser un état logique). Ce signal CE permet d'activer ce circuit, ou d'autres comme le 74LS244 ou le 74LS245, pendant une entrée. Cependant cette fois les données entrées sont dirigées vers le bus de données de l'ORIC à condition que le 6522 soit *désactivé* par une mise à l'état bas de la ligne **I/O CONTROL** afin d'éviter un conflit entre les données internes et externes. Le circuit **I/O CONTROL** est activé quand on adresse en page 3 comme c'est expliqué au chapitre 11.

La figure 10.5 montre un décodage possible du haut de la page 3 à l'aide d'un décodeur à 8 entrées du type 74LS3Ø. La ligne IO est mise à l'état bas chaque fois que la page 3 est adressée et ceci permet d'éviter d'avoir à décoder les 8 bits de poids fort de l'adresse.

Toutes les connections sont faites par l'intermédiaire du bus d'expansion IDC à 34 contacts de l'ORIC. Il est possible de se procurer ces connections ainsi que les circuits intégrés chez les revendeurs d'électronique.

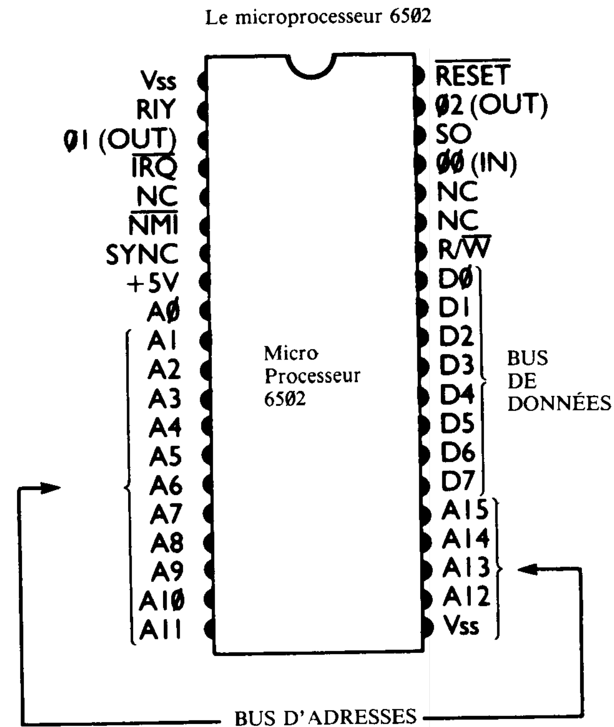
La figure 10.6 montre un circuit utilisant un VIA6522 qui permet d'établir des entrées sorties de manière très souple. C'est d'ailleurs pour cela que l'ORIC en utilise un ! Le VIA comporte 16 registres qui peuvent être repérés dans la mémoire grâce aux lignes d'adresses AØ, A1, A2 et A3 qui peuvent en effet présenter 16 configurations différentes de Ø et de 1. Les registres les plus significatifs sont ORB, ORA, DDRA et DDRB. Si nous les plaçons DDRA à l'adresse #Ø3E3 comme c'est indiqué, l'ordre **POKE** #Ø3E3, 255 mettre le port A en sortie.

POKE #Ø3E1,XX enverra alors la donnée XX vers le port de sortie. De manière analogue, **POKE** #Ø3E3,Ø mettra le port A en entrée. On pourra alors lire l'état du port A par **PEEK(** #Ø3E1).

Le décodage est plus simple car seules 12 lignes d'adresse doivent être décodées. 4 circuits 17154 qui sont des décodeurs 4-16 lignes donneront quatre lignes de sorties qui produiront, une fois réunies par une porte, le signal de validation CE pour le VIA ainsi que les signaux d'invalidation qui dépendent de la zone mémoire utilisée.

Lorsque l'on utilise l'interface de l'imprimante pour les entrées/sorties, il n'y a pas besoin de validations. On n'utilise pas **I/O CONTROL, MAP** et **ROMDIS**. Le **74LS235**semble approprié car il peut être utilisé en entrée ou en sortie sur les mêmes lignes de données. Il n'est cependant possible de tirer de la puissance que sur le connecteur à 34 broches en l'absence d'une alimentation extérieure.

Le microprocesseur 65Ø2



Les états différents des lignes pour chaque adresse permettent de les décoder de manière à obtenir un signal de validation à l'état bas qui permettra d'activer un circuit déterminé.

DØ – D7 : BUS DE DONNÉES
AØ – A15 : BUS D'ADRESSES
***R/W** : (LECTURE/ÉCRITURE)
***IRQ** : Demande d'interruption, activée seulementsi le flog I est à Ø et si IRQ est mis à l'état bas.
***RESET** : Initialisation du 65Ø2

William Saint-Cricq / wsaintcr@wanadoo.fr   Page 156 sur 163   15/08/2000

Numérisation http://wsc.n3.net

BIT	NIVEAU	PUISSANCE	DECIMAL
A_0	1	2^0	1
A_1	1	2^1	2
A_2	1	2^2	4
A_3	1	2^3	8
A_4	1	2^4	16
A_5	1	2^5	32
A_6	1	2^6	64
A_7	1	2^7	128
A_8	1	2^8	256
A_9	1	2^9	512
A_{10}	1	2^{10}	1Ø24
A_{11}	1	2^{11}	2Ø48
A_{12}	1	2^{12}	4Ø96
A_{13}	1	2^{13}	8192
A_{14}	1	2^{14}	16384
A_{15}	1	2^{15}	32768

ADRESSAGES DES MÉMOIRES

TOTAL=65 535+1=65 536 ADRESSES MÉMOIRES

↓ TOUTES LES LIGNES AØ ADRESSE ØØØØ

LIGNES DE DONNÉES

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	BIT
	1	1	1	1	1	1	1	1	NIVEAU BINAIRE
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	PUISSANCE DE 2
255=	128	64	32	16	8	4	2	1	DECIMAL

DÉCIMAL	BINAIRE	HEXA
Ø	ØØØØ	Ø
1	ØØØ1	1
2	ØØ1Ø	2
3	ØØ11	3
4	Ø1ØØ	4
5	Ø1Ø1	5
6	Ø11Ø	6
7	Ø111	7
8	1ØØØ	8
9	1ØØ1	9
1Ø	1Ø1Ø	A
11	1Ø11	B
12	11ØØ	C
13	11Ø1	D
14	111Ø	E
15	1111	F

William Saint-Cricq / wsaintcr@wanadoo.fr Page 157 sur 163 15/08/2000

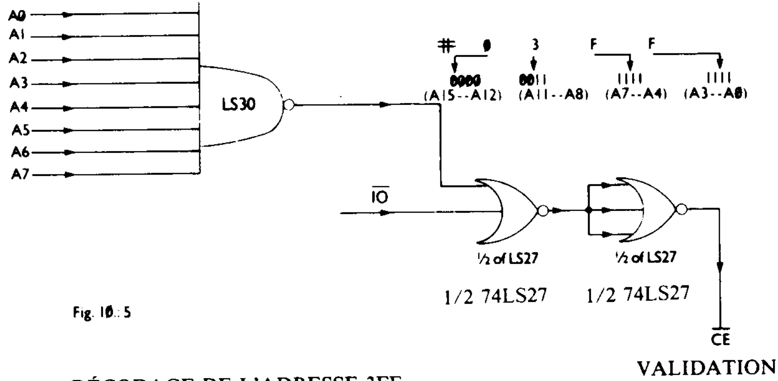
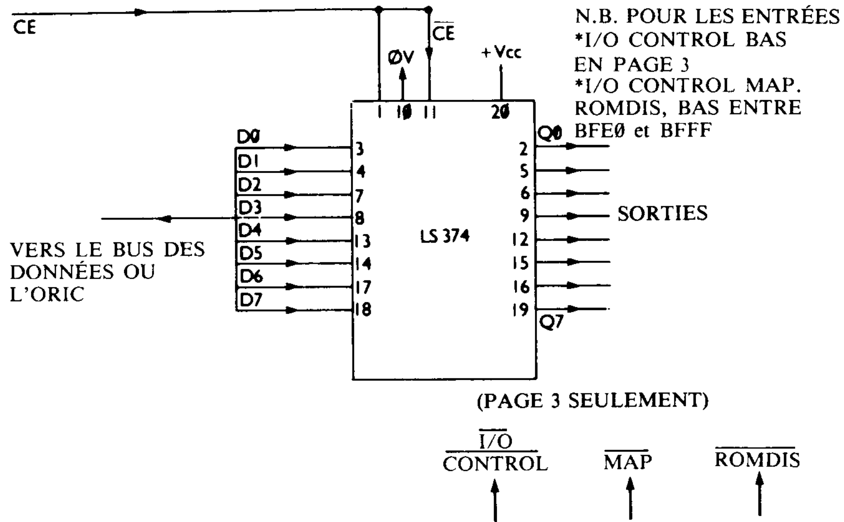
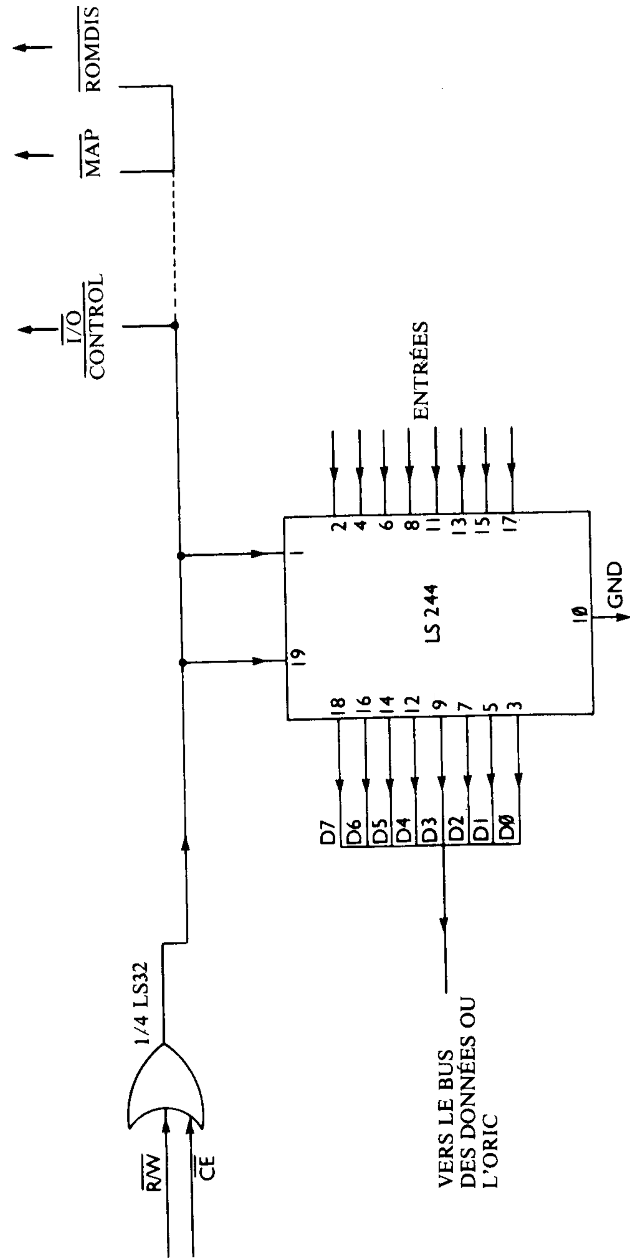


Fig. 10.5

DÉCODAGE DE L'ADRESSE 3FF
(Haut de la page 3)

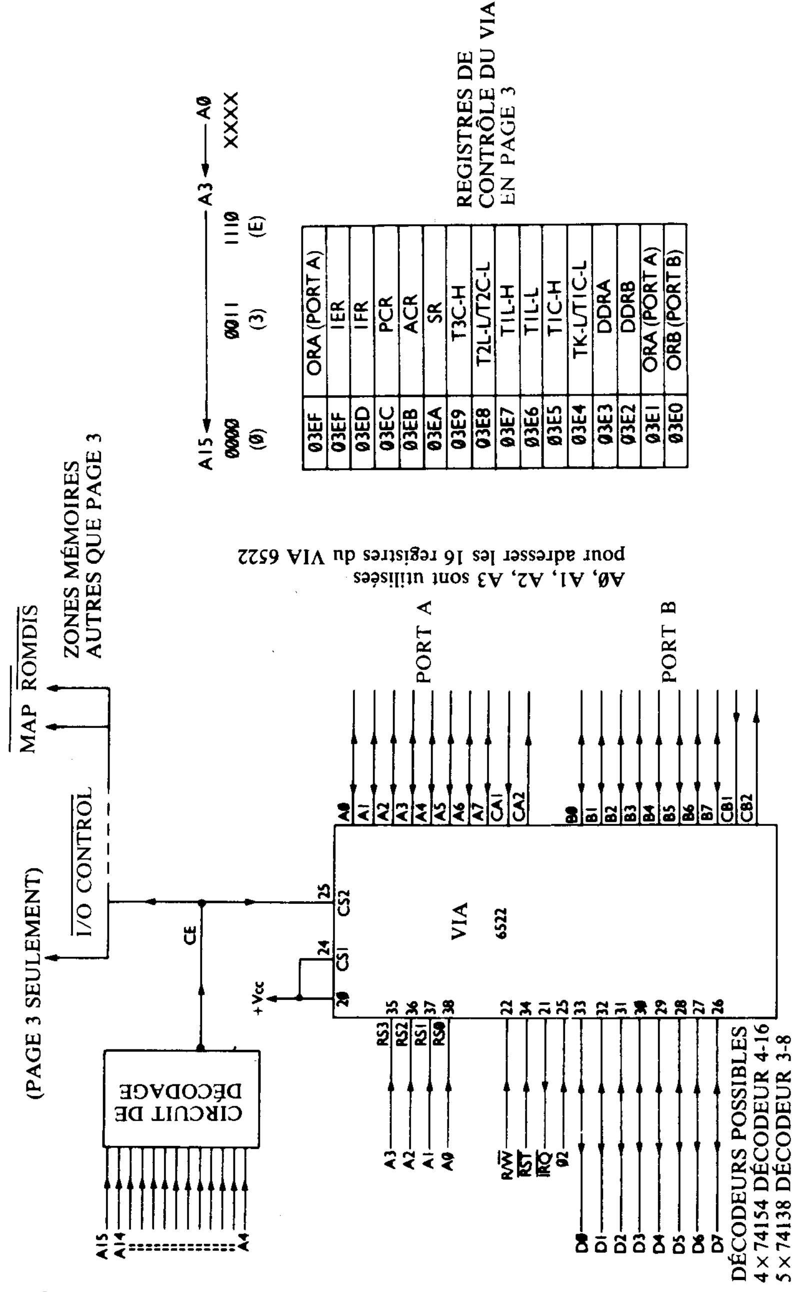




EXEMPLES DE CIRCUITS D'ENTRÉES/SORTIES POUR L'ORIC

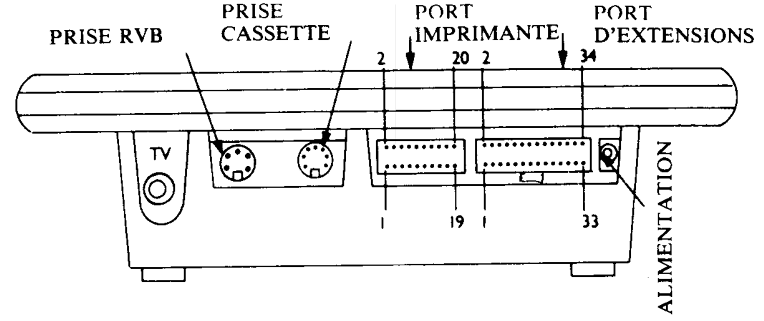
Numérisation http://wsc.n3.net

Numérisation http://wsc.n3.net



Adresse	Registre
Ø3EF	ORA (PORT A)
Ø3EF	IER
Ø3ED	IFR
Ø3EC	PCR
Ø3EB	ACR
Ø3EA	SR
Ø3E9	T3C-H
Ø3E8	T2L-L/T2C-L
Ø3E7	T1L-H
Ø3E6	T1L-L
Ø3E5	T1C-H
Ø3E4	TK-L/T1C-L
Ø3E3	DDRA
Ø3E2	DDRB
Ø3E1	ORA (PORT A)
Ø3E0	ORB (PORT B)

ANNEXE 11
Connecteurs de l'ATMOS

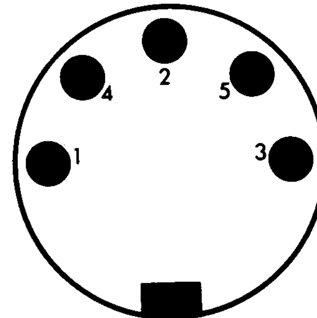


PRISE D'EXTENSIONS

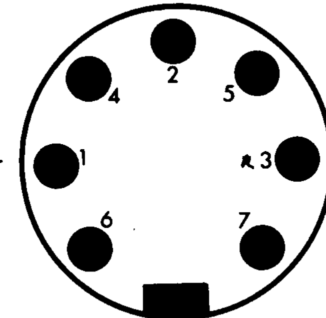
2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34
				Dd	D1	D6	D3	D4	A4	D7	A15	A14	A13	A12	A11	GND
1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33
MAP	02	I0	RW	D2	A3	A0	A1	A2	D5	A5	A6	A7	A8	A9	A10	+5v

PRISE D'IMPRIMANTE

2	4	6	8	10	12	14	16	18	20
GRD									
1	3	5	7	9	11	13	15	17	19
STB	D0	D1	D2	D3	D4	D5	D6	D7	ACK



PRISE RVB
1 Rouge
2 Vert
3 Bleu
4 Cyan
5 Masse

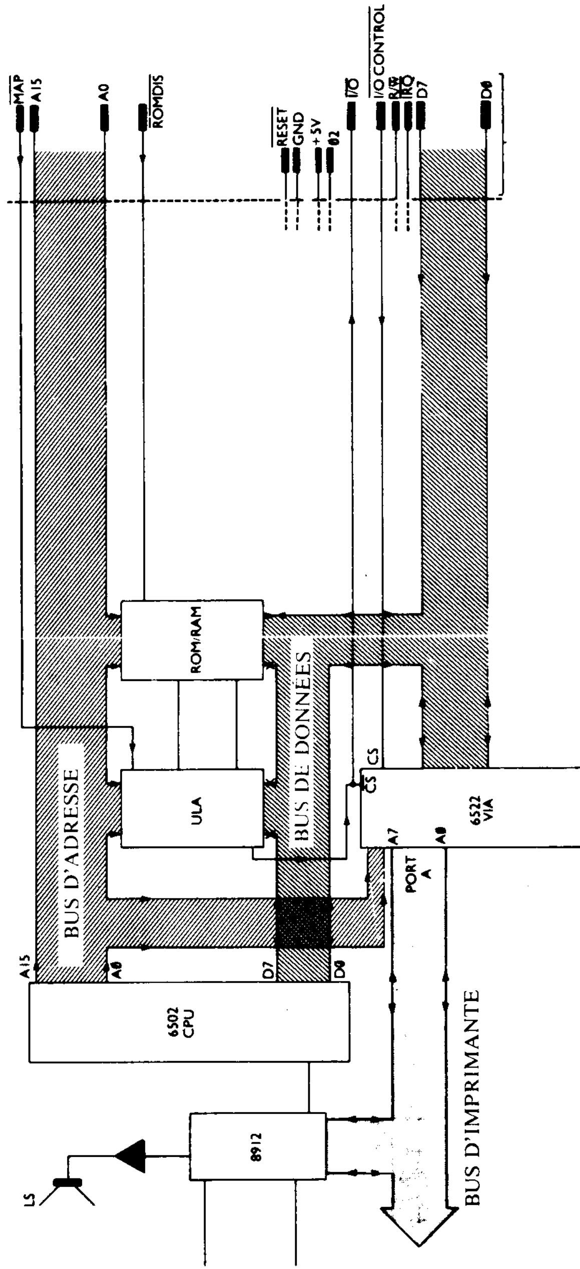


PRISE CASSETTE
1 Sortie
2 Masse
3 Entrée cassette
4, 5 } Son
6, 7 } Relais

William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Numérisation http://wsc.n3.net



SCHEMA GÉNÉRAL

ANNEXE 12

Mots réservés au Basic ainsi que leur codes

Voici la liste des mots réservés au Basic ainsi que les codes qui permettent de les ranger en mémoire sur un octet. Ces mots sont réservés et ne doivent pas figurer dans les noms de variables.

Mot	Code	Mot	Code	Mot	Code
ABS	216	FOR	141	PAPER	177
AND	209	FRE	218	PATTERN	174
ASC	236	GET	190	PEEK	230
ATN	229	GO	247	PI	238
AUTO	199	GOSUB	155	PING	166
CALL	191	GOTO	151	PLAY	169
CHAR	11	GRAB	159	PLOT	135
CHR$	237	HEX$	220	POINT	243
CIRCLE	173	HIMEM	158	POKE	185
CLEAR	189	HIRES	162	POP	134
CLOAD	182	IF	153	POS	219
CLS	148	INK	178	PRINT	186
CONT	187	INPUT	146	PULL	136
COS	226	INT	215	READ	149
CSAVE	183	KEY$	241	RECALL	131
CURMOV	171	LEFT$	244	RELEASE	160
CURSET	170	LEN	233	REM	157
DATA	145	LET	150	REPEAT	139
DEEK	231	LIST	188	RESTORE	154
DEF	184	LLIST	142	RETURN	156
DIM	147	LN	224	RIGHT$	245

William Saint-Cricq / wsaintcr@wanadoo.fr

15/08/2000

Numérisation http://wsc.n3.net

DOKE	138	LOG	232	RND	223
DRAW	172	LORES	137	RUN	152
EDIT	129	LPRINT	143	SCRN	242
ELSE	200	MID$	146	SGN	214
END	128	MUSIC	168	SHOOT	163
EXP	225	NEW	193	SIN	227
EXPLODE	164	NEXT	144	SOUND	167
FALSE	240	NOT	202	SPC	197
FILL	175	ON	180		
FN	196	OR	210		

INDEX

ABS **40, 49, 129**
Adressage **47, 226, 231**
  en code machine **227**
AND **58, 130**
Apostrophe **27**
Arrondis **34**
ASC **35, 131**
ASCII (codes) annexe
ATN **40, 132**
Attributs **93**
  code ASCII annexe
  effet de Escape **93**
AUTO

BASIC
  interpréteur **15, 219**
  langage **15**
  mots réservés annexe
  codes annexe
  Stockage de programme **69**
Binaire
  notation **34, 69, 221**
  table de conversion.
Bits **65, 97**
Boucles **43**
  FOR... NEXT **154**
  emboîtées **46, 56**
  REPEAT... UNTIL
Break **27**

CALL **133**
  Redéfinis **107, 110**
  non affichables **36**
en Haute Résolution **99**

Cassette (stockage sur)
  lancement automatique du programme **81**
  jonction de deux programmes **82**
  chargement d'un programme **78**
  sauvegarde d'un programme **78**
  stockage et rappel de tableaux **83**
  stockage et rappel de blocs mémoire **81**
Cassettes
  Soins et choix **78, 84**
CHAR **105, 134**
CHR$ **135**
CIRCLE **101**
Classement **54, 55**
CLEAR **30, 74, 135**
CLOAD **78, 137**
CLS **28, 139**
Conditions **49**
CONT **139**
Contrôle (caractère de) **84, 91, 94**

COS **40, 140**
  Couleur
  attributs **90**
  effet d'Escape **93**
  caractère inverses **95**
CSAVE **78, 141**
CTRL **21**
CURMOV **100, 142**
CURSET **99, 142**
Curseur **11, 15**

DATA **44, 46, 143**
DEEK **70, 144**
DEF **144**
DEF FN **41, 144**
DELete (touche) **20, 22**
DIM **44, 83, 145**
DOKE **70, 147**
Dollar ($) **29**
Double hauteur **94**
DRAW **99, 148**

EDIT **30, 149**
ELSE **29, 163**
END **149**
Entrées/Sorties **257**
Enveloppe **122**
ESCape (effets) **92, 93**
Exclamation (point d') **235**
EXP **40, 150**
EXPLODE **43, 150**

FALSE **53, 59, 151**
FILL **103, 152**
FN **40, 153**
FOR... NEXT **43, 154**
FRE **155**

Gamme chromatique **114**
GET **30, 35, 156**
GOSUB **60, 157**
GOTO **29, 60 158**
GRAB **69, 159**
Graphiques
  caractères **106**
  haute résolution **99**
  basse résolution **95**
  résolution moyenne **96**
  sur imprimante Oric MCP-40 annexe
  texte **107**

Haute résolution **99**
HEX$ **38, 160**
Héxadécimal **35, 221**
  tables de conversion annexe

William Saint-Cricq / wsaintcr@wanadoo.fr
Page 161 sur 163
15/08/2000

Numérisation http://wsc.n3.net

HIMEM 61
HIRES (mode) 69, 99, 162
grilles d'affichage annexe

IF... THEN 50, 163
ILLEGAL QUANTITY ERROR 38, 189
imprimante annexe
INK 90, 164
INPUT 26, 33, 165
INT 38, 40, 166
Interruption 245

KEY$ 167
LEFT$ 39, 168
LEN 39, 168
LET 18, 169
LIST 23, 24, 170
LLIST 171
LN 40, 71
LOG 40, 71
LORES 0 95, 172
LORES 1 172
caractères alternés annexe
grille d'affichage annexe
LPRINT 173

Machine (code)
adressage 231
instructions 233
registres 228
Mantisse 74
Mémoire annexe
MID$ 39, 173
MUSIC 113, 144

NEW 175
NEXT 154
Nombres
Binaires 220
codes binaires DCB 225
décimal 220
hexadécimal 221
négatif 223
NOT 58, 176

Octets 65
ON 60, 177
Opérateur :
artithmétiques 17
conditionnel 49
ordre de priorité 18
logique 58
OR 58, 178
OUT OF DATA ERROR 83
OUT OF MEMORY ERROR 47

Pages 68
PAPER 90, 179
PATTERN 101, 180
PEEK 69, 180
PI (π) constante 40, 181
PING 113, 181
PLAY 113, 120, 182
PLOT 95, 184
POINT 102, 185
Point virgule 17, 29, 36, 88
POKE 70, 96, 104, 185
POP 186
POS 187
pour cent (%) 34, 45
PRINT 91, 95, 188
PRINT 19, 87, 95, 188
Priorité 18
PULL 57, 189

RAM (mémoire vive) 68, 77
READ 44, 46, 191
RECALL 83, 192
REDIM'D ARRAY ERROR 145
REDO FROM START 165
Registres 228
RELEASE 69, 192
REM 62, 192
abréviation 27
REPEAT... UNTIL 47, 52, 194
Réservés (mots) 20
Reset 16
RESTORE 194
RETURN 60, 195
RIGHT$ 39, 196
RND 40, 197
ROM 40, 68
RUN 30, 198

SCRN 96, 199
SGN 40, 201
SHIFT (touches majuscules) 16, 22
SHOOT 113, 201
SIN 40, 202
SOUND 113, 125, 204
SPC 70, 206
SQR 40, 207
STEP 43, 154
STOP 207
STORE 83, 208
STR$ 37, 209
SYNTAX ERROR 18, 25

TAB 89, 210
Tableaux 44, 46
TAN 40, 211
TEXT 69, 212
grille d'écran annexe
THEN 163
TO 154
TROFF 212
TRON 213
TRUE 53, 59
TV 8

UNTIL 194
USR 215

VAL 37
Variables 19, 29
virgule flottante 74
de boucle 43
stockage 74
chaîne 28
Virgule 87

WAIT 28, 216

ZAP 113, 117, 119, 217

ACHEVÉ D'IMPRIMER
SUR LES PRESSES DE
L'IMPRIMERIE HÉRISSEY
A ÉVREUX (EURE)
LE 3 AVRIL 1984

D.L. : Avril 1984 — N° d'imp. 34415
Imprimé en France

15/08/2000 Page 162 sur 163

William Saint-Cricq / wsaintcr@wanadoo.fr

William Saint-Cricq
http://wsc.n3.net
ou
http://perso.wanadoo.fr/william.saint-cricq