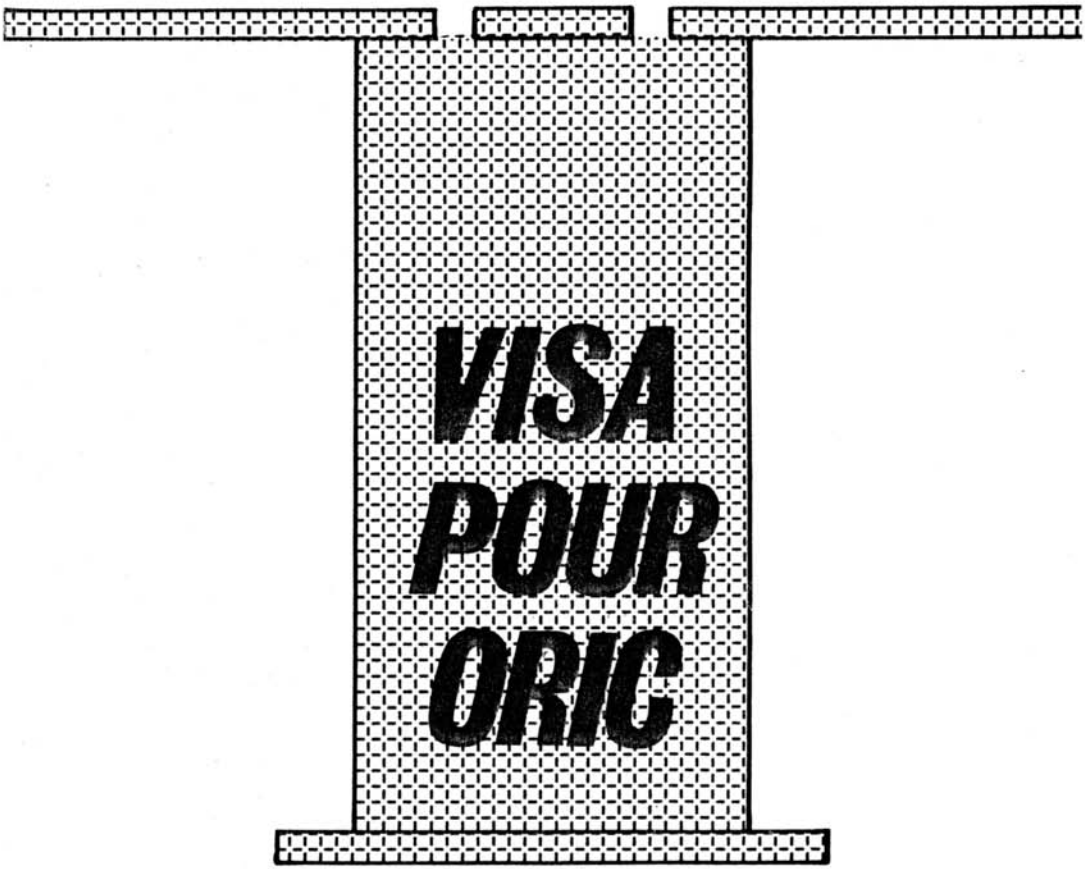
VISA POUR ORIC

FREDERIC BLANC FRANCOIS NORMANT



DIFFUSION
ASN Z.I. «La Haie Griselle»
94470 Boissy-St-Léger

INTRODUCTION

L'ORIC ouvre aujourd'hui une ère nouvelle dans le domaine de la microinformatique individuelle.
Pourtant d'un prix modique, celui-ci apporte de nombreuses possibilités d'utilisation.
Nous n'avons voulu en aucun cas dans cet opuscule donner les bases de la programmation qui sont supposées acquises par le lecteur.
En effet ce livre rassemble un ensemble d'astuces permettant à un utilisateur averti de tirer un meilleur parti de son appareil.
Bien sûr d'autres astuces restent à trouver, notre courte recherche ne nous ayant pas permis de tout découvrir. Nous proposons seulement quelques éléments du puzzle que forme l'ensemble des variables systèmes. Cet ouvrage ne se veut donc pas exhaustif, mais se classe plutôt comme un simple carnet de trucs et astuces utiles pour la programmation.
Nous espérons que la lecture de ce livret vous permettra de vous familiariser avec l'ORIC et d'approfondir vos connaissances sur les spécificités de cet ordinateur.
Sans plus attendre passons aux « choses sérieuses » sans oublier cependant de remercier tout particulièrement Jacques et Georges Dagousset pour leur participation aux « recherches ».

SOMMAIRE

- LES POINTEURS	#9A,	
	#9B	154, 155
	#501,	1281,
	#502	1282
	#E9,	
	#EA	233, 234
	#B0,	
	#B1	176, 177
	#A8,	
	#A9	168, 169
	#A6,	
	#A7	166, 167
	#A2,	
	#A3	162, 163
	#9E, #9F	158, 159
- LE MAGNETO	#302	770
	#5F,	
	#60	95, 96
	#61,	
	#62	97, 98
	#67	103
	#64	100
	#63	99
	#35	53
	#E57B	58747
	#E6CA	59082
	#E804	59396
	#E4A8	58536
- LE CLAVIER	#20E	526
	#306,	774,
	#307	775
	#E6CA	59082
	#E804	59396
	#20A	522
	#208	520
	#209	521
	#35	53

- NEW	#501,	1281,
	#502	1282
	#9E, #9F	158, 159
	#9C,	
	#9D	156, 157
	#A0,	
	#A1	160, 161
- CTRL et POKE	#26A	618

INDEX DES ADRESSES

Adresse		Chapitre(s) où se reporter
HEXA	**DECIMAL**	
#12, #13	18, 19	ECRAN
#1E	28	L'ECRAN
#35	53	LE CLAVIER, MAGNETO
#5F, #60	95, 96	MAGNETO
#61, #62	97, 98	MAGNETO
#63	99	MAGNETO
#64	100	MAGNETO
#67	103	MAGNETO
#9A, #9B	154, 155	LES POINTEURS, NEW
#9E, #9F	158, 159	LES POINTEURS, NEW
#A2, #A3	162, 163	LES POINTEURS
#A6, #A7	166, 167	LES POINTEURS
#A8, #A9	168, 169	LES POINTEURS
#B0, #B1	176, 177	LES POINTEURS
#E9, #EA	233, 234	LES POINTEURS
#208	520	LE CLAVIER
#209	521	LE CLAVIER
#20A	522	LE CLAVIER

#20C	524	L'ECRAN, INITIALISATION ET PROTECTION
#20E	526	LE CLAVIER
#22B	555	INITIALISATION ET PROTECTIONS
#268	616	L'ECRAN
#269	617	L'ECRAN
#26A	618	CTRL et POKE
#26B	619	L'ECRAN
#26C	620	L'ECRAN
#26D	621	L'ECRAN
#26E	622	L'ECRAN
#26F	623	L'ECRAN
#276, #277	630, 631	LE TIMER
#2C0	704	L'ECRAN
#302	770	MAGNETO
#306, #307	774, 775	LE CLAVIER
#501, #502	1281, 1282	NEW, LES POINTEURS

Routines en ROM utiles :

#E4A8	58536	MAGNETO
# E57B	58747	MAGNETO
#E6CA	59082	MAGNETO, LE CLAVIER
#E804	59396	MAGNETO, LE CLAVIER
#F42D	62509	INITIALISATION ET PROTECTIONS
#F430	62512	INITIALISATION ET PROTECTIONS
#F882	63562	INITIALISATION ET PROTECTIONS
#F84A	63618	INITIALISATION ET PROTECTIONS

plus évidemment toutes les adresses des fonctions que nous ne mentionneront pas ici (cf. chapitre FONCTIONS).

L'ÉCRAN

Une bonne utilisation de l'écran permet une programmation plus simple et plus efficace. Voici les quelques astuces, concernant l'écran et l'affichage, que nous avons pu rassembler :
L'écran TEXT occupe la zone mémoire comprise entre les adresses #BB80 et #BFDF (ou 48000 et 49119 en décimal).
L'écran HIRES occupe la zone mémoire comprise entre les adresses #A000 et #BF3F (ou 40960 et 48959) pour les 200 lignes de 40 cellules et la zone comprise entre #BF40 et #BFDF (ou 48960 et 49119) pour les 3 lignes de bas d'écran.
En mode TEXT, l'utilisation de POKE dans les adresses données permet divers effets : exemples :
* le mot CAPS figurant en haut de l'écran à droite est souvent désagréable lors de l'exécution d'un programme. On peut supprimer le mot CAPS (et rester cependant en majuscules) en faisant :

```
POKE # BBA4,32
POKE # BBA5,32
POKE # BBA6,32
POKE #BBA7,32
```

ou bien la routine : 10 FOR I = #BBA4 TO #BBA7 : POKEI,32 : NEXTI
le 32 poker aux diverses adresses correspond à un blanc ce qui se traduit dans la pratique par l'effacement du caractère se trouvant à cette adresse (nous verrons plus tard un moyen encore plus simple de supprimer le mot CAPS).

* On peut également, toujours à l'aide de POKE et des adresses données, définir la couleur de l'encre de telle ou telle ligne ou bien sa couleur de fond : si vous essayez le programme suivant vous comprendrez tout de suite (il faut que l'écran contienne du texte).

```
10 FOR I = # BB80 TO # BFDF STEP 40
20 POKE I,27 : POKE I+1, INT(RND(1)*8)
30 NEXT.
```

Ce programme a pour but de déterminer la couleur de l'encre de chaque ligne de manière aléatoire (le fond est alors noir puisqu'on ne le définit pas), (à condition qu'il soit noir au départ). Si maintenant vous remplacez la ligne 20 par : 20 POKE I,27 : POKE I+1, INT(RND(1)*8) + 16 vous définissez le fond de chaque ligne de manière aléatoire.

Vous pouvez aussi définir le fond et l'encre d'une ligne donnée ; mais rappelez-vous tout de même que la cellule où vous avez **POKE le « 27 » et la suivante** ne seront plus utilisables sous peine de perte de l'effet voulu. Vous perdez ainsi 2 cellules mémoire lorsque vous redéfinissez l'encre ou le fond à un endroit donné de l'écran.

* Ces adresses peuvent également servir à afficher des parties de textes en inversion vidéo : pour obtenir l'inversion vidéo, il suffit de POKER à l'adresse désirée le code ASCII du caractère voulu augmenté de 128 :

Par exemple si vous exécutez la ligne :

```
POKE #BBA4,67+128 : POKE #BBA 5,65+128 :
POKE # BBA6,80+128 : POKE # BBA7,83+128.
```

Vous verrez apparaître le mot CAPS non plus en BLANC sur fond NOIR mais en noir sur fond BLANC. Vous pouvez faire de même dans tout l'écran. Si le mot CAPS vous gêne, vous l'effacez. Mais si par hasard vous devez passer en minuscules puis repasser en majuscules, il réapparaîtra si vous le faites par CTRL T. Il existe une autre méthode permettant de conserver l'écran sans CAPS. Il suffit de modifier

une variable système qui est l'indicateur minuscule/majuscule. Son adresse est #20C (ou 524 en décimal).
Si vous êtes en majuscules, ?PEEK(#20C) donne 255. Si vous êtes en minuscules, ?PEEK(#20C) donne 127. Donc, pour changer de mode, vous n'avez plus qu'à POKER 127 ou 255 selon le cas dans la cellule #20C. Pour vous amusez, vous pouvez également essayer d'autres valeurs ; mais alors attention, car vous perdez le clavier actuel pour en récupérer un autre : à titre d'exemple, POKE 524,234 donne le résultat suivant :
lorsqu'on tape Q on obtient @
lorsqu'on tape W on obtient B
lorsqu'on tape G on obtient C
lorsqu'on tape R on obtient B
lorsqu'on tape T on obtient C
lorsqu'on tape Y on obtient H
etc.

Les résultats sont surprenants mais obligent souvent à débrancher l'ORIC ou de faire un RESET, pour récupérer un clavier utilisable.
Passons à présent à d'autres petites astuces concernant l'affichage :
— Souvent, vous redéfinissez des caractères lors d'un programme et lorsque vous voulez lister ce programme, il devient incompréhensible à cause des caractères redéfinis. Vous pouvez faire un Reset mais ceci oblige d'une part à retourner l'ordinateur et en plus, vous perdez votre encre, votre papier et tout un tas d'autres choses que vous avez redéfinies. Il existe une méthode plus simple : CALL #F89B, cette routine remet tous les caractères dans leur état initial.
— Il peut vous arriver d'être gêné par le « Ready » : en effet, si au mode TEXT celui-ci ne pose pas de problème puisqu'il laisse voir les erreurs faites étant

donné le nombre de lignes, il peut devenir indésirable en HIRES : en effet, si vous exécutez une ligne en HIRES et que vous faites une faute ou si vous voulez connaître la valeur d'une variable, les messages SYNTAX ERROR et Ready peuvent vous gêner : si vous faites POKE #1E,0 (ou POKE 28,0) le message Ready n'apparaîtra plus, et lors d'un message d'erreur, le curseur restera sur la même ligne que le message vous permettant ainsi de voir votre faute où la variable désirée selon le cas. A l'origine ?PEEK(28) = 203.

Pour rétablir le Ready, il faut donc faire POKE 28,203. Si vous POKEZ autre chose à cette adresse, vous bloquerez purement et simplement l'ORIC.

— A l'origine, l'ORIC est censé éditer des textes ou des listings sur 28 lignes. En fait, seules 27 de ces lignes sont utilisables puisque la 28e est la ligne des STATUS (CAPS, Searching...). Il existe un moyen simple de récupérer cette ligne : l'adresse #26F ou 623 contient le nombre de lignes sur lequel on peut travailler (c'est-à-dire le nombre de lignes utilisées dans un listing ou effacé lors d'un CTRL L). A l'origine, ?PEEK(#26F) = 27 (nombre réel de lignes utilisables). On pense alors à faire POKE #26F,28 ; ceci a pour effet d'ajouter effectivement une ligne, mais en bas de page. Donc, lorsque vous ferez un listing, celui-ci descendra un peu plus avant de translater l'ensemble de l'écran. Cependant, la mémoire écran étant trop petite, la remontée des lignes fera apparaître des choses curieuses. Le POKE #26F,28 ne suffit donc pas. Il faut en plus agir sur une autre commande qui se trouve à l'adresse #26D, à l'origine, ?PEEK #26D = 128. Si au lieu de laisser 128, ou exécute POKE #26D,88 alors la première ligne des STATUS est utilisable (le 88 correspond à 128-40, 40 étant le nombre de caractères dont on veut reculer = 1 ligne).

Il faut donc pour avoir ces 28 lignes faire POKE 623,28 : POKE 621,88, à noter que le fait d'avoir 28 lignes nous supprime en même temps le mot CAPS comme vous le voyez, les adresses 621 et 623 sont très importantes dans l'édition de textes à l'écran. On peut donc s'en servir à d'autre fins :
Supposons à présent qu'il vous faille conserver les dix lignes du bas de votre écran et travailler sur les 17 restantes. Il vous suffit alors de faire POKE 623,17 : alors tous les listings ne se feront plus que sur les 17 lignes du haut de l'écran, le CTRL L n'agira plus que sur ces 17 lignes.
Si vous POKEZ un paramètre supérieur à 28 dans la cellule 623, vous aurez quelques problèmes étant donné la longueur de la mémoire écran en mode TEXT. Vous pouvez cependant, vous ne risquez rien.

De même, vous pouvez vous resservir de la cellule 621 à votre gré : si au lieu de faire POKE 621, 88, vous faites POKE 621, 108 le Ready sera affiché au milieu de l'écran et vos listings commenceront à s'imprimer à partir de cet endroit-là ; de plus, les 20 colonnes de droite, seront normales et les 20 de gauche en inversion vidéo.

— Vous pouvez également, si vous le désirez redéfinir la couleur de l'encre et du fond uniquement dans la zone définie par le POKE 623,?? :
Essayez l'exemple qui suit :
POKE 623,15 : PAPER 3 : le résultat est le suivant : le POKE 623,15 définit le nombre de lignes utilisables. Le PAPER 3 change la couleur du fond de l'ensemble de l'écran. Tapez maintenant : PAPER 7 :
POKE 623,15 : POKE 619,17 : faites alors remonter le curseur jusqu'en haut de l'écran et continuez à appuyer sur la touche ↑ : vous voyez alors la couleur rouge du fond qui n'apparaît que sur les 15

lignes définies en 623.
La cellule 619 (#26B) définit donc la couleur du fond dans le nombre de lignes utilisables et seulement dans celles-là, le paramètre à POKER en 619 est évidemment compris entre 16 et 23 inclus.
De même, la cellule 620 (#26C) définit la couleur de l'encre dans la zone utilisable. Le paramètre à POKER en 620 est compris entre 0 et 7 inclus.
Nous avons vu tout à l'heure l'utilité des adresses 621 et 623. En fait, l'adresse 621 est couplée avec l'adresse 622 pour définir le plancher écran c'est le début de la zone sur laquelle l'ordinateur listera un programme et la zone dont vous disposerez. Essayez l'exemple suivant : introduisez un programme basic dans l'ORIC pour pouvoir faire un listing. Lorsque ceci est fait tapez : DOKE 621, 48400
POKE 623,7
CLS
puis faites LIST. Je pense que vous avez compris le processus grâce à cet exemple. De même, si on est en HIRES, ces 3 cellules permettent des choses abérrantes : essayez l'exemple suivant : introduisez un programme dans l'ORIC comme précédemment. Puis tapez ce qui suit :
HIRES
DOKES 621, 40960
POKE 623, 200
puis faites LIST. Etonnant non !? Vous pouvez ainsi avoir 200 lignes en même temps, malheureusement, il faut un œil assez exercé pour arriver à lire un tel listing.
Il reste deux autres adresses à explorer dans la même région, ce sont les adresses 616 et 617. Ces adresses commandent le curseur. L'adresse 617 peut servir à la tabulation horizontale : tapez par exemple : POKE 617,15 : ?« COUCOU » ceci vous imprimera le mot « COUCOU » à la ligne suivante, 15 cases après le départ de l'écran.

On peut ainsi créer un print A en basic : après avoir poké en 616 la valeur de la ligne, faire un print pour activer le curseur à la ligne prévue puis poker en 617 la valeur de la colonne - 1. En ligne 8 le POKE 618,10 enlève un curseur immobile et superflu.

```
0 REM SIMULATION DU PRINT @
1 CLS
5 INPUT"COLONNE,LIGNE";X,Y
6 IFX<1ORY<1THENPING:GOTO1
7 IFX>40ORY>27THENPING:GOTO1
8 POKE618,10
10 POKE616,Y-1:PRINT:POKE617,X-1:PRINT"*";
20 PRINT"AAAA";
25 POKE618,11
30 GETA$:PRINTA$;:IFA$="/"THENPOKE618,3:END
40 GOTO30
```

Une autre adresse concernant l'écran est l'adresse #12 (ou 18 en décimal). Cette adresse contient la position de la prochaine impression : essayer par exemple DOKE 18,48034 : ?« GLOU ».
Ceci va vous imprimer GLOU à la place de CAPS car l'adresse DOKE en 18 est celle de la cellule précédant CAPS de 1 caractère.
Vous pourrez appliquer le même principe dans tout l'écran. Les adresses à DOKER en 18 vont de 47998 à 49117.
Enfin, une dernière adresse concernant l'écran, l'adresse #2CØ (= 704 en décimal). Celle-ci sert à déterminer si on retrouve en mode TEXT ou en HIRES. Si on est en mode TEXT, ?PEEK(#2CØ) =

2.
Si on est en HIRES, ?PEEK(#2C0) = 3.
Utilité : on peut se servir des fonctions propres à HIRES en mode TEXT en POKANT 3 en #2CØ de même, on peut se servir des fonctions propres au mode TEXT en HIRES en POKANT2 en #2C0 il suffit ensuite de construire une routine permettant de passer de TEXT à HIRES sans effacer le dessin se trouvant en HIRES. De même pour le passage HIRES-TEXT.

Voici à présent un exemple vous permettant de comprendre mieux l'utilité de l'adresse absolue pour afficher quelque chose à l'écran : tapez le programme suivant :
10 CLS : POKE 618,2
20 FOR I = #BB8Ø TO #BFDC (adresses de début et de fin d'écran)

3Ø POKE I − 1,32 (efface le précédent 0)
4Ø POKE I,79 ; POKE I+1,82 ; POKE I+2,73 POKE I+3,67
5Ø WAIT 1Ø : NEXT
6Ø RUN 1Ø.
Vous voyez à l'exécution le mot ORIC qui défile sur l'écran ; ce programme monte la légèreté de la programmation utilisant POKE et les adresses écran par rapport à la lourdeur de la fonction PLOT avec laquelle on aurait besoin de plus de temps et de plus en plus de place pour faire la même chose.

INITIALISATIONS ET PROTECTIONS

* Simulation d'une initialisation : au lieu de débrancher l'ORIC du courant lorsque l'on veut tout réinitialiser (programmes, variables, écran, contrôle, encre papier, etc.), taper CALL #F42D ou bien CALL #F84A. Au bout de 3 secondes, vous voyez apparaître le message
ORIC EXTENDED BASIC VI.O
C 1983 TAGERINE
47870 BYTES FREE
Ready
soit le message qui apparaît lors de la mise sous tension.

* Simulation d'un Reset : au lieu de retourner l'appareil, ce qui est fort désagréable, tapez CALL 555
ou CALL #F430 = CALL 62512
ou CALL #F882 = CALL 63618

Utilisation pour la protection des programmes :
En premier lieu, on peut bloquer le Reset. Si vous désassemblez à partir de l'adresse 555, vous trouvez la routine suivante :

JMP #F430

soit en HEXA :
#20
#30
#F4

Si vous supprimez le JMP pour le remplacer par un RTI (Return from Interuption) le Reset est bloqué. Le code du RTI est #40 = 64, donc pour réaliser cette opération, tapez simplement POKE 555,64. Mais vous pouvez faire encore mieux en remplaçant #F430 par une autre adresse. Si vous prenez par exemple l'adresse du ZAP qui est #F41B et que vous faites DOKE 556, #F41B lorsque vous appuyez sur la touche RESET vous obtenez un ZAP, et l'ORIC est ensuite bloqué, voici déjà une protection sommaire.
Si vous voulez faire une protection encore plus efficace, vous n'avez qu'à utiliser l'adresse du départ à froid. Faites DOKE 556, #F42D et ensuite RESET. L'ordinateur se réinitialise donc et perd tout ce qu'il avait en mémoire.

* Blocage du clavier

Cependant la protection du RESET n'est pas absolue puisque l'utilisateur peut l'arrêter par un CTRLC. Pour éviter ceci il faut bloquer le clavier.

— soit en supprimant la scrutation, ce qui est expliqué dans le chapitre sur le clavier.

— soit en faisant DOKE #400, #6078 puis CALL #400.

Ce qui correspond en langage machine aux mnémoniques suivants :

#400 SEI
#401 RTS.

L'exécution de la routine interdit les interruptions et rend la main au basic.
Pour rétablir les interruptions, il suffit de taper DOKE #402, #6058.
Soit en langage machine, les mnémoniques suivants :

#402 CLI
#403 RTS.

L'exécution de la routine par CALL #402 rétablit les interruptions et rend la main au basic.

Ces deux routines doivent être utilisées ensemble dans un programme car si vous utilisez la première en basic, vous perdez évidemment la main.
A noter que l'on retrouve les mêmes inconvénients avec ces routines qu'avec la suppression de la scrutation clavier (se rapporter au chapitre sur le clavier).
— Enfin un dernier moyen pour bloquer votre clavier. Vous pouvez POKER n'importe quel nombre 0 et 255, sauf 127 et 255 à l'adresse 524 (qui correspond à la bascule majuscule/minuscule). Une autre valeur ne fait pas perdre le clavier, mais change la signification des touches.
Ainsi CTRL C n'arrêtera pas le programme d'une manière générale (sauf exception) et même n'aura parfois plus de correspondance, certaines touches étant répétées et d'autres supprimées.
Nous allons maintenant, pour illustrer ces systèmes de protections, vous donner un exemple de programme protégé :

```
1Ø DOKE 556, #F41B     sautera sur l'adresse
2Ø POKE 555, #20       du ZAP lors
                       d'un RESET
                       (JSR #F41B)
3Ø DOKE #400,          Routine inhibant
   #6078               les interruptions
4Ø DOKE #402,          Routine rétablissant
   #6058               les interruptions
5Ø CALL #402           rétablit
                       les interruptions
6Ø GETR $
7Ø CALL #400           supprime
                       les interruptions
8Ø PING : WAIT 1Ø      ZAP
9Ø SHOOT : WAIT 1Ø
   EXPLODE :
   WAIT 1Ø
1ØØ GOTO 5Ø
```

Dans ce programme, on bloque le RESET en remplaçant à l'adresse de celui-ci le JMP #F430 par un JSR #F41B (= ZAP). On construit ensuite 2 routines en page 4 permettant de supprimer ou de rétablir les interruptions.
Puis on rétablit les interruptions pour permettre une prise de données et on les supprime immédiatement après pour exécuter le reste du programme. Ainsi l'utilisateur ne peut pas arrêter le programme par un CTRL C, s'il fait un RESET, il obtient un ZAP sans retour au basic. Il ne peut donc pas arrêter ce programme.

LES FONCTIONS DE L'ORIC

L'ORIC possède un certain nombre d'instructions, de fonctions et de messages d'erreurs définis par des codes opération compris entre 128 et 255.
La connaissance de code opération peut servir à identifier une fonction dans la RAM par la fonction PEEK.
De plus, la connaissance des adresses de ces fonctions peut être utile (surtout si l'on travaille en assembleur).
Nous avons donc dressé la liste de ces fonctions avec leur code opération et leur adresse dans la ROM.
A noter que certaines de ces fonctions existent mais ne sont pas utilisables : c'est le cas des fonctions : INVERSE, NORMAL, GO.
La liste sera présentée sous forme de tableau comprenant le code opération de la fonction, son nom et son adresse en ROM.

Code HEXA	Code DECIMAL	FONCTION	Adresse ROM (HEXA et DECIMAL)
80	128	END	#C941 = 51521
81	129	EDIT	# C6A5 = 50853
82	130	INVERSE	#CFE4 = 53220
83	131	NORMAL	#CFE4 = 53220
84	132	TRON	#CC8C = 52364
85	133	TROFF	#CC8F = 52367
86	134	POP	# C9E0 = 51680
87	135	PLOT	# D9C6 = 55750
88	136	PULL	#DA16 = 55830
89	137	LORES	# D937 = 55607
8A	138	DOKE	#D8AC = 55468
8B	139	REPEAT	# D9FA = 55802
8C	140	UNTIL	#DA16 = 55830
8D	141	FOR	#C841 = 51265
8E	142	LLIST	#C824 = 51236
8F	143	LPRINT	#C832 = 51250
90	144	NEXT	#CEØC = 52748
91	145	DATA	#CAØA = 51722
92	146	INPUT	# CCC9 = 52425
93	147	DIM	#DØF2 = 53490
94	148	CLS	#CCOA = 52234
95	149	READ	#CCFD = 52477
96	150	LET	#CAD2 = 51922
97	151	GOTO	#C9B3 = 51635
98	152	RUN	#C98D = 51597
99	153	IF	#CA3E = 51774
9A	154	RESTORE	#C91F = 51487
9B	155	GOSUB	#C996 = 51606
9C	156	RETURN	#C9E0 = 51680
9D	157	REM	#CA61 = 51809
9E	158	HIMEM	#E95B = 59739
9F	159	GRAB	#E974 = 59764
AO	160	RELEASE	#E994 = 59796
A1	161	TEXT	#E9A9 = 59817
A2	162	HIRES	#E9BB = 59835

Code HEXA	Code DECIMAL	FONCTION	Adresse ROM (HEXA et DECIMAL)
A3	163	SHOOT	#F415 = 62485
A4	164	EXPLODE	#F418 = 62488
A5	165	ZAP	#F41B = 62491
A6	166	PING	#F412 = 62482
A7	167	SOUND	# FB26 = 62294
A8	168	MUSIC	# FBFE = 64510
A9	169	PLAY	# FBB6 = 64438
AA	170	CURSET	# F02D = 61485
AB	171	CURMOV	# F064 = 61540
AC	172	DRAW	# F079 = 61561
AD	173	CIRCLE	# E87D # F2E5 = 62181
AE	174	PATTERN	# F093 = 61587
AF	175	FILL	# F1E5 = 61925
B0	176	CHAR	# F0A5 = 61605
B1	177	PAPER	# E889 # F17F = 61823
B2	178	INK	# E889 # F18B = 61835
B3	179	STOP	# C93F = 51519
B4	180	ON	#CA78 = 51832
B5	181	WAIT	#D89D = 55453
B6	182	CLOAD	#E7AA = 59306
B7	183	CSAVE	#E7DB = 59355
B8	184	DEF	#D401 = 54273
B9	185	POKE	#D894 = 55444
BA	186	PRINT	#CB61 = 52065
BB	187	CONT	#C96E = 51566
BC	188	LIST	#C773 = 51059
BD	189	CLEAR	#C738 = 51000
BE	190	GET	#CCBA = 52410
BF	191	CALL	#E80D = 59405
C0	192	!	#CC89 = 52361
C1	193	NEW	#C71B = 50971
C2	194	TAB(	#DF13 = 57107
C3	195	TO	#DFA6 = 57254
C4	196	FN	#DF32 = 57138
C5	197	SPC(	# DRA0 = 53920

Code HEXA	Code DECIMAL	FONCTION	Adresse ROM (HEXA et DECIMAL)
C6	198	@	#D3D7 = 54231
C7	199	AUTO	#D3FB = 54267
C8	200	ELSE	#D918 = 55576
C9	201	THEN	# D2A0 = 53920
CA	202	NOT	#E22B = 57899
CB	203	STEP	#E34C = 58188
CC	204	+	#DC7A = 56442
CD	205	−	#E2A7 = 58023
CE	206	*	#E388 = 58248
CF	207	/	#E38F = 57255
D0	208	↑	#E3D8 = 58328
D1	209	AND	#E43C = 58428
D2	210	OR	#D87C = 55420
D3	211	>	#D8C9 = 55497
D4	212	=	#DDD1 = 56785
D5	213	<	#D7EC = 55276
D6	214	SGN	#D4D9 = 54489
D7	215	INT	#D81D = 55325
D8	216	ABS	#D7FB = 55291
D9	217	USR	#D75C = 55132
DA	218	FRE	#D8EF = 55535
DB	219	POS	#DFØ1 = 57089
DC	220	HEX $	#DEFD = 57085
DD	221	&	#DA5Ø = 55888
DE	222	SQR	#D9B5 = 55733
DF	223	RND	#E9CE = 59854
E0	224	LN	#D77Ø = 55152
E1	225	EXP	#D79C = 55196
E2	226	COS	#D7A7 = 55207
E3	227	SIN	#DA9A = 55962
E4	228	TAN	#DA83 = 55939
E5	229	ATN	#DCBA = 56506
E6	230	PEEK	#DDE3 = 56803
E7	231	DEEK	#E234 = 57908
E8	232	LOG	#DØ5A = 53338

Code HEXA	Code DECIMAL	FONCTION	Adresse ROM (HEXA et DECIMAL)	
E9	233	LEN	#DØ57 =	53335
EA	234	STR $	#E26D =	57965
EB	235	VAL	#CFBØ =	53168
EC	236	ASC	#DØ87 =	53383
ED	237	CHR $	#C44F =	50255
EE	238	PI		
EF	239	TRUE		
F0	240	FALSE		
F1	241	KEY $	#C554 =	50516
F2	242	SCRN		
F3	243	POINT		
F4	244	LEFT $	#CC42 =	52290
F5	245	RIGHT $		
F6	246	MID $		
F7	247	GO		
F8	248	NEXT WITHOUT FOR		
F9	249	SYNTAX		
FA	250	RETURN WITHOUT GOSUB		
FB	251	OUT OF DATA		
FC	252	ILLEGAL QUANTITY		
FD	253	OVERFLOW		
FE	254	OUT OF MEMORY		
FF	255	UNDEF'D STATEMENT		

Bien sûr, certaines de ces fonctions nécessitent certains paramètres. Il serait trop long ici de décrire tous les paramètres pour chaque fonction. Cependant, si vous désirez les connaître, il vous suffit, à partir de l'adresse que nous vous donnons pour une fonction de désassembler la partie de la ROM qui suit pour connaître l'emplacement et les conditions sur les paramètres nécessaires.

TIMER

Comme certains micros, ORIC possède une fonction TIME. Cette fonction peut servir à la temporisation lorsque l'on en a besoin. ORIC s'en sert lui-même lors de l'exécution de la fonction WAIT.
On peut également s'en servir pour construire une fonction RND de meilleure qualité. En effet, à l'initialisation de l'ORIC, la fonction RND est remise à une certaine valeur toujours identique pour un appareil donné. Si on exécute un même jeu 2 fois, juste après la mise sous tension, et si ce jeu utilise la fonction RND, il se déroulera toujours de la même manière.
L'adresse de cette fonction TIME est # 276, # 277. Pour obtenir un nombre aléatoire compris entre # ØØØØ et # FFFF, il suffit de faire un ?DEEK (# 276). Pour obtenir un nombre aléatoire compris entre Ø et 256, faire simplement ?PEEK (* 276). Le résultat obtenu est vraiment aléatoire car il n'y a rien de plus aléatoire que le temps.
Vous pouvez grâce à ce chrono, mesurer le temps écoulé entre 2 événements. Il ne s'agit en fait pas d'un chronomètre mais d'un décompteur (il compte « à l'envers »). Il faut tout d'abord initialiser ce décompteur, exécuter la tâche dont on veut mesurer la durée puis lire le résultat.
Tapez par exemple le programme suivant 1Ø DOKE # 276,Ø. Initialise le décompteur à zéro.

2Ø FOR I = 1 TO 1Ø ØØØ : NEXT boucle à vide dont on veut mesurer la durée. 3Ø PRINT (# FFFF - DEEK (#276) /100 temps écoulé, faites RUN puis attendez. Au bout d'un certain temps apparaît un temps de l'ordre de 16 secondes.
Vous avez tout simplement mesure le temps nécessaire à l'exécution, de 1Ø ØØØ boucles à vide.

A noter que si vous remplacez le NEXT par un NEXTI, le temps nécessaire pour effectuer les 1Ø ØØØ boucles passe à 20 secondes soit une augmentation d'environ 25 % du temps de calcul.
Le temps est grâce à ce chronomètre mesuré au centième de seconde. La capacité maximale de cc chronomètre est #FFFF en centièmes de seconde soit 10' 55'' 35/100e, ce qui est largement suffisant pour une utilisation courante.
D'autre part, ce chronomètre est très précis ; les mesures que nous avons fait sur # FFFF 100e de secondes (10' 55'') ne nous ont pas permis de déceler une variation par rapport à un chronomètre à quartz.
Vous pouvez si vous voulez, insérer un sous-programme de chronométrage lorsque vous en avez besoin pour cela, il vous faut réaliser les

- en secondes par ? (#FFFF-DEK (#276) / 100 ;
- en 100ème de secondes par ? (#FFFF-DEEK (#276) ;
- en minutes, secondes, 100ème par 5 A = #FFFF-DEEK (#276) : B = INT (A/1000) 10 ? B ; «min», INT ((A – 60 * B) / 100) ; «sec», 100 * (A/100 – INT (A/100)) ; «cent»

```
5 A=#FFFF-DEEK(#276):B=INT(A/1000)
10 PRINTB;"min",INT((A-60*B)/100);"sec",
100*(A/100-INT(A/100));"cent"
```

LES POINTEURS

En ce qui concerne les pointeurs au sujet du magnétophone, il faut se référer au chapitre « magnétophone ».
Il existe deux pointeurs dans le système qui définissent le début et la fin d'un programme Basic.
Le pointeur de début de programme se trouve à l'adresse #9A, #9B (pour l'obtenir tapez ?DEEK (#9A))

D'une manière générale, la valeur obtenue est #501 car les programmes Basic commencent toujours en bas de la mémoire. Mais on peut à son gré modifier la valeur qui se trouve à cette adresse : Tapez par exemple le programme suivant 1Ø ZAP : WAIT 1ØØ 2Ø EXPLODE.
Si vous l'exécutez, vous entendez un ZAP suivi d'un EXPLODE. Maintenant, tapez DOKE #9A, DEEK (#501). Si maintenant vous faites LIST vous ne voyez plus que 2Ø EXPLODE.
Pourquoi ? en #501 se trouve l'adresse de la ligne BASIC suivante.

Lorsque vous DOKEZ cette valeur en # 9A, vous fixez un nouveau plancher BASIC, le système ignore alors tout ce qui se trouve en dessous de cette adresse, que ce soit lors d'un listing ou de l'exécution du programme. Vous pouvez ainsi cacher des parties de vos programmes. Cependant,

ce .type de modification ne permet pas de sauvegarde « classique » ; c'est-à-dire que si vous exécutez un CSAVE normal, le magnétophone n'enregistrera rien du tout. Il faut pour enregistrer ce type de programme faire un CSAVE " ", A #501 qui redéfinit le début de l'enregistrement. Je vous laisse imaginer le système de protection de programme que vous pouvez élaborer à partir de cette adresse. Peut-être, me direz-vous que ceci ne sert à rien puisque lorsqu'on a modifié l'adresse se trouvant en #9A, on ne peut plus exécuter les lignes situées avant.

Réfléchissons ! Nous allons maintenant définir le pointeur programme qui pourra vous servir à faire un saut dans cette zone cachée.
Le pointeur programme a pour adresse : #E9 et #EA : vous pouvez, en programme, voir l'octet sur lequel vous vous trouvez par un ? DEEK (#E9).

Grâce à cette adresse, vous pourrez allez exécuter une routine cachée par le système vu précédemment. Il vous suffit de DOKER la bonne adresse en # 9. Exemple : tapez le programme suivant :

```
1 ZAP : WAIT 1ØØ
2 EXPLODE
3 END
10 DOKE #E9, #500.
```

Si vous tapez RUN 1Ø, vous entendez un ZAP suivi d'un EXPLODE.
Tapez à présent DOKE #9A, DEEK (#501) si vous listez le programme, vous obtenez :

```
2 EXPLODE
3 END
10 DOKE #E9, #500.
```

Le 1 ZAP : WAIT 1ØØ a disparu par le procédé expliqué plus haut.

* Si vous faites RUN, seul le EXPLODE est exécuté. En revanche, si vous tapez RUN 1Ø, le pointeur programme est positionné sur l'octet de départ du programme précédent donc le ZAP est exécuté, suivi du EXPLODE. Vous pouvez ainsi exécuter une zone que vous avez dissimulée.
Nous avons vu l'utilité du pointeur programme, voyons maintenant celle du pointeur data.
* Le pointeur data.
Il existe dans l'ordinateur, au même titre que le pointeur programme qui indique l'adresse de la ligne suivante à interpréter, un pointeur data qui donne l'adresse de l'information qui sera lue par la prochaine instruction READ.

Vous avez dû voir que si dans un programme vous avez un grand nombre de DATA, disons 1 000 pour donner un exemple et que vous avez besoin d'informations situées à partir du 800e DATA, vous êtes obligé de faire une boucle qui lit les 799 premiers DATA pour pouvoir avoir enfin le 800e, l'instruction RESTORE n n'existant pas dans le basic de l'Oric.
La connaissance du pointeur data va remédier à ce problème.

Ce pointeur a pour adresse #BØ et #B1. Vous pouvez lire une valeur en faisant ? DEEK (#BØ) et mettre une valeur en faisant DOKE (#BØ), XX.
Pour clarifier l'explication imaginez que vous avez dans un programme 4 catégories de noms situés dans des DATA :
Une catégorie de fruits, une d'animaux, une de fleurs et une de capitales.
Tapez pour commencer les quelques lignes suivantes :

```
1Ø DATA POMME, PECHE, ABRICOT, CERISE
2Ø DATA CHIEN, CHAT, SOURIS, LION, CHEVRE
3Ø DATA TULIPE, ROSE, MUGUET, LILAS
4Ø DATA PARIS, LONDRE, BERLIN, MADRID.
```

Imaginez qu'il s'agisse d'un jeu de pendu et que vous ayez besoin uniquement d'une catégorie précise à la fois.
Par la méthode normale, si vous avez besoin de la catégorie de fleurs, il vous faut faire restore et FOR I : 1TO9: READ A$: NEXT. Le pointeur sera alors positionné sur le premier nom de fleurs.
La méthode que nous vous proposons est beaucoup plus rapide puisqu'il s'agit de mettre directement dans le pointeur data l'adresse du premier nom de fleurs.
Pour ce faire, il nous faut l'adresse du début de chaque catégorie.
Ajoutez donc au programme tapé les lignes suivantes :

```
1ØØ RESTORE : CLS
11Ø FOR I : 1TO17, READ A$ : ?A$ ; SPC (10) ;
DEEK (#B0) : NEXT.
```

Ces deux lignes ont pour effet de remettre le pointeur data en début de liste et ensuite d'afficher pour chaque nom, l'adresse de celui qui suit.
Il vous suffira donc de noter précisément après avoir fait RUN les valeurs qui suivent les mots CERISE, CHEVRE, LILAS pour avoir respectivement les adresses du début des catégories animaux, fleurs, capitales.
Soit pour la première 1312, la seconde 1347, la troisième 1377.
Vous avez maintenant pour chaque catégorie l'adresse de départ (pour celle des fruits il suffit de faire RESTORE).
Vous allez donc pouvoir les utiliser pour faire appel à chaque catégorie individuellement selon vos besoins.

Continuons donc notre exemple et ajoutons les lignes suivantes :

```
10ØØ ? " Catégorie des fleurs : "
101Ø DOKE #B0, 1347
```

```
1Ø20 FOR I = 1TO4 : READ A$ :?A$; '' '' ; :
NEXT : ?
11ØØ ? '' CATEGORIE DES ANIMAUX : ''
111Ø DOKE #B0, 1312
1120 FOR I 3 1505 : READ A$ : ?A$ ; '' '' ; :
NEXT : ?
12ØØ ? '' CATEGORIE DES CAPITALES : ''
121Ø DOKE #BØ, 1377
1120 FOR I = 1 T04 : READA$ : PRINTA# ; ''␣'' ;
: NEXT : PRINT
1300 ? '' CATEGORIE DES FRUITS : ''
1310 RESTORE
1320 FOR I = 1T04 : READ A$ : '' '' ; : NEXT
```

Faites RUN 1ØØØ et vous voyez s'afficher chaque catégorie prise dans un ordre différent de celui des DATA.

Vous comprendrez donc aisément l'utilité de ce pointeur qui permet un gain de temps très appréciable et même une simplification des programmes malgré le côté apparemment plus compliqué par l'obligation de noter chaque adresse. Il y a également un gain de place appréciable en MEV.
Un seul impératif pour utiliser à bon escient le pointeur DATA.
Après avoir noté les adresses des sous-parties de DATA qui vous intéressent, il ne faut absolument pas modifier les lignes qui précèdent les DATA sous peine de modifier les adresses et de rendre inopérant tout le travail que vous avez effectué.

La solution est de placer les DATA en tout début de programme. De cette manière vous pouvez modifier la suite sans vous faire de soucis.
Le pointeur ligne en cours d'exécution.
Il se situe en # A8 et # A9 et on obtient le numéro de ligne sur laquelle l'ordinateur passe en faisant PRINT DEEK (# A8).
Cette adresse n'est cependant pas très utile car on ne peut pas la modifier utilement comme le pointeur

programme par exemple. De plus il existe la fonction TRON, donc son utilisation en lecture n'est pas intéressante non plus.

Le pointeur plafond mémoire programme.

Il se situe aux octets #A6 et #A7 et est recopié par l'Oric aux octets #A2 et #A3. Il correspond en fait à la valeur mise dans l'ordinateur par un HIMEM.

Vous pouvez le vérifier en faisant HIMEM XXXXX puis ? DEEK (# A6).

Vous devez alors voir s'afficher XXXXX. Lorsque vous allumez l'ordinateur la valeur initiale est #9FØØ, vous pouvez la modifiant en DOKANT en #A6 la nouvelle valeur qui vous intéresse.

Le pointeur fin de programme basic.

Nous avons vu au début de ce chapitre qu'il existe deux pointeurs dans le système définissant le début et la fin d'un programme. Nous avons étudié le pointeur de début, voyons maintenant celui de fin.

Il se situe aux adresses # 9E et # 9F.

Vous pouvez obtenir l'adresse du dernier octet basic en faisant ?DEEK (#9E).

Ce pointeur n'est pas modifiable utilement mais il peut permettre par exemple de calculer précisément la longueur d'un programme basic.

Pour ce faire tapez :

?DEEK (# 9E)—DEEK (# 9A).

Vous avez alors le nombre d'octets de votre programme.

MAGNETO

Beaucoup d'adresses concernent le magnéto. On peut, en les utilisant correctement faire à peu près ce que l'on veut à l'aide du magnéto.
La première des adresses que nous aborderons est #302 (= 770 en décimal) : cette adresse sert à commander la télécommande c'est-à-dire le relais qui se trouve à l'intérieur de l'ORIC : si vous tapez POKE#302, 7, le magnétophone (s'il est relié à l'ORIC) se met en route. Pour l'arrêter, il faut alors taper POKE#302, 247.
Cette télécommande sert si on veut enregistrer des programmes ou des blocs mémoire par le logiciel. Mais elle peut également servir à bien d'autres choses : couplée avec le TIMER, cette commande peut effectuer un travail à une heure déterminée. Les applications sont donc multiples.
Si vous voulez vous servir de cette commande pour enregistrer, il faut stocker d'autres valeurs dans certaines variables systèmes :
— Vous devez tout d'abord donner l'adresse de départ et l'adresse de fin du bloc mémoire à enregistrer. Ces valeurs doivent être stockées en #5F et #61 par des DOKE puisqu'il s'agit d'adresses qui occupent 2 octets.
Vous faites donc DOKE#5F, adresse départ
DOKE#61, adresse fin.
— Vous devez également indiquer si votre enregistrement doit se faire en 300 bauds ou en 2400

bauds : pour cela, l'adresse est # 67.
Si vous voulez enregister en fast (2400 bands) vous tapez POKE#67,0.
Si vous voulez enregistrer en slow (300 bauds) vous tapez POKE # 67,1.
— Vous devez ensuite indiquer à l'ORIC s'il s'agit d'un programme en Basic ou s'il s'agit d'un programme en langage machine : pour cela, utilisez l'adresse #64.

Si le programme est en Basic, vous devez POKER 0 à cette adresse donc faire POKE#64,0.
Si le programme est en langage machine, POKER n'importe quel nombre différent de 0 à cette adresse.
— Vous devez enfin indiquer si votre programme doit être enregistré en AUTO ou non. Ceci permet entre autre, dans un programme totalement en langage machine de demander le programme à l'adresse de sa première ligne. La cellule concernée par l'AUTO est la cellule #63.

Si votre programme doit être enregistré en AUTO, alors tapez POKE#63,1.
— Si votre programme ne doit pas être enregistré en AUTO alors tapez POKE#63,0.
— Enfin vous pouvez, mais ceci est accessoire, donner un nom à votre programme en plaçant ce nom à partir de l'adresse 53 jusqu'à l'adresse 70 puisque les noms de programmes ne doivent pas dépasser 17 caractères, d'autre part vous devez rajouter un octet nul à la fin du nom que vous donnez au programme.
Il vous suffit ensuite d'enregistrer le programme grâce à la routine se trouvant en #E57B en n'oubliant pas de supprimer la scrutation clavier par la routine placée en #E6CA puis de la remettre ensuite par celle qui se trouve en #E804.

Pour lire un programme, la démarche est la même si ce n'est qu'il faut remplacer la routine d'enregistrement par la routine de lecture qui se trouve en ROM à l'adresse #E4A8.
Bien sûr ces dernières manœuvres doivent être exécutées en mode non direct sous peine de destruction du nom du programme se trouvant à partir de l'adresse 53 : il s'agit en effet de l'adresse de départ du Buffer qui emmagasine donc au fur et à mesure le nom des fonctions exécutées. Vous pouvez par exemple vous servir d'une routine en langage machine placée en début de la page 4.

LE CLAVIER

Composé de 57 touches disposées aux normes anglaises (QWERTY), ce clavier est un périphérique permettant la liaison entre l'unité centrale (le 6502A) et l'utilisateur.
Ces 57 touches utilisées en position normale, avec shift ou CTRL sont à répétition automatique lorsqu'elles sont enfoncées 1 s et ensuite ce délai passe à 0,125 seconde entre 2 répétitions.
De plus, chaque touche émet un Bëep sonore qu'il est possible de supprimer par CTRL F, mais aussi par une autre méthode qui est précisée dans le chapitre des contrôles.
Voyons maintenant quelques astuces concernant le clavier.

1° Fonctionnement de la répétition automatique

Le fonctionnement de la répétition est simple puisqu'il ne s'agit en fait que d'un simple décrémentation de l'octot # 2ØE.
Au départ la valeur #2O se trouve à cette adresse et lorsqu'une touche est appuyée elle commence à décroître.
Si elle arrive à 1 la touche est repérée une fois et la valeur 4 est mise en # 2ØE.
La décrémentation reprend et le processus continue jusqu'à ce que la touche soit relachée.
Lorsqu'elle est relachée la volen # 20 est remise en # 2ØE et l'ordinateur est à nouveau prêt pour recevoir d'autres commandes.

On voit donc que le délai entre deux répétitions est par ce moyen 8 fois plus faible que celui entre le fonctionnnement normal et la première répétition (4 au lieu de # 20).
Vous pouvez vérifier ceci par le petit programme suivant :
0 CLS
1 PLOT 10,10 STR$ (PEEK (# 20E)) + "▫" : GOTO 1 faites RUN et appuyez sur une touche, vous comprendrez alors aisément le processus décrit précédément.

2° Variation de la vitesse de la répétition automatique.

Si l'on a besoin d'une répétition rapide (pour faire par exemple des corrections) on peut trouver qu'un délai d'1 seconde est trop long.
Il est donc intéressant de pouvoir modifier ce délai pour faire varier la vitesse de la répétition.
Ceci est possible en changeant la valeur de la paire d'octets # 306, # 307 qui est au départ de # 2710.
Sachant que plus la valeur mise est faible, plus la répétition est rapide et inversement, on peut la modifier utilement.
De plus la vitesse de la répétition est inversement proportionnelle à celle du CPU.
Si vous augmentez donc la vitesse de l'un, vous diminuez celle de l'autre.
Le seul impératif pour modifier la valeur des 2 octets est de ne pas descendre en-dessous de # 960 car sinon l'ordinateur ne fonctionne plus correctement.
Voici maintenant 2 exemples pour illustrer l'effet des modifications de ces 2 octets :
— variation de la vitesse du CPU
0 CLS : INPUT " VALEUR ", V : DOKE # 306, V
1 FORI = 1 TO 1000 : NEXT.
Faites RUN et répondez la première fois # 960, puis # 2710 et enfin # FFFF. Comparez alors à chaque fois la durée mise pour effectuer la boucle.

— Variation de la vitesse du repeat.
Faites DOKE # 306, # 960.
Appuyez sur une touche.
Faites DOKE # 306, # FFFF.
Appuyez sur une touche vous serez surpris de la différence de temporisation.
On comprend donc l'utilité de cette adresse qui va permettre ou bien l'augmentation de la vitesse du repeat ou bien celle de l'ordinateur.
Voyons maintenant un moyen encore plus efficace pour accélérer le CPU, et ceci en supprimant la scrupation clavier.

Scrutation clavier.

Vous savez sans doute que l'ordinateur effectue en permanence une scrutation du clavier ce qui lui prend beaucoup de temps et ce à l'insu de la vitesse de déroulement du programme.
Il est donc intéressant de pouvoir supprimer cette scrutation lorsque l'on n'a pas besoin d'utiliser le clavier.
C'est possible en faisant appel à une routine de la ROM par CALL # E6CA.
Pour rétablir la scrutation clavier, il suffit de faire CALL # E804 (à ne pas oublier pour rendre la main à l'utilisateur).

Inconvénients :

— Blocage de l'horloge interne : impossibilité d'utiliser la fonction WAIT.
— Déconnection de la télécommande magnétophone.
— Problèmes en haute résolution graphique.

Avantages :

— Blocage du CTRLC (impossibilité d'arrêter le programme autrement que par RESET).
— Gain de vitesse de 30 %.

4° Remplacement de la fonction KEY S

Vous savez sans doute que lorsqu'on utilise la fonction KEY S celle-ci est tributaire de la répétition automatique. C'est-à-dire que l'information sur la

touche appuyée va arriver par à coup à l'ordinateur d'où une perte de temps et un mouvement saccadé s'il s'agit d'un jeu.
Il est possible de remplacer cette fonction par un test sur certaines adresses qui changent de valeur suivant la touche appuyée et ce de façon régulière. Deux adresses sont utilisables, les autres ne seront même pas précisées, car ne présentant pas d'intérêt particulier.

a) Division du clavier en zones verticales.

Imaginez le clavier à l'exception de SHIFT et de CRTL divisé en 14 groupes verticaux numérotés chacun de 0 à 7.
La valeur initiale est #FF.

1	2	3	4	5	6	7	8	9	0	—	=	\	
ESC	Q	W	E	R	T	Y	U	I	O	P	[	]	DEL
A	S	D	F	G	H	J	K	L	;		'		RET.
Z	X	C	V	B	N	M		,	.	/			
←	↓					SPC							

Si une touche est enfoncée le bit correspondant au numéro du groupe dont fait partie la touche est mis à zéro et y reste jusqu'à ce qu'une autre touche soit appuyée.

Il peut donc y avoir 7 valeurs différentes (il n'y a pas de groupe n° 4) qui sont les suivantes :

Touche du groupe	Valeur binaire	Valeur hex.
0	11111110	#FE
1	11111101	#FD
2	11111011	#FB
3	11110111	#F7
5	11011111	#DF
6	10111111	#BF
7	01111111	# 7F

L'adresse qui donne ces valeurs est # 2Ø A.
L'utilité de cette adresse est que la valeur qui y est est celle de la dernière touche appuyée. Ceci étant très pratique pour effectuer des déplacements dans un jeu, puisqu'il ne suffit que d'appuyer une seule fois sur la touche au lieu de laisser le doigt dessus pour que l'action concernée s'effectue.
Voici un petit programme illustrant le fonctionnement du codage par zone.

```
0 CLS
1 B=PEEK(#20A)
5 A=128:FORI=0TO7:C(8-I)=-(B>=A)
6 IFC(8-I)=0THENPLOT12,14,"ZONE"+STR$(7-I)
7 IFB>=ATHENB=B-A
8 A=A/2:NEXT
9 PLOT12,6,HEX$(PEEK(#20A))
10 FORI=8TO1STEP-1:PLOT9+(I*2),1,STR$(C(I))
:NEXT:GOTO1
```

Le seul inconvénient de cette méthode est le fait qu'il s'agit d'un codage par zone, donc un même code pour plusieurs touches.
Il peut être intéressant d'avoir une valeur particulière pour chaque touche enfoncée, c'est ce que nous allons voir maintenant.

b) Division du clavier en zones verticales avec distinction entre chaque touche.

Le clavier est maintenant divisé en 8 zones verticales qui se définissent par le « chiffre » des « dizaines » (nous sommes en base 16 !) de la valeur donnée à chaque touche.
Cette valeur se trouve à l'adresse # 208.
Adresse qui comporte la valeur # 38 lorsqu'aucune touche n'est appuyée qui change de valeur uniquement lorsqu'une touche est enfoncée et qui revient à sa valeur initiale lorsque la touche est relâchée.
Ceci est très utile lorsque l'on veut que l'action définie par la touche ne s'effectue que lorsque celle-ci est enfoncée (par exemple transformation du clavier de l'ORIC en piano).
De plus chaque touche a une valeur différente ce qui permet de donner à des touches de la même zone des fonctions différentes.

Sur la page suivante vous pouvez voir le tableau des valeurs pour chaque touche et le vérifier à l'aide du petit programme suivant :

```
Ø CLS
1 PLOT 1Ø,1Ø HEX# PEEK(#208)) : GOTO 1.
```

On peut comprendre l'utilité de cette adresse qui remplace avantageusement la fonction Key# car si vous voulez tester la touche 2 par exemple il suffit de faire :

```
IF PEEK (#208) = #AA THEN xxxx
```

au lieu de

```
IF KEY # = « Z » THEN xxxx
```

ce qui n'est pas plus long mais qui apporte de la régularité dans les déplacements, les tirs...
Cependant par cette méthode on ne peut toujours pas par exemple déplacer un vaisseau et tirer en même temps.
Nous allons donc voir une autre méthode qui permet d'effectuer jusqu'à 2 actions différentes au même instant !

c) **Codage du Shift et du Control**

Nous avons vu jusqu'à maintenant le codage des touches du clavier excepté SHIFT ET CTRL, ce que nous allons faire maintenant.
Le codage de ces touches se situe à l'OCTET # 209

Le CTRL a pour valeur	∞A2
Le Shift de gauche	#A4
Le Shift de droite	#A7

Lorsqu'aucune de ces 3 touches n'est appuyée la valeur est #38.
Le fonctionnement est donc le mêne qu'en #208, c'est-à-dire que lorsque l'une des 3 touches est appuyée l'Octet change de valeur et revient à la valeur initiale lorsque la touche est relâchée.
Vous pouvez essayer le programme suivant :

```
0 CLS
```

Tableau des valeurs de l'adresse # 208

1 A8	2 B2	3 B8	4 9A	5 90	6 8A	7 80	8 87	9 83	0 97	— 93	= BF	\ B3	
ESC A9	Q B1	W BE	E 9E	R 91	T 89	Y 86	U 85	I 8D	O 95	P 9D	[BD	] B5	Del AD
A AE	S B6	D B9	F 99	G 96	H 8E	J 81	K 83	L 8F	; 93		' BB		Ret. AF
Z AA	X B0	C BA	V 98	B 92	N 88	M 82		, 8C	. 94	/ 9F			
← AC	↓ B4					Space 84				↑ 9C	→ BC		
Zone A	Zone B		Zone 9		Zone 8		Zone D		Zone 3		Zone B		Zone A

1 PLOT 1Ø, 1Ø, HEX $ (PEEK (#209)) : GOTO 1
Faites RUN et vérifiez les valeurs des touches SHIFT et CTRL.
En résumé on voit que d'une part les touches normales et d'autre part les touches SHIFT et CTRL sont indépendantes ce qui permet leur utilisation simultanée pour 2 actions différentes, ce qui était impossible avec KEY #.

d) **Interprétation directe du GET.**

Souvent par une prise de données sur clavier avec un GET, on n'a besoin en fait que du code ASCII. En fait ORIC fait ce travail lui-même et vous évite d'utiliser les fonctions ASC.
La valeur ASCII est placé par l'ORIC en #35.
Il vous suffit pour la récupérer de faire PRINT PEEK (#35).
A noter que si on utilise CTRL ou SHIFT le code est encore placé à la même adresse.
Essayer par exemple le programme suivant :

```
10 GETR$
20 ? PEEK (#35)
30 GOTO 10
```

faites RUN puis tapez sur une touche : vous voyez alors apparaître le code ASCII de la touche appuyée. Si vous appuyer sur CTRL et C en même temps, vous obtenez alors 3 ce qui correspond bien au résultat escompté. De même, si vous êtes en mode minuscule et que vous appuyez sur A, vous obtenez 97 (code ASCII de a) et si vous prenez SHIFT et A, vous obtenez, comme prévu, 65.

NEW

Le NEW est une fonction qui, théoriquement, efface tout le programme se trouvant dans l'ORIC. En fait, il n'en est rien : le NEW ne fait que réinitialiser certaines variables et n'efface en aucun cas ce qu'il y a dans la mémoire. Il suffit donc de rétablir les valeurs initiales de ces variables si vous avez fait un NEW malencontreux.
Pour commencer, initialisez l'ORIC puis tapez le programme suivant :

```
10 FOR I = 0 TO 7
20 PAPER I : WAIT 30
30 NEXT
```

exécutez-le puis tapez NEW, faites LIST, le programme a disparu. Tapez à présent DOKE#501, 1292, faites LIST, le programme est à nouveau là. Cependant, il n'est plus exécutable, nous verrons comment le rendre exécutable plus tard, pour l'instant, analysons le DOKE#501, 1292.
Les adresses #501, #502 contiennent à l'origine l'adresse de fin de première ligne. Lors d'un NEW, cette valeur est remise à 0. En rétablissant la valeur initiale, vous autorisez à nouveau les LIST. Cependant, le programme n'est pas exécutable car lors d'un NEW, l'adresse du dernier octet Basic est remise à 1283. Il faut donc rétablir la valeur initiale de ces variables systèmes. Pour cela, réinitialisez l'ORIC.

Tapez le programme donné plus haut puis NEW.
Tapez ensuite :
DOKE#501, 1292
DOKE#9E, 1311
DOKE#AO, 1311
DOKE#9C, 1311
Vous pouvez à présent réexécuter le programme sans aucun problème, le sauvegarder si vous aviez oublier de le faire.
Ceci est bien beau me direz-vous, mais tous les programmes n'ont pas la même longueur totale ni la même 1re ligne. Il faut donc un moyen pour trouver pratiquement la longueur de la 1re ligne ainsi que l'adresse du dernier octet basic.
Si vous voulez le faire manuellement, il existe une méthode simple :
— Pour trouver l'adresse à DOKER en #501, faites ?PEEK(#505), ?PEEK(#506)... etc jusqu'à ce que vous trouviez un O. Lorsque vous avez trouvé un O, ajouter 1 à l'adresse où se trouve ce O et vous obtenez le nombre à DOKER en #501.
— Pour trouver l'adresse à DOKER en #9E, #9C, #AO, PEEKER la RAM à partir du haut et en descendant. Vous avez normalement des 85(=#55). Au bout d'un moment, vous avez un O, l'adresse à DOKER en#9E, #9C, #AO est alors l'adresse du dernier 85 obtenu. Vous ne pouvez en aucun cas refaire un programme BASIC pour faire ce travail car vous détruiriez ainsi votre ancien programme. Vous pouvez par contre avoir une routine en assembleur quelque part dans la mémoire qui, appelée par un CALL, fasse le travail seule. Vous pouvez, par exemple, stocker ce petit programme à partir de l'adresse #A000 et faire CALL #A000 lorsque vous en avez besoin.

CTRL ET POKE

Il est très désagréable d'avoir à faire à des CHR $ (x) pour effectuer des CTRL en mode programme. En effet, si par exemple vous faites un CHR $ (17) pour éliminer le curseur, que vous arrêtez et que vous le relancez, le curseur va réapparaître puisqu'il s'agit d'une bascule.
Il en est de même pour tous les contrôles.
Il est donc intéressant de trouver un moyen qui permette de simuler les contrôles et ce d'une façon sûre en supprimant l'effet de bascule. Pour ce faire, il existe l'adresse # 26 A ou 618 en décimal.
La valeur initiale de cette adresse (à la mise à froid) est 3.

Si vous faites CTRL Q, puis ?PEEK(618) vous obtenez 2. Ceci signifie donc que la suppression du curseur correspond à une soustraction de 1 dans le registre 618.
De même un CTRL S après une initialisation correspond à une soustraction de 2.
Les CTRL concernés sont :
Q, S, P, F, [,], D.

On peut récapituler les effets obtenus grâce à cette variable système sous forme d'un tableau :

	128	64	32	16	8	4	2	1
0		Simple	38 col.		sonore		suppres. vidéo	pas de curseur
1		Double	40 col.	ESC	muet		vidéo	curseur
	?	D	]	[	F	P	S	Q

l'utilisation de ce tableau est simple : on choisit ce que l'on veut, on additionne alors le poids des colonnes dont le bit doit être à 1 et on a le nombre à POKER en 618.
Comme vous pouvez le constater, le 3 qui se trouve initialement dans la cellule 618 correspond à ceci :

0	0	0	0	0	0	1	1
?	simple auteur	38 colonnes	pas d'ESC	clavier sonore	pas d'effet sur l'imprimante	vidéo active	curseur

Supposons à présent que vous vouliez être en 40 colonnes avec le curseur, la vidéo active et sans le bruit de clavier : on obtient alors :

128	64	32	16	8	4	2	1
0	0	1	0	1	0	1	1
	Simple hauteur	40 colones	pas d'ESC	clavier muet		vidéo active	curseur visible

Le CTRL P à un effet sur l'imprimante que nous ignorons, ce périphérique nous faisant défaut.
Nous n'avons pas trouver l'application du bit de plus fort poids pour l'instant.
A noter que cette cellule ne comprend pas le CTRL T qui se trouve en 524, cellule vue dans le chapitre ECRAN.

ADDENDUM.

Bientôt apparaîtra en France, une deuxième version de la ROM de l'ORIC. Nous avons donc tenu à écrire un livre utilisable pour les deux versions de la ROM. Voici donc un tableau de conversion des adresses des variables systèmes et des adresses ROM.

1re R.O.M. (VI.O) Hexa	Décimal	2e R.O.M. (VI.1) Hexa	Décimal
# 12, # 13	18,19	# 12, # B	18,19
# 1E	28	# 1E	28
# 35	53	# 35	53
# 5F, # 60	95,96	# 2A9, # 2AA	681,682
# 61, # 62	97,98	# 2AB, # 2AC	683,684
# 63	99	# 2AD	685
# 64	100	# 2AE	686
# 67	103	# 24D	589
# 9A, # 9B	154,155	# 9A, # 9B	154,155
# 9E, # 9F	158,159	# 9E, # 9F	158,159
# A2, # A3	162,163	# A2, # A3	162,163
# A6, # A7	166,167	# A6, # A7	166,167
# A8, # A9	168,169	# A8, # A9	168,169
# BØ, # B1	176,177	# BØ, # B1	176,177
# E9, # EA	233,234	# E9, # EA	233,234
# 2Ø8	52Ø	# 2Ø8	52Ø
# 2Ø9	521	# 2Ø9	521
# 2ØA	522	# 2ØA	522
# 2ØC	524	# 2ØC	524

Hexa	Décimal	Hexa	Décimal
# 20E	526	# 20E	526
# 22B	555	# 247	583
# 268	616	# 268	616
# 269	617	# 269	617
# 26A	618	# 26A	618
# 26B	619	# 26B	619
# 26C	620	# 26C	620
# 26D	621	# 27A	634
# 26E	622	# 27B	635
# 26F	623	# 27E	638
# 276, # 277	630,631	# 276, # 277	630 631
# 2C0	704	# 2C0	704
# 302	770	# 302	770
# 306, # 30	774,775	# 306, # 307	774,775
# 501, # 502	1 281, 1 282	# 501, # 502	1 281, 1 282
# E4A8	58 536	# E57D	58749
		# E4AC	58540
# E57B	58 747	# E585	58757
		# E607	58887
		# E62E	58926
# E6CA	59 082	# E76A	59 242
# E804	59 396	# E93D	59 709
# F42D	62 509	# F88F	63 631
# F430	62 512	# F8B2	63 666
# F882	63 562	# F8B8	63 672
# F84A	62 618		Sans équivalent

Adresses R.O.M.

Code opération	Fonction	Adresse
128	END	# C973
129	EDIT	# C692
130	STORE	# E987
131	RECALL	# E9D1
132	TRON	# CD16
133	TROFF	# CD19
134	POP	# CA12
135	PLOT	# DA51
136	PULL	# DAA1
137	LORES	# D9DE
138	DOKE	# D967
139	REPEAT	# DA85
140	UNTIL	# DAA1
141	FOR	# C855
142	LLIST	# C7ED
143	LPRINT	# C809
144	NEXT	# CE98
145	DATA	# CA3C
146	INPUT	# CD55
147	DIM	# D17E
148	CLS	# CCCE
149	READ	# CD89
150	LET	# CB1C
151	GOTO	# C9E5
152	RUN	# C9BD
153	IF	# CA70
154	RESTORE	# C952
155	GOSUB	# C9C8
156	RETURN	# CA12
157	REM	# CA99
158	HIMEM	# EBCE
159	GRAB	# EBE7
160	RELEASE	# EC0C
161	TEXT	# EC21
162	HIRES	# EC33
163	SHOOT	# FAB5

164	EXPLODE	# FACB
165	ZAP	# FAE1
166	PING	# FA9F
167	SOUND	# FB40
168	MUSIC	# FC18
169	PLAY	# FBD0
170	CURSET	# F0C8
171	CURMOV	# F0FD
172	DRAW	# F110
173	CIRCLE	# F37F
174	PATTERN	# F11D
175	FILL	# F268
176	CHAR	# F12D
177	PAPER	# F204
178	INK	# F210
179	STOP	# C971
180	ON	# CAC2
181	WAIT	# D958
182	CLOAD	# E85B
183	CSAVE	# E909
184	DEF	# D4BA
185	POKE	# D94F
186	PRINT	# CBAB
187	CONT	# C9A0
188	LIST	# C748
189	CLEAR	# C70D
190	GET	# CD46
191	CALL	# E946
192	!	# CD13
193	NEW	# C6EE
194	TABC	# DF22
195	TO	# DFBE
196	FN	# DF4A
197	SPCC	#
198	@	# D47F
199	AUTO	# D4A7
200 /	ELSE	# D9B6
201	THEN	#
202	NOT	# E22F

203	STEP	# E350
204	+	# DCB0
205	—	# E2AB
206	*	# E38C
207	/	# E393
208	^ # E3DC	
209	AND	# E440
210	OR	# D939
211	>	# D984
212	=	# DDD5
213	<	# D8A7
214	SGN	# D594
215	INT	# D8D8
216	ABS	# D8B6
217	USR	# D817
218	FRE	# DE78
219	POS	# DF10
220	HEX$	# DF0C
221	&	# DADB
222	SQR	# DA40
223	RND	# EC46
224	LN	# D82B
225	EXP	# D857
226	COS	# D862
227	SIN	#
228	TAN	#
229	ATN	# DB0E
230	PEEK	# EF7C
231	DEEK	#
232	LOG	# DDE7
233	LEN	#
234	STR$	#
235	VAL	# D0E6
236	ASC	# E247
237	CHR$	#
238	PI	# E271
239	TRUE	#
240	FALSE	#

241	KEY$	# D113
242	SCRN	#
243	POINT	# F1C8
244	LEFT$	#
245	RIGHT$	#
246	MID$	#
247	NEXT WITHOUT FOR	
248	SYNTAX	
249	RETURN WITHOUT GOSUB	
250	OUT OF DATA	
251	ILLEGAL QUANTITY	
252	OVER FLOW	
253	OUT OF MEMORY	
254	UNDEF'D STATEMENT	
255	BAD SUBSCRIPT	